POLITECNICO DI TORINO
Repository ISTITUZIONALE

Learning Localized Representations of Point Clouds with Graph-Convolutional Generative Adversarial Networks

Original
Learning Localized Representations of Point Clouds with Graph-Convolutional Generative Adversarial Networks / Valsesia, D.; Fracastoro, G.; Magli, E.. - In: IEEE TRANSACTIONS ON MULTIMEDIA. - ISSN 1520-9210. - 23:(2021), pp. 402-414. [10.1109/TMM.2020.2976627]

Availability:
This version is available at: 11583/2864112 since: 2021-01-20T16:23:40Z

Publisher:
Institute of Electrical and Electronics Engineers Inc.

Published
DOI:10.1109/TMM.2020.2976627

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

28 March 2024

Learning Localized Representations of Point Clouds with Graph-Convolutional Generative Adversarial Networks

Diego Valsesia, *Member, IEEE,* Giulia Fracastoro, *Member, IEEE,* and Enrico Magli, *Fellow, IEEE*

***Abstract*—Point clouds are an important type of geometric data generated by 3D acquisition devices, and have widespread use in computer graphics and vision. However, learning representations for point clouds is particularly challenging due to their nature as being an unordered collection of points irregularly distributed in 3D space. Recently, supervised and semisupervised problems for point clouds leveraged graph convolution, a generalization of the convolution operation for data defined over graphs. This operation has been shown to be very successful at extracting localized features from point clouds. In this paper, we study the unsupervised problem of a *generative* model exploiting graph convolution. Employing graph convolution operations in generative models is not straightforward and it poses some unique challenges. In particular, we focus on the generator of a GAN, where the graph is not known in advance as it is the very output of the generator. We show that the proposed architecture can learn to generate the graph and the features simultaneously. We also study the problem of defining an upsampling layer in the graph-convolutional generator, proposing two methods that respectively learn to exploit a multi-resolution or self-similarity prior to sample the data distribution.**

***Keywords*—Generative Adversarial Networks, Graph convolution, Point clouds**

I. Introduction

Convolutional neural networks (CNNs) are at the core of highly successful models in image generation and understanding. This success is due to the ability of the convolution operation to exploit the principles of locality, stationarity and compositionality that hold true for many data of interest. In particular, convolution allows to extract local features, and weight sharing across the data domain greatly reduces the number of parameters in the model, simplifying training and countering overfitting. However, while images are defined on an underlying regular grid structure, several other data types naturally lie on irregular or non-Euclidean domains [1]. Examples include problems in 3D models [2], [3], keypoints in sketch images [4], computational biology [5], [6] or social network graphs [7]. Being able to define CNN-like architectures is key to exploit useful priors on the data to obtain more powerful representations. In this paper we focus on point clouds [8], [9], [10], which are a set of 3D coordinates representing the geometry of an object or scene. Point clouds are becoming increasingly popular due to their ability to provide a more detailed and immersive representation of the real world, and due to the increased availability of acquisition instruments such as LiDAR scanners or sets of cameras. However, their processing is challenging due to their irregular structure.

Graph convolution is emerging as one of the most successful approaches to deal with data where the irregular domain can be represented as a graph. In this case, the data are defined as vectors on the nodes of a graph. Defining a convolution-like operation for this kind of data is not a trivial task, as even simple notions such as shifts are undefined over graphs. The literature has identified two main approaches to define graph convolution, namely spectral or spatial. In the former case [11], [12], [7], the convolution operator is defined in the spectral domain through the graph Fourier transform [13]. Fast polynomial approximations [12] exist that allow an efficient implementation of this operation. This spectral approach has been successfully used in problems of semi-supervised classification [7] and link prediction [14]. However, the main drawback of these techniques is that the structure of the graph is supposed to be fixed and it is not clear how to handle the case where the graph structure varies. The latter class of methods [15], [16] defines the convolution operator using a spatial approach, where the convolution is performed by local aggregations, i.e., weighted combinations of the vectors restricted to a neighborhood. Since in this case the convolution is defined at a neighborhood level, the operation remains well defined even when the graph structure varies.

Generative models are powerful tools in unsupervised learning that aim at learning the distribution of the data, which can be used to sample new data or regularize inverse problems. Generative Adversarial Networks [17] (GANs) have enjoyed great success in recent years, thanks to their results on the challenging image generation tasks where they have been shown to create novel sharp and realistic images. In particular, they seem to provide better approximations of the data distribution with respect to other methods such as Variational Auto-Encoders [18] (VAEs). Their latent space has also been shown to capture semantic representations of the data [19]. However, so far very little work has been done on generative models for point clouds. In the first work on the topic, Achlioptas et al. [20] studied some GAN architectures to generate point clouds. However, their generator network is composed of densely connected layers and it is therefore unable to generate localized feature representations, which may capture useful

The authors are with Politecnico di Torino – Department of Electronics and Telecommunications, Italy. email: {name.surname}@polito.it. This research has been funded by the Smart-Data@PoliTO center for Big Data and Machine Learning technologies. We thank Nvidia for donating a Quadro P6000 GPU through the GPU Grant Program.

prior knowledge of the data distribution.

In this paper, we study a generative model for point clouds based on graph convolution. In particular, we focus on the generator of a GAN. Our goal is to create localized representations in the hidden layers so that useful data priors can be exploited. In particular, localized features can exploit a compositionality prior in the data, where the representation of the whole can be constructed from the representations of its parts. Also, we study how to exploit two other priors of the point cloud data: i) a multiresolution structure whereby a low-resolution (low number of points) representation can coarsely describe the high-resolution (high number of points) object; ii) a self-similarity structure whereby the representation of a part of the point cloud is similar to the representation of another part. GAN generators are not well explored in the graph convolution literature as they pose a unique challenge: how can one apply a localized operation (the graph convolution) without knowing the domain (the graph) in advance because it is the very output of the generator? We show that the construction presented in this paper learns domain and features simultaneously and promotes the features to be graph embeddings, i.e. representations in a vector latent space of the local dependencies between a point and its neighbors. Furthermore, we address the problem of upsampling at the generator, i.e. increasing the number of points in the hidden layers. While downsampling, in the form of graph coarsening, is a staple in supervised or semi-supervised problems using graph convolution, it is not obvious how to properly upsample the intermediate layers of a graph-convolutional GAN generator. We propose two alternative methods: one that leverages the multiresolution prior, the other the self-similarity prior.

An earlier version of this work first appeared in [21], introducing the concept of a graph-convolutional GAN generator and providing a preliminary set of results. This paper extends [21] by presenting i) a novel point upsampling method (probabilistic upsampling) for the hidden layers of the generator; ii) a detailed analysis of its properties which exploit a different data prior with respect to the other methods; iii) new qualitative and quantitative experimental results, including additional object classes and a deeper analysis of the representations learned by the network; iv) a discussion on complexity issues.

This paper is organized as follows. Section II introduces the notation, some background material on GANs and discusses related works using neural networks for point cloud analysis and generation. Section III presents the proposed architecture for a GAN generator using graph convolution. Section IV expands the proposed architecture by introducing upsampling layers, for which we propose two alternative approaches. Section V suggests an interpretation of the features learned by the proposed architecture in terms of graph embeddings. Section VI provides quantitative and qualitative experiments as well a detailed analysis of the properties of the features learned by the proposed generator. Finally, Section VII draws some conclusions.

II. Background

A. Notation and Definitions

We denote vectors and matrices by lowercase and uppercase boldface characters, respectively.

The notation $\mathbf{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ means that the random vector $\mathbf{a}$ is Gaussian distributed, its mean is $\boldsymbol{\mu}$, and its covariance matrix is $\boldsymbol{\Sigma}$.

