Feature visualization for 3d point cloud autoencoders

Rios, T.; Stein, N. van; Menzel, S.; Bäck, T.H.W.; Sendhoff, B.; Wollstadt, P.

Feature Visualization for 3D Point Cloud Autoencoders

Thiago Rios*, Bas van Stein†, Stefan Menzel*, Thomas Bäck†, Bernhard Sendhoff* and Patricia Wollstadt*
*Honda Research Institute Europe GmbH, Carl-Legien-Str. 30, 63073 Offenbach, Germany
†Leiden Institute of Advanced Computer Science (LIACS), Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
Email: {thiago.rios, stefan.menzel, bernhard.sendhoff, patricia.wollstadt}@honda-ri.de,
{b.van.stein, t.h.w.baeck}@liacs.leidenuniv.nl

***Abstract*—In order to reduce the dimensionality of 3D point cloud representations, autoencoder architectures generate increasingly abstract, compressed features of the input data. Visualizing these features is central to understanding the learning process, however, while successful visualization techniques exist for neural networks applied to computer vision tasks, similar methods for geometric, especially non-Euclidean, input data are currently lacking. Hence, we propose a first-of-kind method to project the features learned by point cloud autoencoders into a 3D-space augmented with color maps. Our proposal explores the properties of 1D-convolutions, used in state-of-the art point cloud autoencoder architectures to handle the input data, which leads to an intuitive interpretation of the visualized features. Furthermore, we tackle the search for relevant co-activations in the feature space by clustering the input data in the latent space, where we explore the correspondence between network features and geometric characteristics of typical shapes of the clusters. We tested our approach with experiments on a benchmark data set, and with three different configurations of a point cloud autoencoder, where we show that the features learned by the autoencoder correlate with the occupancy of the input space by the training data.**

***Index Terms*—autoencoder, deep learning, feature visualization**

I. Introduction

Training deep learning models on point clouds is a challenging task: point clouds are defined on a non-Euclidean domain, require permutation-invariant network architectures, and lack explicit large-scale information, e.g., point adjacency. Existing work handles these challenges through preprocessing [1], [2], permutation-invariant operations such as maximum pooling [3], or combinations of point clouds with different resolutions [4]. Using these approaches has led to promising accuracy in classification tasks and enabled complex shape transformations that are valuable to different engineering tasks using learned latent variables [5], [6].

In particular, 1D-convolutions provide an elegant solution to the required permutation-invariance (e.g., [3], [4]). The operator handles points individually and yields independent features that can be processed further with global operators, e.g. maximum pooling, at deeper layers of the network. Even though one would expect 1D-convolutions to diminish a network's capability to encode large scale geometric characteristics, existing work on point cloud autoencoders shows that learned latent features can not only be used for operations like data compression and classification, but also for complex tasks such as shape generation in engineering [3], [6].

Although results are visually impressive, most models remain black boxes such that a deeper understanding of how models succesfully represent high-dimensional input data is still widely missing. This lack of understanding hinders the improvement of deep learning architectures and tuning of hyperparameters. These tasks already involve high-dimensional, multi-modal and mixed-integer domains, where an exhaustive search is practically infeasible and the behavior of the model is usually described only by the loss value.

In such cases, feature visualization techniques help to better understand complex model architectures by projecting network features into a human-comprehensible space. In the computer vision field, various feature visualization techniques exist [7]–[9], which enable downstream tasks, for example, transfer of artistic styles [10]. For geometric deep learning models, however, similar techniques are lacking and methods used in the 2D-domain cannot be immediately transferred to geometric input data, due to its high-dimensionality and non-Euclidean nature.

In this paper, we present a novel feature visualization technique for geometric deep learning models. We introduce a formal approach to visualizing and interpreting features in a 3D point cloud autoencoder, which exploits the properties of the 1D-convolutions. 1D-convolutions are point-wise operations that allow to map layer activations back onto the input point clouds, and visualize them as color maps in a series of 3D-scatter plots. Furthermore, we investigate multivariate representations of network features by clustering the input data based on their respective latent representations, such that we can infer on correspondences between geometric characteristics and co-activated units. We demonstrate the ability of the proposed technique to reveal footprints of how the network processes point clouds in order to learn abstract representations by applying our method on a model trained on the car class of ShapeNet core [11]. Then, we compared the geometric characteristics identified by the network to components of the vehicle, e.g., wheel mounts and windshields. In addition, our analysis of the convolution operation provides insights into

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 766186.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

the challenges of transferring features through gradient-based operation using point cloud autoencoders, e.g., as proposed in [12] for 3D neural style transfer.

The remainder of this paper is organized as follows: In Section II we present a literature review with emphasis on point cloud autoencoder architectures and methods for visualizing features of deep neural networks (DNNs). Then, we introduce in Section III the mathematical concepts necessary in visualizing features learned by the proposed point cloud autoencoder. In Section IV, we present the experiments on three configurations of the point cloud autoencoder trained on the car class of ShapeNet core [11], exploring the visualization of features learned by the models. Finally, we present our conclusion and outlook in Section V.

II. Literature review

A. Point Cloud Autoencoders

The development of increasingly powerful graphic processing units (GPUs) and storage technology in the past few years has boosted the research on deep learning models. It allowed for an increase in size and complexity of the models, achieved through the development of new nonlinear activation functions and sophisticated architectures [13]. These advances led to super-human performance in machine learning tasks for a variety of fields. This development made it also possible to handle high-dimensional 3D geometries as input data, which has become increasingly available through novel data-acquisition and 3D scanning technologies, e.g., in computer vision, robotics, or autonomous driving [14], [15].

There exists no canonical representation for 3D data. Frequently used representations include point clouds, voxels, or polygon meshes [14], [16], among which point clouds are a simple and efficient choice for representing 3D data in many tasks [1], [12]. They require less pre-processing effort than voxels, are memory-efficient, and are sampled either directly from real-world objects through 3D scanning or computer aided engineering (CAE) models, and preserve sufficient geometric detail for most applications. Furthermore, deep learning models that process point clouds do not require any topological compatibility between samples in the data set, as expected for meshes [17], widening their range of applications. In real-world scenarios, point clouds are frequently used in computer vision tasks, e.g., in autonomous driving [18]–[20].

Yet, learning on unorganized point clouds can be surprisingly challenging, since operations have to be permutation-invariant, i.e., geometric characteristics have to be learned independently of changes in the ordering of the points. In [1], the authors proposed an autoencoder that contained only *fully-connected* layers combined with a pre-processing operation that yielded a representation of the input that was less sensitive to permutations. The work was later extended in [2], where the authors used a hierarchical structure that recursively applied the previous architecture to partitions of the input data, enabling the abstraction of larger-scale information, which was not achieved in the previous model.

Yang et al. [21] also proposed an architecture with *fully-connected* layers, but with a decoder that was fundamentally different: it emulated the folding of a 2D grid into the 3D shape. The main advantage of their approach was the severe reduction in the number of parameters, requiring less computational effort for model training and improving the scalability to larger point clouds [22]. The presented work pioneered the use of point clouds in geometric deep learning. However, it was shown that these architectures were still sensitive to noise, ordering of the points and performed poorly in input optimization tasks [12]. The latter property is crucial for feature visualization in the 2D domain and thus precludes the application of such techniques in point cloud-based architectures, as we will further discuss in the next section.

In more recent work, authors replaced the *fully-connected* layers by 1D-convolutions: The architecture proposed in [3] contained an encoder with five 1D-convolution layers, each activated by a rectified linear unit (ReLU), followed by a maximum pooling operator and a decoder composed of three *fully-connected* layers. Since the points were handled individually and the pooling was performed over all points, but feature-independently, the permutation of the points did neither affect the calculated features, nor the values of the latent variables. Even though the convolutions operated only on local properties of the point clouds, the authors showed that their architecture enabled both classical machine learning tasks—data compression and classification—and large-scale geometric operations in the latent space, such as shape interpolation and generation.

To overcome the limitations of receptive fields with fixed size, Gadelha et al. [4] proposed an autoencoder that required as input three representations of the point clouds sampled at different resolutions. The underlying motivation was that, while the high-resolution representations allowed the model to learn details of shapes, the low-resolution ones pushed the network to abstract large scale properties. Hence, the model proposed by the authors contained three main streams that operated on each representation, with intermediate resampling and concatenation operators that combined the results obtained in each stream. The proposed architecture was considerably more complex than the proposal of [3], yet it achieved comparable performance to the state-of-the-art models in classification tasks on benchmark data sets.

B. Feature Visualization Techniques for Deep Neural Networks

Understanding how deep neural networks process data is central to guiding architecture optimization and hyperparameter tuning [7]. Feature visualization techniques facilitate this understanding by mapping activations in the trained model to characteristics of the input data, and thus visualize information represented by the models. Literature on visualization of features learned by convolutional neural networks started with the work of Zeiler & Fergus [7]. In [23]–[25] the authors present a comprehensive survey on the methods for feature visualization

and divide them into three main classes: *input modification*, *deconvolutional methods* and *input reconstruction methods*.

Input modification methods, similar to sensitivity analyses, quantify changes in the activation values in a feature map of a specific layer due do local changes in the input data. In [26], the authors adopted the occlusion technique, which belongs to this class, as a data driven approach to determine the receptive fields of a convolutional neural network (CNN). The approach presented to a trained CNN several replications of an input image, where for each image a different region was occluded by a 3×3 patch of random pixels. Then, the difference in the feature map between the original and occluded image was calculated. The underlying assumption was that occluding the most important pixel to a feature map should yield the highest differences in the activations; hence, differences with the highest magnitude would reveal the features encoded by the network at the layer of interest. The approach was found to be effective for 2D data, but extending the method for 3D non-Euclidean data is nontrivial and it is expected to scale poorly to high-dimensional CAE data.

The methods in the deconvolutional class compute the contribution of each input value to the activations obtained in a feature map of interest. The common ground of the techniques in this class is that the values that best characterize the input data for the network task, e.g. classification, contribute more to the output of a layer than uncorrelated ones. The first approach was proposed in [7], where the authors used deconvolution networks to reverse the path of activations back to the input space, revealing which patterns of pixels in the input image were responsible for the calculated activations. Later, in [27], the authors adopted the assumption that the input that contributed the most to the output of a layer was the one that yielded higher derivative values; assuming that the higher the sensitivity of a layer output to a pixel in the input, the higher the relevance of the latter to the network results. Although this assumption is arguable—it neglects gradients with zero magnitude, which may indicate pixels that yield local maxima and, therefore, that drive the network to a certain behavior—the authors handled the nonlinearity of the layers with Taylor-series expansions, which simplifies the calculation of the derivatives and can be extended to other architectures.

Lastly, input reconstruction methods iteratively modify the input data as in an optimization problem, driving the activations in a feature map towards a certain objective. However, selecting the right objectives for the input reconstruction in order to reveal features of the network is not a straightforward task, because the activation space enables numerous combinations of activated units to represent a single feature in the input space. In [28], the authors proposed a multifaceted feature visualization approach, also based on a computer vision problem, where they first identified which activations fire more frequently when specific classes of their input data were presented to the network. Then, for a given random input, they iteratively modified the input data to recover some of the previously identified co-activations and, thus, revealing features in the input space. Differently, [9], [29] used random search in the activation space to determine which co-activations could be used as targets in the optimization of the input to reveal features learned by the network.