B. Generative Adversarial Networks

GANs [17] are state-of-the-art generative models that learn the distribution of training data and allow to draw new samples from it. The key insight of GANs is treating the training process as a game between two neural networks: a generator G, whose goal is to generate realistic samples reproducing the data distribution, and a discriminator D, whose goal is recognize the fake samples from real ones. The generator learns a function that maps a random vector $\mathbf{z}$ in a latent space to a sample $\mathbf{x}$ from the data distribution. In the original formulation, the discriminator worked as a traditional binary classifier trained to separate real samples from generated ones. This formulation was discovered [22] to be minimizing the Jensen-Shannon divergence between the true data distribution and the distribution of the generated data. However, it suffered from mode collapse and training instability issues, where the generator would get stuck on a mode of the distribution and always generate the same samples or would not converge at all. Recently, the Wasserstein GAN [23] addressed such issues by modifying the loss function to be a dual formulation of an optimal transport problem using the Wasserstein metric. This formulation requires constraining the discriminator to have a bounded Lipschitz constant and the optimal G and D can be obtained by solving the following optimization problem:

$$\min_{G} \max_{\|D\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[D(\mathbf{x}) \right] - \mathbb{E}_{\mathbf{z} \sim p_z} \left[D(G(\mathbf{z})) \right], \quad (1)$$

being p_{data} the distribution of training data and p_z a prior distribution on the latent vectors, typically uniform or spherical Gaussian. In this paper, we use the gradient penalty method [24] to enforce the Lipschitz constraint at the discriminator.

Extensive literature is available on using GANs to generate natural images [25], [26], [27], showing the ability to produce sharper and more realistic images when compared with other methods, such as Variational Auto-Encoders (VAEs) [28]. The adversarial training principle, i.e., using a game between two networks to approximate a distribution, has also been used to augment the training objective of several problems such as superresolution [29], inpainting [30], regularization of inverse problems [31], and many more [32], [33], [34], [35], yielding improved results.

C. Related work

Point clouds provide a challenge for traditional convolutional models due to the irregular positioning of the points and to being an unordered set of points. This means that any permutation of the order of its members, while changing the representation, does not change its semantic meaning.

Earlier work on point cloud data has been typically focused on supervised problems such as classification and segmentation of point clouds, while little work has been done on generative models. Classification and segmentation has been addressed using three approaches: voxelization, permutation-invariant networks, and graph convolution. Voxelization [36], [37] is based on the idea of approximating the irregular point structure with a regular 3D grid. Instead, networks like PointNet [38], [39] address the problem of permutation invariance by processing each point identically and independently before applying a globally symmetric operation such as average or max pooling. The most recent approaches [15], [16] build graphs in the Euclidean space of the point cloud and use graph convolution operations. This has shown multiple advantages in i) reducing the degrees of freedom in the learned models by enforcing some kind of weight sharing, and ii) extracting localized features that successfully capture dependencies among neighboring points. Notice that the PointNet++ model [39] also proposes to use localized operations, while not fitting the definition of graph convolution.

This paper, however, studies a *generative model* for point clouds, which has unique challenges and has been less studied in the literature. Some approaches use VAEs: Fan et al. [40] generate point clouds conditioned on an input image; Nash and Williams [41] use object segmentation labels to generate point clouds by parts; Litany et al. [42] focus on generating vertices on meshes with a fixed and given topology. The work closest to this paper is the one by Achiloptas et al. [20] where a GAN to generate point clouds is studied for the first time. However, this GAN has a generator composed of fully connected layers, which is unable to generate interpretable localized features for the points and is not able to exploit the powerful priors for point cloud data leveraged by the convolution operation.

Finally, it is worth mentioning a different class of generative problems, i.e., the ones involving generation of graphs [43], [44] where the objective is to learn the distribution of the adjacency matrix of a class of graphs. Generating point clouds is different due to the fact that each vertex has a signal composed of the x,y,z coordinates (and optionally color), so that it is not enough to generate an adjancency matrix to actually generate the point cloud.

III. Proposed Graph-convolutional GAN

In this section, we describe the proposed generative model. GANs have provided outstanding results for image generation and they have shown that they can approximate the data distribution better than other generative models such as VAEs [18]. For this reason, designing a GAN to generate point clouds can be of high interest.

As explained in the previous section, a GAN is composed by a generator and a discriminator network. The focus of this paper is on the generator network, where we propose a new generator based on graph convolution operations. This can provide several advantages, because it allows us to learn localized features and also reduce the number of parameters by exploiting weight sharing. The work in [20] already presents a GAN to generate point clouds, but in this case it uses a fully-connected generator network. Therefore, it is unable to provide any localized interpretation of its hidden layers.

Introducing graph convolution operations at the generator is not straightforward since the graph is not known in advance but it should be an output of the network. For this reason, the definition of the architecture of a graph-based generator is a very interesting and challenging problem. Instead, at the discriminator the graph is known in advance, because in this case the input of the network is a point cloud. However, since point clouds are unordered set of points, we need to define an architecture that employs permutation invariant operations. So, the discriminator can use one of the architectures that have been developed for supervised problems on point clouds, such as [38], [39], [15], [16].

In the following, we first present the definition of the graph convolution operation employed in this work at the generator, then we describe the architecture of the proposed graph-convolutional generator.

A. Edge-Conditioned Convolution

In the proposed graph-based generator, we use the Edge-Conditioned Convolution presented in [15] which falls under the category of spatial approaches to graph convolution. This operation exploits edge labels in order to perform weighted local aggregations. Using this operation, we can define filters whose weights are conditioned on edge labels and are dynamically generated for each input. This allows us to use the same filter on different graphs and it represents a strong advantage with respect to other definitions of the graph convolution operation, where it is not possible to deal with multiple graphs.

Let us consider a layer l with N^l feature vectors of dimensionality d^l and the corresponding graph $\mathcal{G}^l(\mathcal{V}^l, \mathcal{E}^l)$ where $\mathcal{V}^l$ is the set of vertices with $|\mathcal{V}^l| = N^l$ and $\mathcal{E} \subseteq \mathcal{V}^l \times \mathcal{V}^l$ is the set of edges. We assume that the edges of the graph are labeled, i.e. there exists a function $\mathcal{L} : \mathcal{E} \rightarrow \mathbb{R}^s$ that assigns a label to each edge. The convolution performs, for each node i of the graph $\mathcal{G}^l$, a weighted local aggregation of the feature vectors $\mathbf{H}^l_j \in \mathbb{R}^{d^l}$ on the neighboring nodes $j \in \mathcal{N}^l_i$, where $\mathcal{N}^l_i$ is the neighborhood of node i. The weights of the local aggregation are defined by a fully-connected network $F^l : \mathbb{R}^{d^l} \rightarrow \mathbb{R}^{d^l \times d^{l+1}}$, which takes as input the edge labels and outputs the corresponding weight matrix $\mathbf{\Theta}^{l,ji} = F^l_{\mathbf{w}^l}(\mathcal{L}(i,j)) \in \mathbb{R}^{d^l \times d^{l+1}}$. In the following, we define the edge labeling function as the difference between the features of the two nodes of the edge, i.e. $\mathcal{L}(i,j) = \mathbf{H}^l_j - \mathbf{H}^l_i$. Hence, the convolution operation is defined as:

$$\begin{aligned}\mathbf{H}^{l+1}_i &= \sigma\left(\sum_{j\in\mathcal{N}^l_i} \frac{F^l_{\mathbf{w}^l}\left(\mathbf{H}^l_j - \mathbf{H}^l_i\right)\mathbf{H}^l_j}{|\mathcal{N}^l_i|} + \mathbf{H}^l_i\mathbf{W}^l + \mathbf{b}^l\right) \\ &= \sigma\left(\underbrace{\sum_{j\in\mathcal{N}^l_i} \frac{\mathbf{\Theta}^{l,ji}\mathbf{H}^l_j}{|\mathcal{N}^l_i|}}_{\text{neighborhood}} + \underbrace{\mathbf{H}^l_i\mathbf{W}^l}_{\text{node}} + \mathbf{b}^l\right),\end{aligned} \tag{2}$$