Similar to the identification of co-activated units for feature visualization, in [30] the authors clustered the input data, using the information in the input space, and compared the patterns of activations with the clustering results. Even though this task was not central to the topic of the paper, clustering the input data based on its representation in the activation space is a potential solution to identify co-activations that are relevant to the network. Furthermore, clustering avoids an exhaustive search in a high-dimensional space, it can be extrapolated to multiple machine learning algorithms and, if the input data set is labeled, the comparison of the labeling might reveal important information on how the network abstracted the input data.

III. Methods

The visualization techniques reviewed in the previous section focus on images as input data. Extending the approaches to geometric deep learning is not straightforward, since the representations are often non-Euclidean and the architectures vary according to the type of input data. In this study, we propose a method usable in architectures employing 1D-convolutions, such as point cloud autoencoders and, therefore, we first provide the necessary mathematical background on the operations performed by the deep neural network.

A. Point Cloud Autoencoder

In this paper, we use an autoencoder that is based on the proposal in [3], with an encoder comprising convolutional layers followed by a maximum pooling operation, as detailed in Table I. In comparison to the original architecture, we changed the activation function in the layers immediately before the maximum pooling to hyperbolic tangent and the activation functions in the output layer to sigmoid. For further details and a validation of the model used, see [6].

TABLE I
BASELINE ARCHITECTURE OF THE POINT CLOUD AUTOENCODER.

Layer	Type	Activation	Features	Output dimensions
1	1D-C[a]	ReLU	64	[N[b]×64]
2	1D-C	ReLU	128	[N×128]
3	1D-C	ReLU	128	[N×128]
4	1D-C	ReLU	256	[N×256]
5	1D-C	tanh	L[c]	[N×L]
6	maxPool	-	L	[1×L]
7	FC[d]	ReLU	256×3	[256×3]
8	FC	ReLU	256×3	[256×3]
9	FC	sigmoid	N×3	[N×3]

[a]1D-C: 1D-convolution
[b]N: Size of the point cloud
[c]L: Number of latent variables
[d]FC: *Fully connected*

The 1D-convolution operates point-wise over the Cartesian coordinates of the input point clouds, such that the size of the point cloud is preserved throughout the layers of the encoder. Mathematically, consider an input point cloud $\mathbf{G} \subset \mathbb{R}^3$, from which we sample a point $\mathbf{p}_i = (x_i, y_i, z_i)$. The F_ℓ activations

$\mathbf{y}_{i,\ell}$ at a given layer ℓ are recursively calculated according to the following equation:

$$\mathbf{y}_{i,\ell} = \varphi\Big(\mathbf{y}_{i,\ell-1} \times \mathbf{W}_\ell + \mathbf{b}_\ell\Big), \tag{1}$$

where φ is the activation function, $\mathbf{y}_{i,\ell-1}[1 \times F_{\ell-1}]$ are the activations of the previous layer and the coordinates of the input point if $\ell = 1$, $\mathbf{W}_\ell[F_{\ell-1} \times F_\ell]$ is the matrix of weights, and $\mathbf{b}_\ell[1 \times F_\ell]$ is the bias vector. Repeating the operations for the remaining $N-1$ points in $\mathbf{G}$, we obtain a matrix of activations $\mathbf{Y}_\ell[N \times F_\ell]$.

In CNNs, the output of a layer contains multiple 2D matrices, where each matrix represents a feature detected by the network. In the case of 1D-convolutions, instead of matrices, a layer returns vectors, which correspond to the columns of $\mathbf{Y}_\ell$. Hence, we here consider a network feature j to be a set of activations $\mathcal{F}_\ell^j[N \times 1] \subseteq \mathbf{Y}_\ell$, i.e., the j-th column of the matrix of activations.

The architecture used has implications for possible approaches to visualizing network features. First, the 1D-convolution operates similarly to a multi-layer perceptron (MLP) that is blindly shared over the points in the input shape (Fig. 1). Hence, following the chain rule, the gradient vector for an activation $\mathbf{y}_{j,\ell}$ in a layer ℓ with respect to an input point $\mathbf{p}_i$ is given by the following equation:

$$\frac{\partial \mathbf{y}_{\ell,\mathbf{j}}}{\partial \mathbf{p_i}} = \frac{\partial \varphi(u_{\ell,j})}{\partial u_\ell} \frac{\partial u_\ell}{\partial \varphi(u_{\ell-1})} \cdots \frac{\partial \varphi(u_1)}{\partial u_1} \frac{\partial u_1}{\partial \mathbf{p}_i}, \tag{2}$$

where u_ℓ is the linear operation performed in the layer ℓ. Since the weights do not depend on the input points, maximizing the activations of $\mathcal{F}_\ell^j$ potentially drives the points to singular regions of the input space. These regions correspond to local maxima of the MLP, such that the formed clusters of points do note relate to human-interpretable geometric structures.

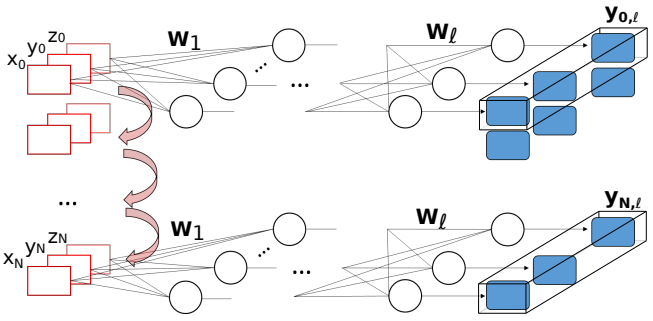

Fig. 1. Schematic of the 1D-convolutions interpreted as operations in an MLP.

If the layer of interest contains rectified linear units (ReLU) as activations, $\nabla y_{\ell,j} = \vec{0}, \forall\ u_{\ell,j} < 0$. Hence, input reconstruction methods based on gradient optimization algorithms might fail to reveal any geometric structure, because the points that satisfy this condition would remain static during the optimization process. A possible solution to overcome the stagnation of the points is to use gradient optimization algorithms with stochastic components, such as the Adam optimizer [31]. Such an approach increases the chances to modify points that led to zero activation, however without any guarantee of converging to an interpretable shape.

Second, since the features are calculated independently for each point, mapping the activations to the samples in the point cloud is a straightforward task. Although deconvolutions and input modification methods are applicable, it is not necessary to perform any additional mathematical manipulation to associate the activations with the input data, which avoids the issues related to the scalability and inversion of complex nonlinear functions.

In sum, visualizing network features in point cloud-based architectures employing 1D-convoltuions can be solved using a simpler, and potentially more efficient, solution than an extension of the methods available in the literature. Differently from the encoder, relating the activations of the decoder to the input coordinates is a much harder task due to the maximum pooling operator and operations with *fully-connected layers*, which prevent a direct mapping between activations and input points. Also, the decoder's task is the generation of shapes rather than dimensionality reduction, which is not our focus in this study. Therefore, we will constrain our analyses in this paper to the features learned by the encoder.

B. Projecting Features onto Point Cloud Representations

Our method is based on two aspects revealed by the previous analyses: first, the direct correspondence between individual points in the input and activations matrices, second, our definition of network features. Hence, in order to visualize a feature $\mathcal{F}_\ell^j$, we propose to project the activations in the j-th column of $\mathbf{Y}_\ell$ as a color map onto the 3D-scatter plot that represents the input point cloud, as depicted in Fig. 2.

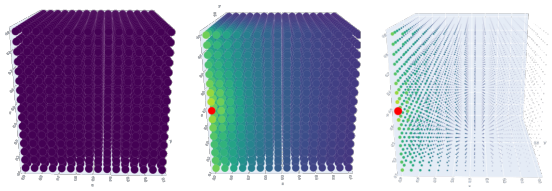

Fig. 2. Example of plot for feature visualization, where the red marker identifies the maximum activation value. Left: original point cloud. Center: feature visualization with colors. Right: enhanced visualization with changes in the markers size.

As presented in the literature review, the analysis of co-activated units can also reveal features learned by the autoencoder [9], [28], [29]. Exploring all possible combinations in an exhaustive fashion, however, is infeasible due to the high-dimensionality of the activation space. Therefore, more efficiently than pure random exploration, we tackle the selection of co-activated units with unsupervised clustering based on the latent representations of the input data. The clustering organizes the data according to their characteristics in the latent space only. Thus, it provides evidence on how the network differentiates the input samples by revealing characteristic co-activations from the feature space.

In order to reduce misclassification, we remove shapes that yield extreme loss values prior to the clustering. In other words, we aim to remove samples that have a very high or low loss value, indicating an under- or overfitting of the model to these samples. We use kernel density estimation (KDE) to describe the distribution of losses in the data and exclude samples that fall below a minimum density, τ. This filtering of the data set selects shapes that are predominantly represented by the features learned by the autoencoder, reducing the noise for clustering the representations and avoiding a misinterpretation of the network features.

After applying the clustering algorithm, the results are verified by calculating silhouette scores for each shape, which indicates the degree of similarity between the individuals in a cluster and provides an overview of the clustering quality. Then, in an additional step, we calculated the mean representation of each cluster and the respective deviations of the latent variables. We explored how geometric properties correspond to large differences between cluster means to find a geometric interpretation of the features learned by the network.

Finally, our approach is summarized in the workflow depicted in Figure 3. The tasks were described for a general case, however the method can be adapted according to the objectives of the analysis of the network features.

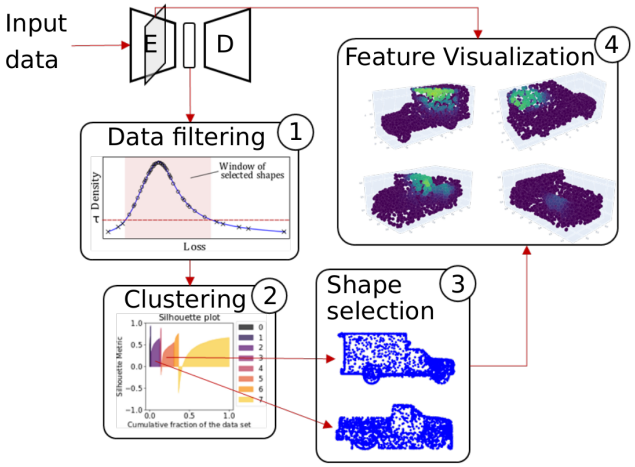

Fig. 3. Worflow with the procedures to visualize and compare network features obtained from different shapes.

IV. Experiments and Discussions

In this section we present a sequence of experiments that illustrate possible approaches to interpret the network activations using the proposed feature visualization method.

A. Training the Autoencoder

For our experiments, we trained three versions of the autoencoder described in Table I, varying only the number of latent variables: 2 (model LR2), 10 (model LR10) and 20 (model LR20). The data set used for training the model contained uniformly sampled shapes from the car class of ShapeNetCore [11], following the implementation in [3]. We used the data partitioning tree algorithm [4] for pre-processing the point clouds, which does not affect calculating the features, but supports the identification and labeling of the points. We split the data into 90 % samples for training and 10 % for testing the autoencoder.