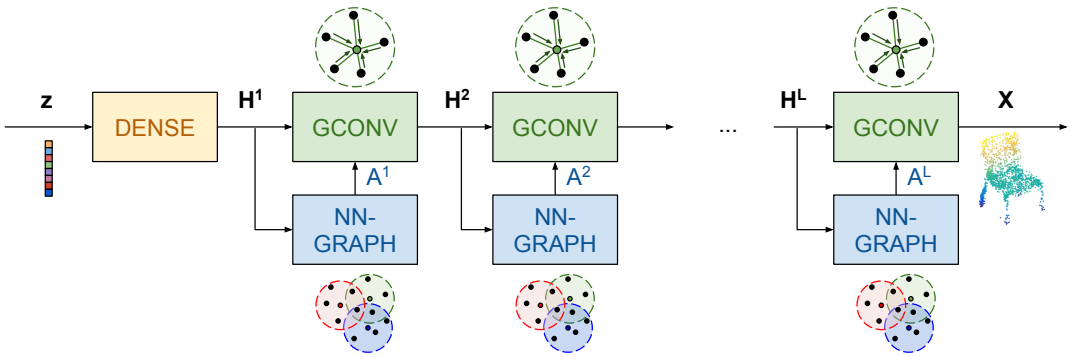


Fig. 1: Graph-convolutional generator without upsampling. The feature matrices $\mathbf{H}^l$ are $N \times d^l$, being N the number of points in the point cloud and d^l the number of features at layer l. The NN-GRAPH block at layer l computes the adjacency matrix $\mathbf{A}^l$ of a k-nn graph using ℓ_2 distances between feature vectors.

where $\mathbf{w}^l$ are the weights parameterizing network F^l, $\mathbf{W}^l \in \mathbb{R}^{d^l \times d^{l+1}}$ is a linear transformation of the node itself, $\mathbf{b}^l$ a bias, and σ a non-linearity. It is worth noting that the filter weights $\mathbf{\Theta}^{l,ij}$ depend only on the difference between the features of the two nodes. This means that two pairs of nodes that have the same difference will have the same weight $\mathbf{\Theta}^{l,ij}$, even if they are in two different regions of the space. This behaviour produces weight sharing as in the classical CNNs and results in a lower number of degrees of freedom. It is also important to note that the standard convolution operation is a special case of the Edge-Conditioned Convolution, where the edge labels are encoded as one-hot vectors [15]. In addition, if the edge labels are defined as function of the node features (as for example in (2)), the Edge-Conditioned Convolution can be seen as a data-dependent convolution.

B. Graph-based generator

The main focus of this paper is to design a generator for a GAN that is able to use localized operations in the form of graph convolutions. As seen in the previous section, these operations are able to deal with data that lie on irregular domains, such as point clouds. However, introducing graph convolution operations at the generator raises some issues specifically related to generative problems. In particular, the main problem to overcome is that, differently from supervised problems [15], [16] or unsupervised settings involving autoencoders [45], the intermediate layers of the GAN generator do not know the point cloud in advance as it is the very result of the generation operation. It is therefore not obvious how to define an operation that is localized to neighborhoods of a graph that is not known in advance. We propose to solve this issue by exploiting the pairwise distances ($\|\mathbf{H}_j^{l-1} - \mathbf{H}_i^{l-1}\|$) between node features of the preceding layer to build a k-nearest neighbor graph. Fig. 1 shows a block diagram of a graph-based generator where each graph convolution block uses the graph constructed from the input features of the block itself. The intuition behind this solution is that this architecture promotes the features to become graph embeddings, i.e., representations in a high-dimensional metric space of relationships between points. Going through the generator network from the latent space towards the point cloud output, these embeddings are assembled hierarchically and their associated graphs are better and better approximations of the graph of the output point cloud.

IV. Upsampling

The previous section presented the basic outline of a graph-based generator in a GAN. However, one evident shortcoming is the fixed number of points throughout the generator, which is determined by the number of output points. This leads to a network with a high number of parameters, which can easily cause overfitting. Moreover, many data of interest typically display some kind of regularity in the form of multi-resolution or other kinds of compositionality whereby points can be predicted by a smaller number of neighboring points. In the case of 2D images, lower resolutions provide a prediction of higher resolutions by supplying the low-frequency content. However, image pixels are aligned on a 2D grid and it is straightforward to upsample by adding new pixels in the positions defined by the grid before filling the high-frequency content. In fact, convolutional GANs for image generation are composed of a sequence of upsampling and convolutional layers. Extending upsampling to deal with the generation of sets of points without a total ordering is not a trivial task. Many works have addressed the problem of upsampling 3D point clouds, e.g., by creating grids in the 3D space [46]. Notice, however, that introducing upsampling to interleave the graph-convolutional layers outlined in the previous section is a more complex problem because the high dimensionality of the feature vectors makes the gridding approach unfeasible.

If we consider the l-th layer of the generator, we want to define an upsampling operation U that, starting from the output of the graph convolution $\mathbf{H}^l \in \mathbb{R}^{N^l \times d^l}$, generates N^l new feature vectors $\tilde{\mathbf{H}}^l \in \mathbb{R}^{N^l \times d^l}$. The upsampling operation U can be defined as

$$U : \mathbb{R}^{N^l \times d^l} \rightarrow \mathbb{R}^{N^l \times d^l},$$
$$\tilde{\mathbf{H}}^l = U(\mathbf{H}^l).$$

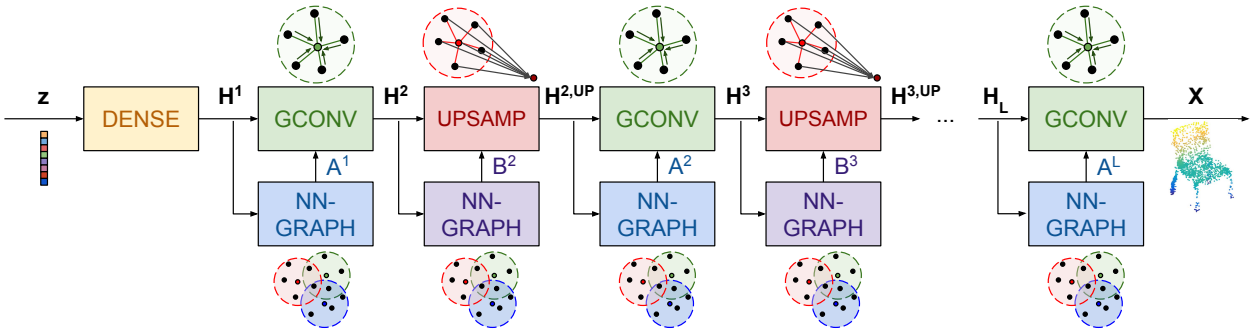


Fig. 2: Graph-convolutional generator with upsampling. The feature matrices $\mathbf{H}^l$ have size $N^l \times d^l$ being N^l the number of points at layer l and d^l the number of features at layer l. The NN-GRAPH block computes the adjacency matrix of a k-nn graph using ℓ_2 distances between feature vectors. At layer l, $\mathbf{A}^l$ and $\mathbf{B}^l$ are the adjacency matrices computed for the GCONV and UPSAMP blocks, respectively. The UPSAMP block computes $N^{l+1} - N^l$ new points and concatenates them to the input ones.

Then, in order to obtain the output $\mathbf{H}^{l,\text{up}} \in \mathbb{R}^{2N^l \times d^l}$ these new feature vectors are concatenated to $\mathbf{H}^l$ as follows

$$\mathbf{H}^{l,\text{up}} = \begin{bmatrix} \mathbf{H}^l \\ \tilde{\mathbf{H}}^l \end{bmatrix} \in \mathbb{R}^{2N^l \times d^l}.$$

In the following, we propose two approaches to define such upsampling operator. Both these methods are localized, i.e. a new point is generated by exploiting only the information about its generating point and the corresponding neighborhood. For this reason, the upsampling operation requires the knowledge of the graph of the input data. To define this graph we exploit the pairwise distances between the input node features, as done for the graph convolution operation. Fig. 2 shows a block diagram of a graph-based generator with upsampling.

A. *Probabilistic approach*

The first upsampling approach that we analyze is based on creating a hierarchy of latent variables, i.e., random vectors $\mathbf{z}^l$ that are used as the input of the upsampling layers. This method generalizes the architecture without upsampling where the only latent variable is $\mathbf{z}$, used as the input to the network. We will refer to this approach as "probabilistic upsampling". The intuition behind this model is that the hierarchy of variables should exploit a multiresolution prior on the data, i.e., using a representation with a lower number of points as a predictor for the representation with a higher number of points, in such a way that each latent variable used in the upsampling operations controls finer details of the generated point cloud.

More in detail, for each point i represented by its feature vector $\mathbf{H}_i^l$ at layer l, we select a set of neighbors and use them to fit a Gaussian distribution of the feature vectors belonging to that neighborhood. Then, we obtain a new feature vector by sampling from this distribution. In particular, given the i-th feature vector $\mathbf{H}_i^l \in \mathbb{R}^{d^l}$ and its corresponding neighborhood

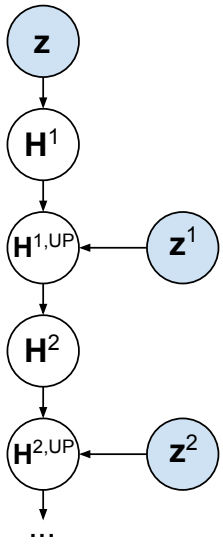


Fig. 3: Probabilistic graphical model of generator with probabilistic upsampling. A hierarchy of latent variables controls finer and finer details of the generator output.

$\mathcal{N}_i^l$, we sample the new feature vector $\tilde{\mathbf{H}}_i^l \in \mathbb{R}^{d_l}$ as

$$\begin{aligned} \tilde{\mathbf{H}}_i^l &\sim \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\Sigma}^{(i)}) \\ \boldsymbol{\mu}^{(i)} &= \frac{1}{|\mathcal{N}_i^l|} \sum_{j \in \mathcal{N}_i^l} \mathbf{H}_j^l \\ \boldsymbol{\Sigma}^{(i)} &= \text{diag}\left(\boldsymbol{\Sigma}_{uu}^{(i)}\right), \boldsymbol{\Sigma}_{uu}^{(i)} = \frac{1}{|\mathcal{N}_i^l|} \sum_{j \in \mathcal{N}_i^l} (\mathbf{H}_{ju}^l - \boldsymbol{\mu}_u^{(i)})^2 \end{aligned} \tag{3}$$

Fig. 3 shows the probabilistic graphical model of the generator when the proposed probabilistic approach is used at the generator. Notice that sampling $\tilde{\mathbf{H}}_i^l$ from the distribution defined in Eq. (3) is equivalent to sampling a latent variable from a standard Normal distribution, $\mathbf{z}^{l,(i)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and then

transforming it by $\tilde{\mathbf{H}}_i^l = \boldsymbol{\mu}^{(i)} + \boldsymbol{\Sigma}^{(i)\,\frac{1}{2}} \mathbf{z}^{l,(i)}$. Using a diagonal covariance is a way to promote features to be uncorrelated as far as the generation of a new point is concerned. Finally, notice that this approach to upsampling does not require extra parameters to be trained and acts as a static generative model of neighborhoods in the hidden layers.

B. Local aggregation approach

Another possible approach to define the upsampling operation is using local aggregations. In this case, the upsampling operation becomes similar to a graph convolution. Given a feature vector $\mathbf{H}_i^l \in \mathbb{R}^{d^l}$, we consider a set of neighbors $\mathcal{N}_i^l$ and we define the new feature vector $\tilde{\mathbf{H}}_i^l \in \mathbb{R}^{d_l}$ as follows

$$\begin{aligned}\tilde{\mathbf{H}}_i^l &= \sigma\left(\sum_{j\in\mathcal{N}_i^l} \frac{\operatorname{diag}\left(F_{\tilde{\mathbf{w}}^l}^{up,l}\left(\mathbf{H}_j^l - \mathbf{H}_i^l\right)\right)\mathbf{H}_j^l}{|\mathcal{N}_i^l|} + \mathbf{H}_i^l\boldsymbol{\Gamma}^l + \mathbf{b}^l\right)\\ &= \sigma\left(\sum_{j\in\mathcal{N}_i^l} \frac{\boldsymbol{\Gamma}^{l,ji}\mathbf{H}_j^l}{|\mathcal{N}_i^l|} + \mathbf{H}_i^l\boldsymbol{\Gamma}^l + \mathbf{b}^l\right)\end{aligned}$$

where $\boldsymbol{\Gamma}^l$ is a diagonal matrix and $F^{up,l} : \mathbb{R}^{d^l} \to \mathbb{R}^{d^l}$ is a fully-connected network which given the difference between $\mathbf{H}_i^l$ and $\mathbf{H}_j^l$ outputs the weight vector $\boldsymbol{\gamma}^{l,ij} \in \mathbb{R}^{d^l}$, which is used to create the diagonal matrix $\boldsymbol{\Gamma}^{l,ji} = \operatorname{diag}\left(\boldsymbol{\gamma}^{l,ji}\right)$.

A key difference from the graph convolution described in Eq. (2) is that where $\boldsymbol{\Theta}^{l,ij}$, and $\mathbf{W}^l$ were dense matrices, now $\boldsymbol{\Gamma}^{l,ji}$ and $\boldsymbol{\Gamma}^l$ are diagonal matrices. This means that during the upsampling operation the local aggregation treats each feature independently. The intuition behind this definition of upsampling is that we want to generate a new point in the same space of the original points (hence not mixing the features) with scaling and translation operations. Notice that if we are generating two points starting from two root points sharing the same neighborhood, the operation will in some way add a "residual" to the root point that depends on the neighborhood and since both points share the neighborhood this residual will be similar, thus approximately preserving the structure of the neighborhood in the position in the space. This allows to exploit a self-similarity prior in the data.

Finally, it is important to note that, in contrast with the previously described probabilistic approach, this upsampling technique requires extra parameters to be trained. In particular, $\tilde{\mathbf{w}}^l$ and $\mathbf{b}^l$ and $\boldsymbol{\Gamma}^l$ are all trainable model parameters. The choice of using diagonal matrices is therefore also motivated by reasons of complexity and in order to avoid overparameterization of the upsampling layer.

V. Graph embedding interpretation

Graph embeddings [47] are representations of graphs in a multidimensional vector space where a feature vector is associated to each node of the graph. For what concerns this paper we consider the following definition of graph embedding, focused on predicting edges from feature vectors.

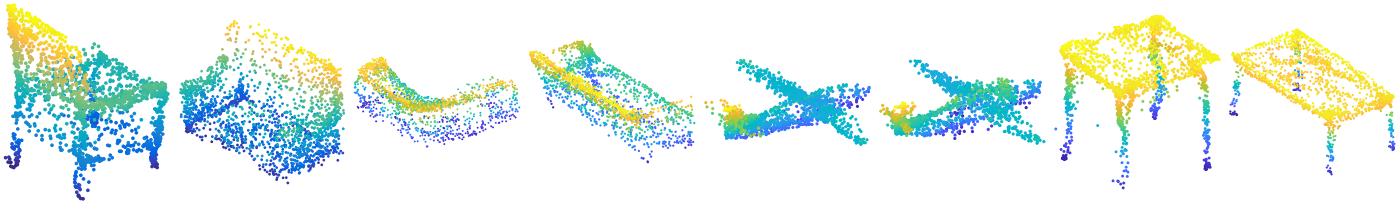
Definition 1. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a graph embedding is a mapping $f : i \to \mathbf{h}_i \in \mathbb{R}^d$, $\forall i \in \mathcal{V}$, such that $d \ll |\mathcal{V}|$ and the function f is defined such that if we consider two pairs of nodes (i, j) and (i, k) where $(i, j) \in \mathcal{E}$ and $(i, k) \notin \mathcal{E}$ then $\|\mathbf{h}_i - \mathbf{h}_j\| < \|\mathbf{h}_i - \mathbf{h}_k\|$.*

No upsampling		**Upsampling**	
Layer	**Output size**	**Layer**	**Output size**
Latent	1×128	Latent	1×128
dense	2048×32	dense	128×96
gconv_0	2048×32	gconv_0	128×48
		upsamp_0	256×48
gconv_1	2048×24	gconv_1	256×32
		upsamp_1	512×32
gconv_2	2048×16	gconv_2	512×16
		upsamp_2	1024×16
gconv_3	2048×8	gconv_3	1024×8
		upsamp_3	2048×8
gconv_4	2048×3	gconv_4	2048×3