We trained the models using the Adam optimizer [31], with learning rate η = 5E-04, β_1 = 0.99 and β_2 = 0.9. The termination criteria were defined as 700 epochs or an average loss function (Chamfer Distance [32]) below 3.0E-04 for the training set. Unless specified otherwise, we adopted a batch size of 50 and did not apply any data augmentation to the input or layer results.

B. Evaluation and Clustering of the Models

In the first step of the analysis, we randomly selected 7400 models from the data set and evaluated their reconstruction using the models LR2, LR10 and LR20. To generate the corresponding KDE models, we used the algorithm implemented in [33] with Gaussian kernels and a bandwidth corresponding to 10 % of the standard deviation of the data set losses. We defined the density threshold τ (Fig. 4) for filtering the geometries as 10 % of the maximum density evaluated (chosen experimentally, results not shown). This setting led to subsets for clustering with 6927 (93.61 %), 7127 (96.31 %) and 7165 (96.82 %) shapes.

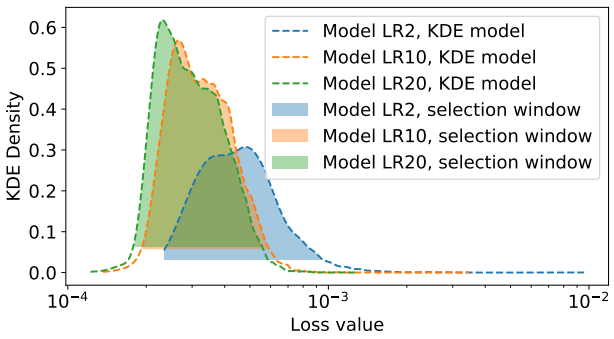

Fig. 4. Selected windows for filtering the designs before clustering the latent representations.

We clustered the latent representations of the filtered set of shapes using Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [34], with Euclidean distance as metric to evaluate similarity in the data. In our experiments, we adjusted the hyper-parameters of minimum neighborhood and samples per cluster, such that the three models yielded the same number of clusters. We observed that our selection of hyper-parameters was nonconservative, leading, in all cases, to a larger cluster 7 with several outliers (e.g. Fig. 5). For our analyses, we did not optimize the hyper-parameters of the clustering algorithm, since clustering performance was not central to the visualization of features. We excluded cluster 7 from our analyses to avoid misinterpretation of the results.

In the following step, we performed a preliminary, visual analysis of the clusters by analyzing correspondences between

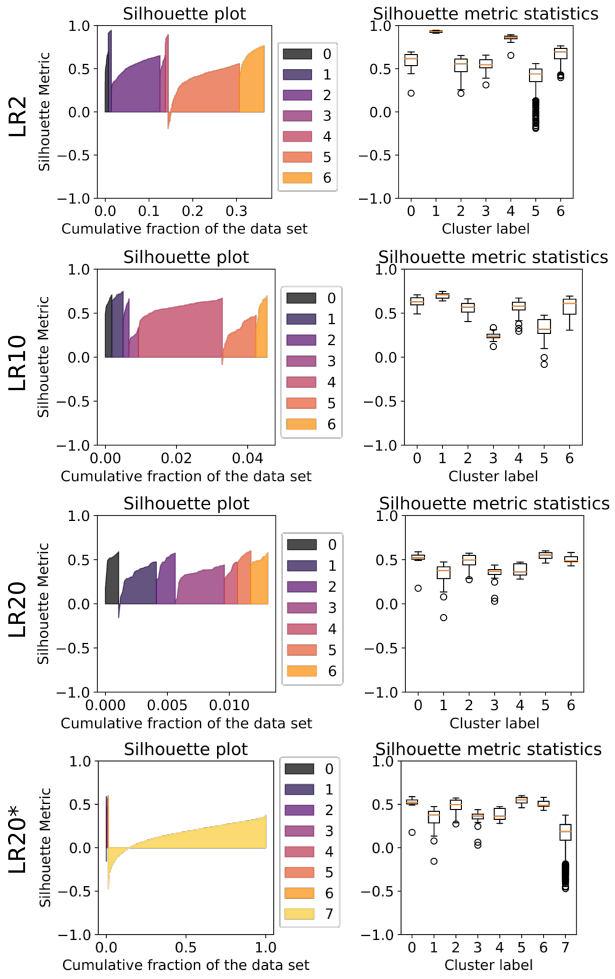

Fig. 5. Evaluation of the silhouette scores for the clustered latent representations. At the bottom, the complete results obtained with model LR20, for comparison.

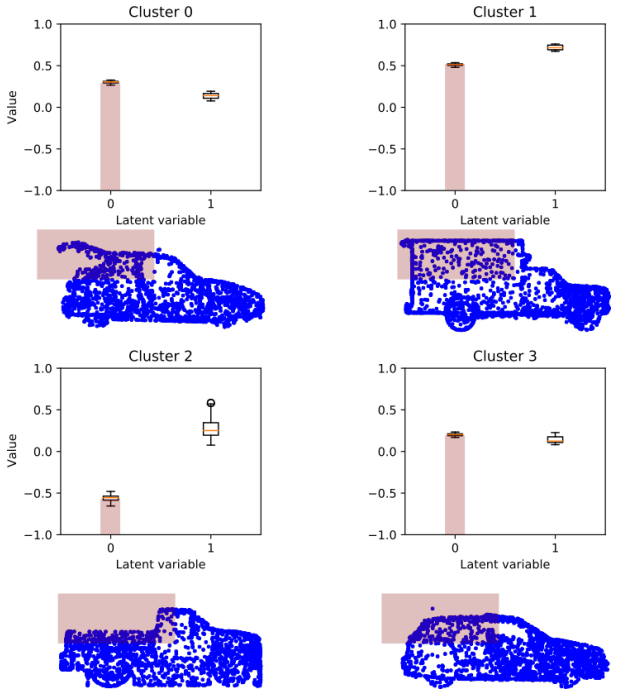

Fig. 6. Comparison of the latent variables and shapes from four different clusters obtained with model LR2.