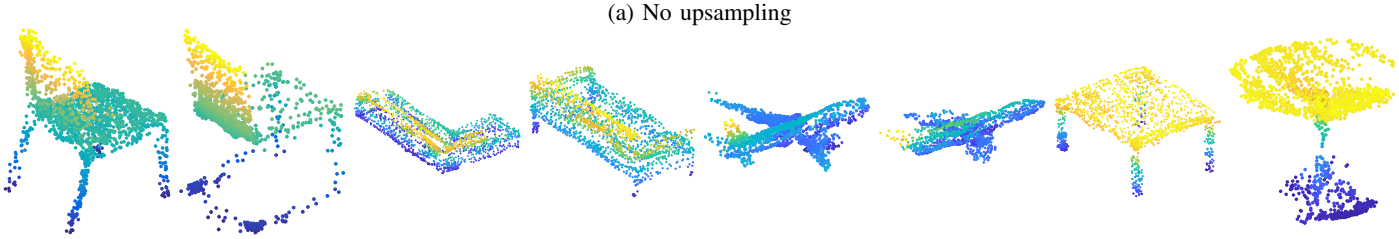
TABLE I: Generator architecture

The graph-convolutional generator presented in this paper can be interpreted as generating graph embeddings of the nearest-neighbor graph of the output point cloud at each hidden layer, thus creating features that are able to capture some properties of the local topology. In order to see why this is the case, we analyze the architecture in Fig. 1 backwards from the output to the input. The final output $\mathbf{x}$ is the result of a graph convolution aggregating features localized to the nearest-neighbor graph computed from the features of the preceding layer. Since the GAN objective is to match the distribution of the output with that of real data, the neighborhoods identified by the last graph must be a good approximation of the neighborhoods in the true data. Therefore, we say that features $\mathbf{H}^L$ are a graph embedding in the sense that they allow to predict the edges of the output graph from their pairwise distances. Proceeding backwards, there is a hierarchy of graph embeddings as the other graphs are constructed from higher-order features.

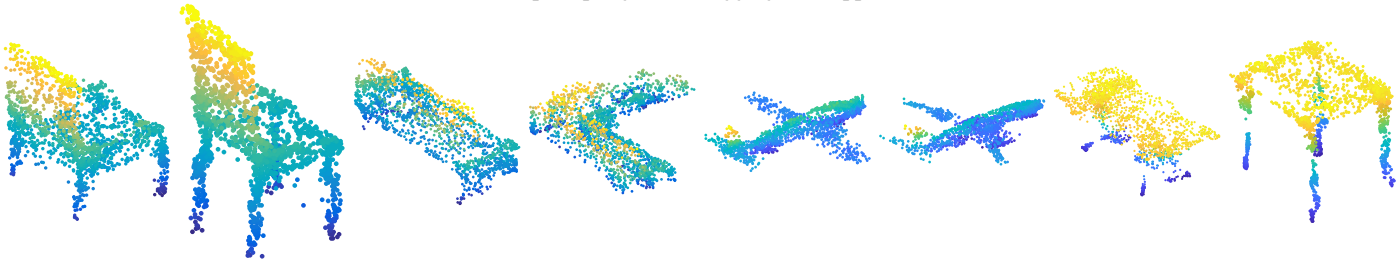
Notice that the upsampling operation in the architecture of Fig. 2 affects this chain of embeddings by introducing new points. While the graph convolution operation promotes the features of all the points after upsampling to be graph embeddings, the upsampling operation affects which points are generated. In the experiments we show that the points generated with the probabilistic method are approximately uniformly distributed over the output point cloud and that the upsampling operation is localized, i.e. the new points fall in the same neighborhood of the generating point. This suggests a generation mechanism exploiting a hierarchy of multiple resolution with new points filling in the details. On the other hand, the aggregation method approximately maintains the neighborhood shape but copies it elsewhere in the point cloud. This suggests a generation mechanism exploiting self-similarities between the features of the point cloud at different locations.



(a) No upsampling



(b) Upsampling - local aggregation approach



(c) Upsampling - probabilistic approach

Fig. 4: Generated point clouds.

No upsampling			**Upsampling**				
				Prob. upsampling		**Aggr. upsampling**	
Layer	**Size 1**	**Size 2**	**Layer**	**Size 1**	**Size 2**	**Size 1**	**Size 2**
gconv_0	512	1024	gconv_0	1024	4608	1024	4608
			upsamp_0	-	-	48	48
gconv_1	256	768	gconv_1	768	1536	768	1536
			upsamp_1	-	-	32	32
gconv_2	128	384	gconv_2	256	512	256	512
			upsamp_2	-	-	16	16
gconv_3	64	128	gconv_3	64	128	64	128
			upsamp_3	-	-	8	8
gconv_4	16	24	gconv_4	16	24	16	24

TABLE II: Architecture of networks F^l and $F^{up,l}$

VI. Experiments

We tested the proposed architecture by using four object classes taken from the ShapeNet repository [48]: "chair", "airplane", "table" and "sofa". The point clouds are obtained by sampling 2048 uniformly distributed points for each 3D model of the considered classes. The points clouds in each class are split as 85% training data, 5% testing and 10% validation, resulting in at least 2500 training point clouds per class. A class-specific model is trained for the desired class of point clouds. Since the focus of this paper is on the features learned by the generator, the architecture for the discriminator is the same as that of the r-GAN in [20], with 4 layers with weights shared across points (number of output features: 64, 128, 256, 512) followed by a global maxpool and by 3 dense layers. The generator architecture is reported in Table I. The fully-connected networks F^l and $F^{up,l}$ are composed by 2 dense layers, and their architecture is described in Table II. The graph is built by selecting the 20 nearest neighbors in terms of Euclidean distance in the feature space. This value has been cross-validated as a good tradeoff between visual quality and computational complexity. We use Leaky ReLUs as nonlinearities and RMSProp as optimization method with a learning rate equal to 10^{-4} for both generator and discriminator. Batch normalization follows every graph convolution. The batch size is 50. The gradient penalty parameter of the WGAN is 1 and the discriminator is optimized for 5 iterations for each generator step. The models have been trained for 1000 epochs. For the "chair" class this required about 5 days without upsampling and 4 days with upsampling. Code and pretrained models are available online[1].

A. Assessment of generated point clouds

In this section we perform qualitative and quantitative comparisons with the generated point clouds.

Visual results: We first visually inspect the generated point clouds for the four classes, as shown in Fig. 4. The results

[1]https://github.com/diegovalsesia/GraphCNN-GAN

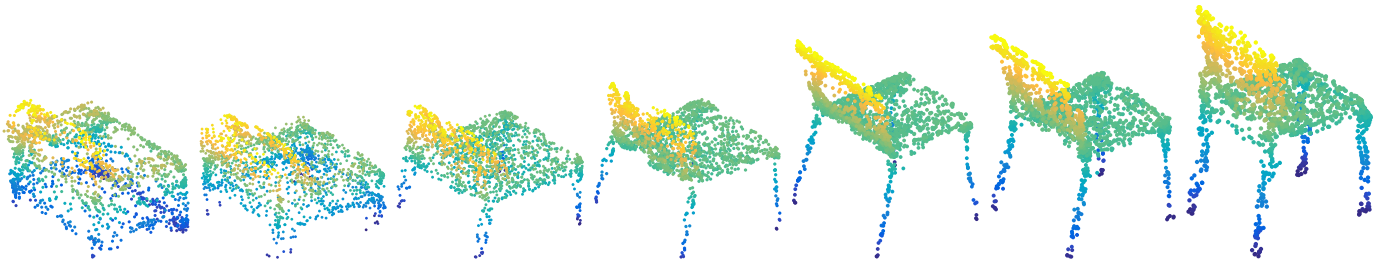


Fig. 5: Latent space interpolation (model with aggregation upsampling).

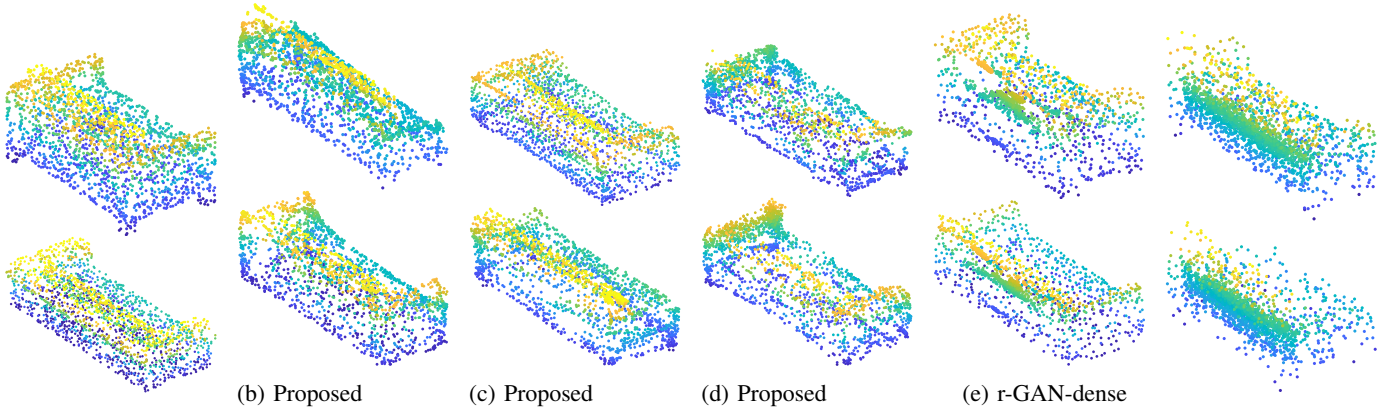
TABLE III: Quantitative comparisons

Class	Model	JSD	MMD-CD	MMD-EMD	COV-CD	COV-EMD
Chair	**r-GAN-dense**	0.238	**0.0029**	0.136	33	13
	r-GAN-conv	0.517	0.0030	0.223	23	4
	Proposed (no up.)	0.119	0.0033	0.104	26	20
	Proposed (aggr. up.)	**0.100**	**0.0029**	**0.097**	30	26
	Proposed (prob. up.)	0.104	0.0034	0.106	**39**	**31**
Sofa	**r-GAN-dense**	0.221	**0.0020**	0.146	32	12
	r-GAN-conv	0.293	0.0025	0.110	21	12
	Proposed (no up.)	0.095	0.0024	0.094	25	19
	Proposed (aggr. up.)	**0.063**	**0.0020**	**0.083**	**39**	**24**
	Proposed (prob. up.)	0.119	0.0022	0.113	32	19
Airplane	**r-GAN-dense**	0.182	0.0009	0.094	31	9
	r-GAN-conv	0.350	**0.0008**	0.101	26	7
	Proposed (no up.)	0.164	0.0010	0.102	24	13
	Proposed (aggr. up.)	**0.083**	**0.0008**	**0.071**	31	14
	Proposed (prob. up.)	0.095	0.0010	0.075	**34**	**19**
Table	**r-GAN-dense**	0.217	**0.0031**	0.139	33	15
	r-GAN-conv	0.359	**0.0031**	0.247	29	4
	Proposed (no up.)	0.171	0.0045	**0.123**	24	18
	Proposed (aggr. up.)	**0.148**	0.0035	0.131	**36**	29
	Proposed (prob. up.)	0.167	0.0037	0.124	33	**34**

are convincing from a visual standpoint and the variety of the generated objects is high, suggesting no mode collapse in the training process. The distribution of points on the object is quite uniform, especially for the method with aggregation upsampling. In order to evaluate the latent space representation, we perform a linear interpolation between two different points in the latent space and observe the corresponding point clouds output by the generator. Fig. 5 shows that interpolating in the latent space provides a smooth semantic transition between two endpoints. In this case we can see a short armchair progressively become a tall chair.

Comparisons with state of the art: To the best of our knowledge, this is the first work addressing GANs for point clouds learning localized features. We compare the proposed GAN for point cloud generation with other GANs able to deal with unordered sets of points. In particular, the "r-GAN-dense" architecture in [20] has a dense generator, which is unable to generate localized representations because there is no mapping between points and feature vectors. As an additional baseline variant, dubbed "r-GAN-conv", we study the use of a generator having as many feature vectors as the points in the point cloud and using a size-1 convolution across the points. Notice that the employed graph convolution can be seen as a generalization of this model, aggregating the features of neighboring points instead of processing each point independently. We point out that we cannot compare the proposed method in a fair way with the variational autoencoders mentioned in Sec. II-C: indeed, [40] generates point clouds conditioned on an input image; [41] uses object segmentation labels to generate point clouds by parts; and [42] focuses on generating vertices on meshes with a fixed and given topology.

In order to perform a quantitative evaluation of the generated point clouds we use the evaluation metrics proposed in [20], employing three different metrics to compare a set of generated samples with the test set. The first one is the Jensen-Shannon divergence (JSD) between marginal distributions defined in the 3D space. Then, we also evaluate the coverage (COV) and the minimum matching distance (MMD), as defined in [20].



(a) Original sample (b) Proposed (No upsampling) (c) Proposed (Aggr. upsampling) (d) Proposed (Prob. upsampling) (e) r-GAN-dense [20] (f) r-GAN-conv

Fig. 6: Generated point clouds from different methods.

The coverage metric measures the fraction of generated point clouds that were matched to point clouds in the test set. The closeness between two point clouds can be measured using two different point-set distances, the earth mover's distance (EMD) and the Chamfer distance (CD), thus yielding two different metrics, COV-EMD and COV-CD. A high coverage score means that the generated point clouds can represent most of the test set. However, this metric does not evaluate how well the covered examples are represented by the generated point clouds. For this reason, [20] also introduces the minimum matching distance metric that measures the fidelity of the generated point clouds with respect to the test set. This metric measures the average minimum matching distance between the generated point clouds and the test set. Also in this case, either point-set distances can be used obtaining two different metrics, MMD-EMD and MMD-CD. Table III shows the obtained results. As can be seen, the proposed methods achieve better values for the metrics under consideration. In particular, the method with aggregation upsampling shows the best overall performance. However, it is interesting to observe that the method with probabilistic upsampling shows a high coverage score, often outperforming all the other methods. Notice also that [20] reports that the Chamfer distance is often unreliable as it fails to penalize non-uniform distributions of points. Fig. 6 visually shows two samples from the test set and two point clouds generated by each method. We can see that the proposed method has better-distributed points, confirming the quantitative results. In particular, the r-GAN-dense shows clusters of points, while the r-GAN-conv also exhibits noisy shapes.

Graph embedding and feature radius: With reference to Table I, the output of each layer is a matrix where every point is associated to a vector of features. In Sec. V we claimed that these features learned by the generator are graph embeddings. We tested this hypothesis by measuring how much the adjacency matrix of the final point cloud, constructed as a nearest-neighbor graph in 3D, is successfully predicted by the nearest-neighbor adjacency matrix computed from hidden features. This is shown in Fig. 7 which reports the percentage of edges correctly predicted as function of the number of neighbors considered for the graph of the output point cloud and a fixed number of 20 neighbors in the feature space. Notice that layers closer to the output correctly predict a higher percentage of edges and in this sense are better graph embeddings of the output geometry.

Fig. 8 shows another experiment concerning localization of features. We applied k-means with 6 clusters to the features of intermediate layers and represented the cluster assignments onto the final point cloud. This experiment confirms that the features are highly localized and progressively more so in the layers closer to the output.

We further investigated the effective receptive field of the convolution operation in Fig. 9. This figure reports histograms of Euclidean distances measured on the output point cloud between neighbors as determined by the nearest neighbor graph in one of the intermediate layers. We can see that layers closer to the output aggregate points which are very close in the final point cloud, thus implementing a highly localized operation. Conversely, layers close to the latent space perform more global operations.

B. Upsampling results

The main drawback of the model without upsampling is the unnecessarily large number of parameters in the first dense layer. This is solved by the introduction of the upsampling layers which exploit multi-resolution or self-similarity patterns to lower the number of parameters by starting with a lower number of points and progressively predicting new points from the generated features.

Probabilistic approach: The probabilistic approach models the features in a neighborhood as being drawn from a Gaussian distribution whose mean and covariance are estimated from the neighboring points. This model is based on the assumption that the upsampling operation should be localized, i.e., the generated point should be close to its generating point and its neighborhood. We experimentally tested this behaviour by measuring how "typical" is the distance between the generated

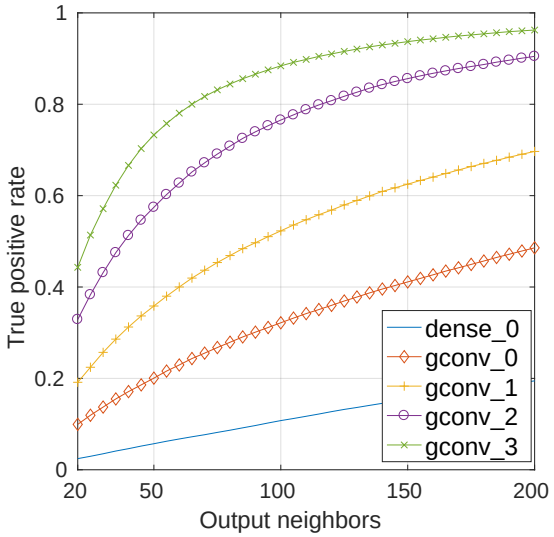


Fig. 7: Accuracy of edge prediction from intermediate layer features.