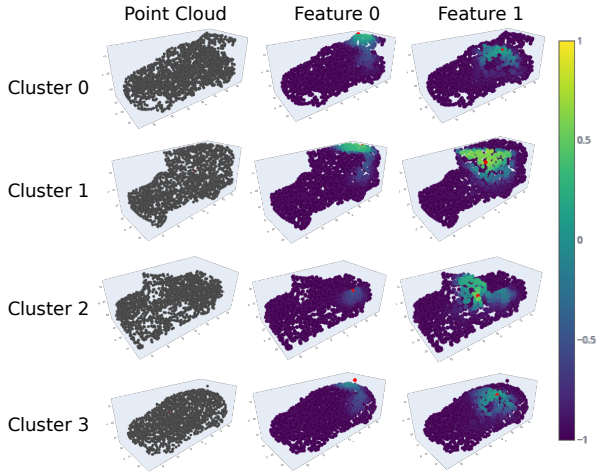

Fig. 7. Comparison of the latent variables and shapes sampled from four different clusters obtained with model LR2.

shape characteristics and mean latent variables. In this preliminary analysis, we observed that the latent variables correlated with the occupancy of the input space. When inspecting the results obtained for the model LR2 (Fig. 6), for example, latent variable 0 yields higher values whenever the input shape occupies the top-rear region of the domain (red boxes). Although visual inspection hints at the features represented by individual latent variables, we can neither confirm to which extend the feature 0 are affected by the input shape, nor identify if the points in the highlighted region caused the highest activations.

C. Feature visualization

In order to verify the relation between the latent variables and distribution of points in the input space, we projected the activations of the last convolutional layer onto the point clouds as illustrated in Fig. 7. Since the last convolutional layer is the lowest-dimensional and is expected to encode the most abstract features, we constrained our analyses to the features in this layer. Nevertheless, the method is valid for any convolutional layer of the decoder.

By visualizing the features, it becomes clear that the points closer to the top corners of the input domain yield the highest activations. Furthermore, we also observed that the features map *similar regions in the input space rather than similar geometric characteristics*, e.g., shape primitives. We further investigated this hypothesis in the experiments presented in the following.

1) Response to shift and rotation of the input data: If autoencoder features represented the occupancy of regions, shifting the input points would also move the activated regions with respect to the shape. In order to test this hypothesis, we calculated and visualized the features for a shape of cluster 1,

obtained with model LR2, with different degrees of shift in the x-direction (Fig. 8) and rotation around the vertical axis (Fig. 9).

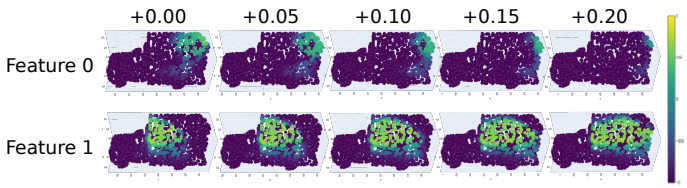

Fig. 8. Visualization of the features obtained for a car shape shifted with different forward steps in the x-direction as input and using model LR2.

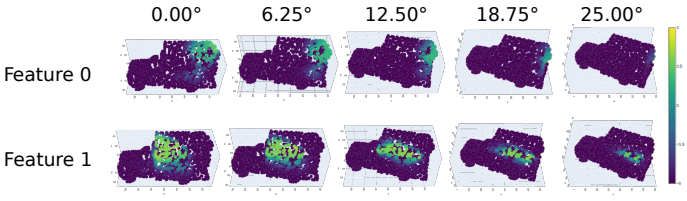

Fig. 9. Visualization of the features obtained for a car shape with different alignments in the (xy)-plane as input and using model LR2.

As can be seen from the feature visualization, when we moved the points, different parts of the geometry were highlighted when mapping the activations onto the point cloud. In other words, the region of activated points remains static with respect to the input space, supporting our hypothesis that the autoencoder learns occupancy in the input space.

2) Transfer of Network Features: As a second test of our hypothesis, we transferred a single network feature, from a geometry that belongs to a cluster, to an initial shape, sampled from a different cluster. More specifically, we substituted the activations of the feature that encodes the region in the underbody of a passenger car of cluster 4, by the corresponding feature obtained from a truck of cluster 0. Here, we used the clusters calculated from the latent representations obtained with model LR10. If our hypothesis holds, reconstructing the shape with transferred features should yield a point cloud with points in the region of the underbody, but without geometric properties that are characteristic to the truck underbody.

As shown in Fig. 10, transferring the network feature added an underbody to the car shape, however, without any specific geometric characteristic of the truck. Although the results apparently match our expectation, a few additional aspects should be highlighted: First, we transferred a single network feature, while describing the underbody might require multiple features and, therefore, the resulting shape did not present any structure that is characteristic of the truck. Second, the maximum pooling layer preceding the decoder operates as a filter, such that only changes in the maximum activation of each feature interferes with the reconstruction of the shape. Hence, transferring only the maximum activation value of the same feature used in the experiment would be sufficient to obtain the same results. Third, we neglected the influence of the decoder on the reconstruction, which in a general case, could have a different architecture and potentially lead to different results.

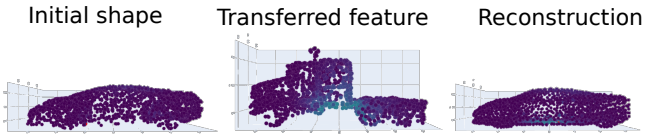

Fig. 10. Interpolation of features between shapes from different clusters and obtained with model LR10.