	Percentage of neighbors	**Distance correlation**
upsamp_0	$(71.8 \pm 7.6)\,\%$	(0.56 ± 0.15)
upsamp_1	$(69.6 \pm 2.8)\,\%$	(0.60 ± 0.03)
upsamp_2	$(61.6 \pm 3.3)\,\%$	(0.53 ± 0.03)
upsamp_3	$(66.4 \pm 3.5)\,\%$	(0.61 ± 0.04)

TABLE IV: Upsampling self-similarity

point and its generator with respect to the distance between the generator and its neighbors. While the neighborhoods are defined in the feature space, the distances are measured as Euclidean distances on the output 3D point cloud. Fig. 10a shows a histogram of the ratio between the generator-generated distance and the average neighborhood distance. We can see that this ratio is concentrated around 1 implying that the generated point is as typical as the original neighbors.

In Fig. 11 we repeat the clustering experiment we previously performed on the model without upsampling. Besides the previously explained graph embedding behaviour, we also notice that the few points in the first layers are generally uniformly distributed and the new points that are generated are localized, creating a multi-resolution hierarchy.

Local aggregation approach: The local aggregation approach computes a new point as a weighted aggregation of neighboring points. The weights of the aggregation are learned by the network, thus letting the network decide the best way to create a new point from a neighborhood, at the expense of an increased number of total parameters. The experiment in Figs. 10b and 12 shows an interesting behavior. First, the generated points are not close to the original point: Fig. 10b shows the ratio between the generator-generated distance and the average neighborhood distance (neighborhoods are defined in the feature space, while distances are measured as Euclidean distances on the output 3D point cloud) and since it is usually significantly larger than 1, we can conclude that the generated point is far from the original generating neighborhood. Then, the clusters in Fig. 12 show that the points in the first layers are not uniformly distributed over the point cloud, but rather form parts of it. The mechanism learned by the network to generate new points is essentially to apply some mild transformation to a neighborhood and copy it in a different area of the feature space. The generated points will no longer be close to their generators, but the structure of the neighborhood resembles the one of the generating neighborhood. This notion is similar to the second-order proximity in the graph embedding literature [47] and it seems that this operation is exploiting the inherent self-similarities between the data features at distant points. To validate this hypothesis we measured two relevant quantities. First, we considered a point i, its neighbors $\mathcal{N}_i^l$ before upsampling, their corresponding points generated by the upsampling operation $\{i^{\text{up}}, \mathcal{N}_i^{l,\text{up}}\}$ and the neighborhood $\mathcal{N}_{i^{\text{up}}}^l$ of point i^{up}. We measured the average percentage of points in $\mathcal{N}_i^{l,\text{up}}$ that were generated from points in $\mathcal{N}_i^l$, i.e. $|\mathcal{N}_{i^{\text{up}}}^l \cap \mathcal{N}_i^{l,\text{up}}|/|\mathcal{N}_i^l|$, as reported in Table IV. The result shows that the neighborhood of a generated point is almost entirely generated by the points that were neighbors of the generator, and that the new points are not neighbors of the original ones. This behavior is consistent over different layers. Then, we measured the Euclidean distances in the feature space between point i and its neighbors $\mathcal{N}_i^l$ and between point i^{up} and $\mathcal{N}_i^{l,\text{up}}$. Table IV reports the correlation coefficient between those distance vectors, which suggests that the shape of the neighborhood is fairly conserved.

C. Complexity considerations

The overall computational complexity of the proposed generator is mainly determined by three operations: i) the graph construction; ii) the computation of the aggregation weights from feature differences; iii) the local aggregations. We now analyze in detail the complexity of each of them.

According to the schemes in Fig. 1 and Fig. 2 every graph convolutional layer requires the construction of the graph from its input features. The graph is constructed by choosing the k nearest neighbors of each point according such some distance metric, e.g., the Euclidean distance, in the feature space. If the number of points is N, then this operation has complexity $O(N^2)$ due to the need to compute all pairwise distances between points. This is clearly a bottleneck for large point clouds. However, some solutions can be devised to deal with this issue. First, we proposed the use of upsampling layers, which gradually increase the number of points in the network layers. As discussed in the previous sections, this is motivated by the fact that the data admit hierarchical priors such as a multiresolution structure or self-similarities which allow to use a smaller number of points to learn a coarser representation and then increase it to learn finer details. Conveniently, due to the quadratic dependence of graph construction, halving the number of points reduces complexity by a factor of four. Also notice that the graph that needs to be constructed from the output of an upsampling layer can reuse the pairwise distances computed from its input since the features of the old points are



Fig. 8: No upsampling: k-means clustering of features of intermediate layers, highlighted onto the output point cloud (leftmost: output of dense layer, rightmost: output point cloud). Notice how layer features generate clusters that are progressively more localized in the output geometry.

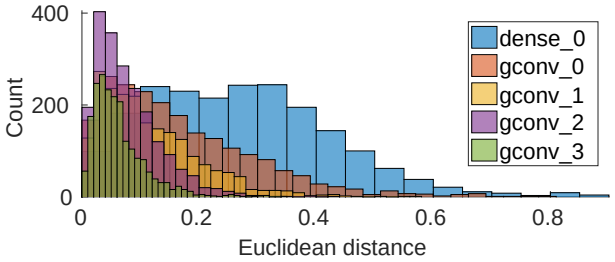


Fig. 9: Histogram of intra-neighborhood distances. Neighborhoods are computed as 20-nearest neighbors in the feature space of each layer. Distances in abscissa are distances in the 3D point cloud. The layer features create neighborhoods that are progressively more localized with respect to the output geometry.

not modified by the upsampling operation and only distances among new points and between old and new points need to be computed. An alternative approach to graph construction that may be considered in future work is limiting the computation of pairwise distances to a search window instead of considering all possible pairs.

The computation of aggregation weights is performed by means of a small neural network F with fully-connected layers. This network contains most of the parameters of the generator. Table II shows the network used for the experiments in this paper, which is composed of two fully connected layers. It can be readily noticed that the last layer packs most of the parameters. In fact, the output of the network is a matrix of aggregation weights of size $d^{l-1} \times d^l$, which means that for a middle layer with d features, the number of parameters is $d \cdot d^l \cdot d^{l-1}$ (since typically $d^{l-1}, d^l = \Omega(d)$, then the number of parameters is $\Omega(d^3)$). This shows that the total number of parameters is dominated by the last layer of the F network and grows cubically with the number of features. This creates computational issues, a risk of overparameterization of the operation, and vanishing gradient problems when the number of features grows. Future work may focus on reducing the number of parameters needed by this operation, thus allowing deeper networks or reducing complexity.

Finally, the aggregation weights must be used in the aggregation operation itself as shown in Eq. 2. This operation requires $O(Nkd^2)$ multiplications for a neighborhood of size k, and N points with d features. We notice that this operation is currently not optimized by frameworks such as Tensorflow [49] and large speedups may be achieved by suitable data structure optimization (e.g., notice that a point may appear in multiple neighborhoods).

VII. Conclusions

We presented a GAN using graph convolutional layers to generate 3D point clouds. In particular, we showed how constructing nearest neighbor graphs from generator features to implement the graph convolution operation promotes the features to be localized and to approximate a graph embedding of the output geometry. We also proposed two upsampling schemes for the generator that exploit multi-resolution decomposition or self-similarities in the samples to be generated.