Nevertheless, the results of the previous experiment also highlight the differences between the tasks of the encoder and decoder. While the encoder learns the occupancy of regions in the input space, the decoder has to reconstruct the coordinates of the points from the combination of occupied spaces. This reconstruction is a much harder task, mirrored by the difference in the number of parameters of both structures—the decoder contains about 10 times more parameters to be trained. We investigated this observation further in a second experiment using a transfer of network features.

In this second experiment, we transferred the features between two pairs of shapes, where the initial geometry is common to both pairs, but in one pair the shapes have more geometric characteristics in common than in the other pair. If our hypotheses were correct, we would expect that the feature transfer led to a smoother interpolation for the geometric more similar pair of shapes than for the more dissimilar.

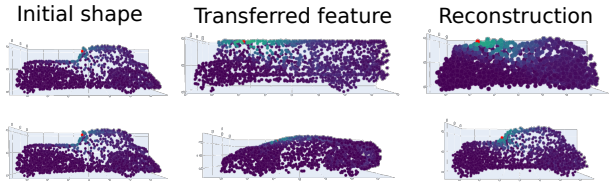

Fig. 11. Transfer of network features between pairs of shapes with different degrees of similarity and reconstruction using the trained decoder.

As shown in Fig. 11, the transfer between the two car shapes yielded a recognizable car shape (bottom), while transferring the feature from the bus yielded a fuzzy point cloud. Along the lines of our previous experiments, many of the car shapes in the training set share common geometric properties and hence, occupy similar portions of the input space. In fact, the more similar pair was closer to the ”average car shape” contained in the data set (see next section). Therefore, for feature transfer between common car shapes, the decoder is still capable of generating a plausible shape from the combination of occupancies. In the case of the transfer from the bus, the data set lacks shapes with geometric characteristics common to cars and buses. As a result, the combination of features is not meaningful for the decoder and it fails to generate an intermediate shape. This further supports the observation that the encoder learns an occupancy of the input space instead of human-interpretable geometric characteristics,

e.g., the "sportiveness" of the car design, which could be transferred to fundamentally different shapes such as the bus.

3) Linear Combination of Features: In this last experiment, we compared the regions most frequently occupied by the training data to the linear combinations of the features calculated using a dense uniform lattice as input. We expected that the visualization of the network features on a lattice would potentially highlight the same regions as the most frequently occupied regions in the input space.

In order to find the regions most frequently occupied by the input, we generated a KDE model over the points in the training data. The kernel was defined as Gaussian with a bandwidth of 6.3×10^{-3}, which corresponds to 10 % of the minimal standard deviation calculated from the point coordinates. Then, we evaluated the KDE model on a uniform lattice with 64^3 samples in the domain $[0.1, 0.9]^3$, which is the input space of the three variations of the autoencoder. To combine the features, we linearly transformed the activation values from the image of the hyperbolic tangent function, $[-1, 1]$, to $[0, 1]$, such that we could sum the activations for all the features and for each point of the lattice without canceling contributions.

In Fig. 12, we superposed the visualization of the density values, obtained with the KDE model, and combined features calculated on the uniform lattice. Differently from our expectations, the highest activations were not restricted to the points with high KDE scores. In fact, for the model LR2, the activated points of the lattice nearly followed the contour of the KDE car shape, and the activations increased for points further away from the car shape. For the models with higher-dimensional latent spaces, however, large portions of the lattice became activated and even from different perspectives of the domain, we could not observe any boundary following the average car shape, as for LR2.

Our interpretation of the results is that the autoencoder learns how to differentiate the shapes from the training set as a priority. When the data is excessively compressed, as in LR2, the activations describe roughly the boundaries between the most typical shape in the training set and adverse ones, such that it can drive the decoder to fair approximations of the coordinates. As the dimensionality of the latent space grows, the autoencoder has more features available to describe geometric details and, therefore, the activated region of the input space grows and no longer describes a clear boundary. Furthermore, our interpretation explains the number of outliers and poor similarity between shapes of cluster 7 obtained with models LR10 and LR20, when compared to LR2 (Fig. 5). Since the autoencoders can describe the shapes with more detail, we would need more clusters to separate the data in more homogeneous classes.

V. Conclusion and Outlook

In this paper we proposed a novel method to visualize and interpret features learned by a 3D point cloud autoencoder. The core of the method exploits the properties of the 1D-convolution operator, used in the architecture, which allows for an intuitive mapping between activations and input data. Furthermore, we tackled the search for co-activations in the learned latent space by clustering the input data based on their respective latent representations, in order to identify features represented by the activation of combinations of units. This clustering approach is more efficient than pure random search in the activation space.

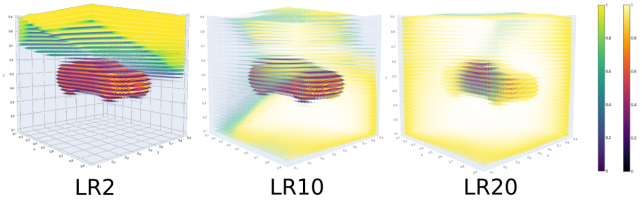

Fig. 12. Visualization of the KDE score and combined features of a dense uniform lattice in the input space. For visualization purposes, the size of the markers varied with the value of the metrics, redundant to the colors. In the representations, the lightest color indicates the highest values for both metrics.