References

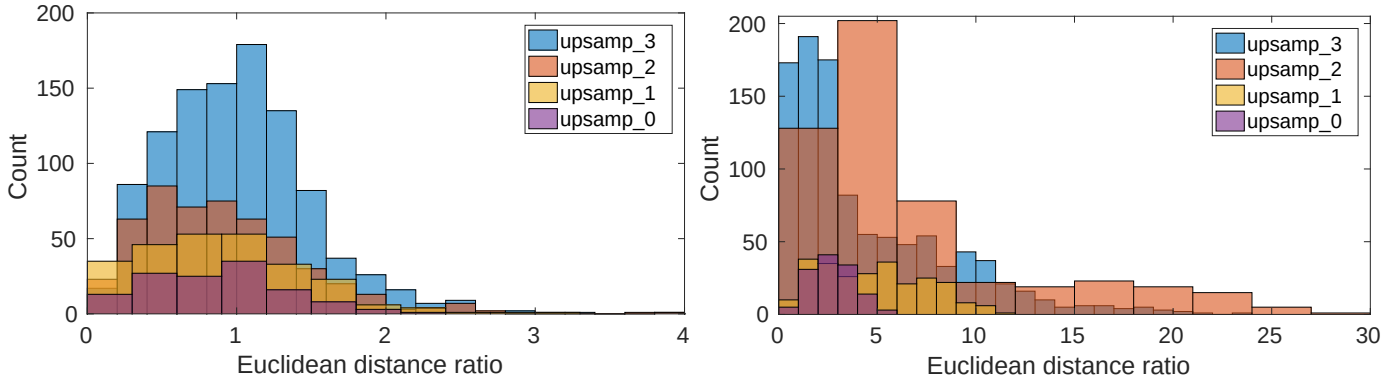
[1] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[2] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 3189–3197.

[3] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proceedings of the IEEE International Conference on Computer Vision workshops*, 2015, pp. 37–45.

[4] H. Zhang, P. She, Y. Liu, J. Gan, X. Cao, and H. Foroosh, "Learning structural representations via dynamic object landmarks discovery for sketch recognition and retrieval," *IEEE Transactions on Image Processing*, vol. 28, no. 9, pp. 4486–4499, Sep. 2019.

[5] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the sequence specificities of dna-and rna-binding proteins by deep learning," *Nature Biotechnology*, vol. 33, no. 8, p. 831, 2015.



(a) Probabilistic upsampling

(b) Aggregation upsampling

Fig. 10: Locality of upsampling.



Fig. 11: Probabilistic upsampling: k-means clustering of features of intermediate layers after upsampling, highlighted onto the output point cloud (leftmost: output of dense layer, rightmost: output point cloud). Black points are not yet generated at the intermediate layers. Notice how the generated points are almost uniformly distributed and how the layer features create neighborhoods that are progressively more localized with respect to the output geometry.

[6] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

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

[8] J. Chen, C. Lin, P. Hsu, and C. Chen, "Point cloud encoding for 3d building model retrieval," *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 337–345, Feb 2014.

[9] P. Wu, Y. Liu, M. Ye, J. Li, and S. Du, "Fast and adaptive 3d reconstruction with extensively high completeness," *IEEE Transactions on Multimedia*, vol. 19, no. 2, pp. 266–278, Feb 2017.

[10] P. d. O. Rente, C. Brites, J. M. Ascenso, and F. M. B. Pereira, "Graph-based static 3d point clouds geometry coding," *IEEE Transactions on Multimedia*, pp. 1–1, 2018.

[11] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[13] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[14] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," *arXiv preprint arXiv:1703.06103*, 2017.

[15] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 29–38.

[16] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *arXiv preprint arXiv:1801.07829*, 2018.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[18] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1558–1566.

[19] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

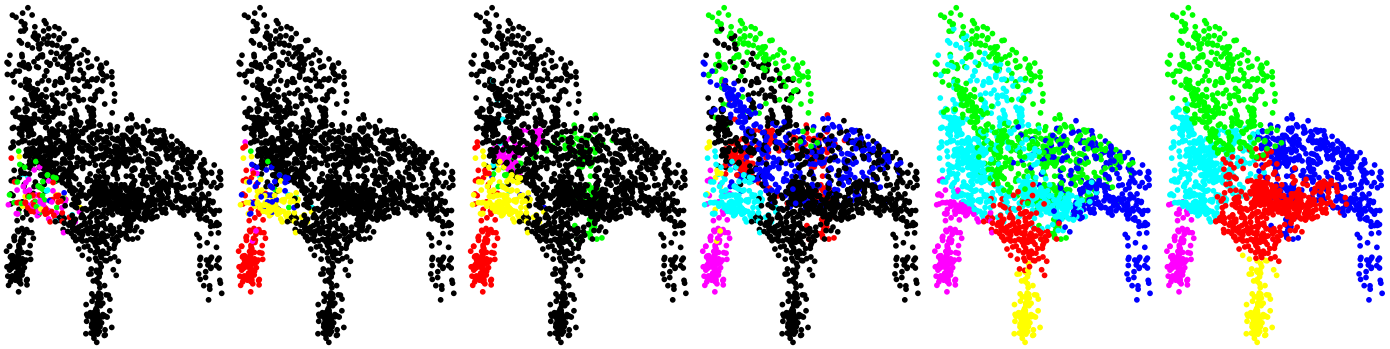


Fig. 12: Aggregation upsampling: k-means clustering of features of intermediate layers after upsampling, highlighted onto the output point cloud (leftmost: output of dense layer, rightmost: output point cloud). Black points are not yet generated at the intermediate layers.

[20] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Representation learning and adversarial generation of 3d point clouds," *arXiv preprint arXiv:1707.02392*, 2017.

[21] D. Valsesia, G. Fracastoro, and E. Magli, "Learning Localized Generative Models for 3D Point Clouds via Graph Convolution," in *International Conference on Learning Representations (ICLR)*, 2019.

[22] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[23] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv preprint arXiv:1701.07875*, 2017.

[24] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Advances in Neural Information Processing Systems*, 2017, pp. 5769–5779.

[25] E. L. Denton, S. Chintala, R. Fergus *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1486–1494.

[26] D. Berthelot, T. Schumm, and L. Metz, "Began: boundary equilibrium generative adversarial networks," *arXiv preprint arXiv:1703.10717*, 2017.

[27] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," in *2018 International Conference on Learning Representations (ICLR)*, 2018.

[28] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[29] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 105–114.

[30] R. A. Yeh, C. Chen, T. Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, "Semantic image inpainting with deep generative models," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 6882–6890.

[31] J. H. R. Chang, C. Li, B. Póczos, and B. V. K. V. Kumar, "One network to solve them all — solving linear inverse problems using deep projection models," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 5889–5898.

[32] M. Yang, W. Zhao, W. Xu, Y. Feng, Z. Zhao, X. Chen, and K. Lei, "Multitask learning for cross-domain image captioning," *IEEE Transactions on Multimedia*, pp. 1–1, 2018.

[33] G. Song, D. Wang, and X. Tan, "Deep memory network for cross-modal retrieval," *IEEE Transactions on Multimedia*, pp. 1–1, 2018.

[34] X. Zhang, X. Zhu, . X. Zhang, N. Zhang, P. Li, and L. Wang, "Seggan: Semantic segmentation with generative adversarial network," in *2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM)*, Sept 2018, pp. 1–5.

[35] X. Liang, L. Lee, W. Dai, and E. P. Xing, "Dual motion gan for future-flow embedded video prediction," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 1762–1770.

[36] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 922–928.

[37] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1912–1920.

[38] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 77–85.

[39] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems*, 2017, pp. 5105–5114.

[40] H. Fan, H. Su, and L. Guibas, "A point set generation network for 3d object reconstruction from a single image," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 2463–2471.

[41] C. Nash and C. K. Williams, "The shape variational autoencoder: A deep generative model of part-segmented 3d objects," in *Computer Graphics Forum*, vol. 36, no. 5. Wiley Online Library, 2017, pp. 1–12.

[42] O. Litany, A. M. Bronstein, M. M. Bronstein, and A. Makadia, "Deformable shape completion with graph convolutional autoencoders," *CoRR*, vol. abs/1712.00268, 2017.

[43] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative Generative Modeling of Graphs," *arXiv preprint arXiv:1803.10459*, 2018.

[44] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," *CoRR*, vol. abs/1711.08267, 2017. [Online]. Available: http://arxiv.org/abs/1711.08267

[45] Y. Yang, C. Feng, Y. Shen, and D. Tian, "FoldingNet: Point Cloud Auto-encoder via Deep Grid Deformation," in *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 3, 2018.

[46] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 922–928.

[47] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *arXiv preprint arXiv:1705.02801*, 2017.

[48] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[49] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/