We demonstrated, that visualizing the activations of the autoencoder with intuitive representations plays an important role in understanding which kind of information the autoencoder learns, and provides a footprint of the operations performed by the network. From our interpretation of the extracted features, we conclude that the encoder preferably learns how to differentiate the shapes based on the occupancy of the input space. This property became particularly evident, when we visualized the activated regions for a single point cloud at different regions in the input space and the combined features calculated from the uniform lattice. In summary, the point cloud autoencoder represents various shapes through encoding of spatial occupancy opposed to, for example, geometric properties related to the stylistic features. However, spatial occupancy and geometric features may coincide for some cases, e.g., for a van versus a pick-up truck, where we may still use the learned latent representation for downstream tasks such as shape classification.

Our analyses leave a wide range of research questions unanswered. First, our approach is limited to features calculated with 1D-convolutions. It thus does not allow for the analysis of models using different operators, such as for example, the decoder, which uses fully connected layers and may learn more high-level features. Visualizing the features of the decoder is hence subject to future work. Also, for the sake of clarity, we analyzed latent spaces that were much lower in dimensionality than latent spaces typically used in the literature [3]. In general, the method is limited to lower number of latent variables as the interpretability of higher numbers of features becomes difficult. Finally, we demonstrated that the 1D-convolutions and ReLU activations have an impact on input optimization tasks, which is not only relevant for selecting training hyper-parameters, but also could be explored in more detail in the field of 3D neural style transfer.

REFERENCES

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2017.

[2] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 5105–5114.

[3] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3D point clouds," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 40–49.

[4] M. Gadelha, R. Wang, and S. Maji, "Multiresolution tree networks for 3D point cloud processing," in *Computer Vision – ECCV 2018*. Springer International Publishing, 2018, pp. 105–122.

[5] N. Umetani, "Exploring generative 3D shapes using autoencoder networks," in *SIGGRAPH Asia 2017 Technical Briefs*. New York, NY, USA: ACM, 2017, pp. 24:1–24:4.

[6] T. Rios, B. Sendhoff, S. Menzel, T. Bäck, and B. van Stein, "On the efficiency of a point cloud autoencoder as a geometric representation for shape optimization," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019, pp. 791–798.

[7] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 818–833.

[8] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 5188–5196, 2015.

[9] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 3319–3327. [Online]. Available: https://doi.org/10.1109/CVPR.2017.354

[10] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2019.

[11] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An information-rich 3D model repository," *arXiv preprint arXiv:1512.03012v1 [cs.GR]*, 2015.

[12] T. Friedrich, N. Aulig, and S. Menzel, "On the potential and challenges of neural style transfer for three-dimensional shape data," in *International Conference on Engineering Optimization*. Springer International Publishing, 2019, pp. 581–592.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 1026–1034, 2015.

[14] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, "Deep learning advances in computer vision with 3D data," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–38, 2017.

[15] Z. Cai, J. Han, L. Liu, and L. Shao, "RGB-D datasets using microsoft kinect or similar sensors: a survey," *Multimedia Tools and Applications*, vol. 76, no. 3, pp. 4313–4355, 2017.

[16] E. Ahmed, A. Saint, A. E. R. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada, and B. Ottersten, "Deep learning advances on different 3D data representations: a survey," *arXiv preprint arXiv:1808.01462v1 [cs.CV]*, 2018.

[17] O. Sorkine, "Laplacian Mesh Processing," *Eurographics - State of the Art Reports*, no. Section 4, pp. 53–70, 2005.

[18] Y. Zhong, S. Wang, S. Xie, Z. Cao, K. Jiang, and D. Yang, "3D scene reconstruction with sparse LiDAR data and monocular image in single frame," *SAE Int. J. Passeng. Cars Electron. Electr. Syst.*, vol. 11, pp. 48–56, Sep. 2017.

[19] Z. Ouyang, Y. Liu, C. Zhang, and J. Niu, "A cGANs-based scene reconstruction model using LiDAR point cloud," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, Dec. 2017, pp. 1107–1114.

[20] H. F. Murcia, M. F. Monroy, and L. F. Mora, "3D scene reconstruction based on a 2D moving LiDAR," in *Applied Informatics*, H. Florez, C. Diaz, and J. Chavarriaga, Eds. Springer International Publishing, 2018, pp. 295–308.

[21] Y. Yang, C. Feng, Y. Shen, and D. Tian, "FoldingNet: Point cloud auto-encoder via deep grid deformation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 206–215, 2018.

[22] P. Mandikal and R. V. Babu, "Dense 3D point cloud reconstruction using a deep pyramid network," *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1052–1060, 2019.

[23] Q. Zhang and S.-C. Zhu, "Visual interpretability for deep learning: a survey," *Frontiers of Information Technology and Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018.

[24] C. Olah, A. Mordvinstev, and L. Schubert, "Feature visualization: How neural networks build up their understanding of images," *Distill*, 2017. [Online]. Available: https://distill.pub/2017/feature-visualization/

[25] F. Grün, C. Rupprecht, N. Navab, and F. Tombari, "A taxonomy and library for visualizing learned features in convolutional neural networks," *arXiv preprint arXiv:1606.07757 [cs.CV]*, 2016.

[26] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene CNNs," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[27] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *2nd International Conference on Learning Representations, ICLR 2014 - Workshop Track Proceedings*, pp. 1–8, 2014.

[28] A. Nguyen, J. Yosinski, and J. Clune, "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks," *arXiv preprint arXiv:1602.03616 [cs.NE]*, 2016.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pp. 1–10, 2014.

[30] D. Wei, B. Zhou, A. Torrabla, and W. Freeman, "Understanding intra-class knowledge inside CNN," *arXiv preprint arXiv:1507.02379 [cs.CV]*, 2015.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[32] H. Fan, H. Su, and L. Guibas, "A point set generation network for 3D object reconstruction from a single image," in *30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Januar, 2017, pp. 2463–2471.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172.