2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Deep Compression for Dense Point Cloud Maps

Louis Wiesmann, Andres Milioto, Xieyuanli Chen, Cyrill Stachniss, Jens Behley

***Abstract*— Many modern robotics applications rely on 3D maps of the environment. Due to the large memory requirements of dense 3D maps, compression techniques are often necessary to store or transmit 3D maps efficiently. In this work, we investigate the problem of compressing dense 3D point cloud maps such as those obtained from an autonomous vehicle in large outdoor environments. We tackle the problem by learning a set of local feature descriptors from which the point cloud can be reconstructed efficiently and effectively. We propose a novel deep convolutional autoencoder architecture that directly operates on the points themselves so that we avoid voxelization. Additionally, we propose a deconvolution operator to upsample point clouds, which allows us to decompress to an arbitrary density. Our experiments show that our learned compression achieves better reconstructions at the same bit rate compared to other state-of-the-art compression algorithms. We furthermore demonstrate that our approach generalizes well to different LiDAR sensors. For example, networks learned on maps generated from KITTI point clouds still achieve state-of-the-art compression results for maps generated from nuScences point clouds.**

I. Introduction

Maps are a key ingredient for robot navigation, involving tasks such as localization, planning, or scene understanding. Over the past fifteen years, a lot of robot navigation systems have moved from 2D to 3D representations. Moving from 2D to 3D substantially increases the required amount of memory. Particularly, if large-scale environments shall be represented, storing and especially transmitting it over network links becomes a challenge due to their large size.

Modern 3D LiDAR sensors, which are typically used on autonomous cars or mobile robots, generate millions of points in just a couple of seconds. Point clouds are naturally unordered and quite memory inefficient, but they represent the raw data that such sensors generate and are the main representation used, for example, for scan registration. Although upcoming 5G networks will enable higher data transfer rates, representing 3D point cloud data in a compact and structured way is key to handle the huge amount of data. Our approach focuses on effectively compressing 3D point cloud data collected with modern 3D LiDAR sensors in large, outdoor environments.

Representing 3D data in a memory-efficient way is a common problem in robotics and computer graphics and several approaches exist that tackle this challenge. Binary space partition approaches, like Octrees [20] or kD-trees [4], utilize a hierarchical data representation that allows for efficient storage and fast neighbor queries. Voxel grids [1] discretize the world enabling constant time access, however, at the cost of incurring a large memory footprint if a fine resolution is desired. This makes it inadequate for large-scale environments. To overcome this problem, Octomaps [12] store voxels in a sparse octree structure, thereby reducing the memory requirements. Other representations [18], [39] make strong assumptions that often do not hold in the real world. Another technique is to use triangular meshes to approximate the environment or use multiple primitives, like planes, cylinders, or spheres [35], but such predefined geometries may not fully exploit the available potential of recurring structures. Therefore, learning-based representations often try to capture common characteristics

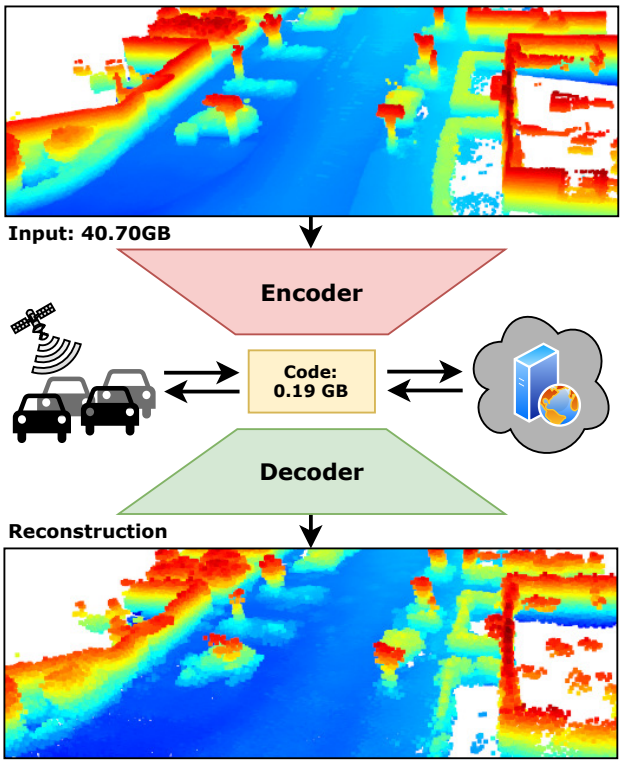

Fig. 1: 3D point clouds obtained by an autonomous car or robot require a large amount of memory. Transferring this data to a fleet of cars or sending it to some cloud service requires a compression of the data. Our approach can reconstruct dense point clouds from a highly compressed representation even when targeting low bit rates. The point cloud color indicates the height above ground ranging from blue (low) to red (high).

Manuscript received: October, 15, 2020; Revised January, 8, 2021; Accepted January, 25, 2021.

This paper was recommended for publication by Editor Javier Civera upon evaluation of the Associate Editor and Reviewers' comments. This work has partially been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070 - 390732324 - PhenoRob.

All authors are with the University of Bonn, Germany. louis.wiesmann@igg.uni-bonn.de

of the scene for further compression [32]. Recent deep neural autoencoder networks [13], [15], [29] provide data compression for different domains and they have also been successfully used to compress 3D point cloud data. This work belongs to this class of approaches and proposes a novel architecture exploiting state-of-the-art representation learning for 3D point clouds to achieve high compression rates.

The main contribution of this paper is a novel deep learning-based approach for compressing 3D LiDAR point cloud data in a lossy fashion for large-scale outdoor environments (see Fig. 1 for an illustration).

Our approach exploits the occurrence of common structures through local feature descriptors. It learns a small and compact set of local feature descriptors that allows for compressing and reconstructing point cloud data. Our approach is end-to-end learnable and provides dense point clouds even when targeting low bit rate compression. Additionally, we propose a novel deconvolution for point clouds with feature propagation and integration. Our deconvolutional kernel operates directly on a set of points, which makes discretization unnecessary. We compare our approach with state-of-the-art compression techniques such as Draco [8] and the octree-based approach from Mekuria et al. [21]. In brief, we are able to provide higher quality maps after decompression than the state-of-the-art compression approaches at the same bit rate. Together with this paper, we plan to release the source code along with the trained models at `https://github.com/PRBonn/deep-point-map-compression`.

II. Related Work

In recent years, we witnessed an increasing interest in deep learning using point clouds from the computer vision and robotics community. Guo et al. [10] survey the field and here we concentrate on representation learning and point cloud compression using neural networks.

Deep Learning on Point Clouds. The seminal PointNet approach [26] uses multi-layer perceptrons (MLPs) followed by a pooling operation to extract an embedding for a whole unordered point cloud. The follow-up work PointNet++ [27] extends this idea to a hierarchical approach by applying the PointNet structure repeatedly to local neighborhoods for different scaling factors. Each layer downsamples the point cloud by iteratively choosing the furthest point from the already collected subset to keep a good coverage of the point cloud. Yongcheng et al. [43] extend this idea by a multi-level feature aggregation to increase the contextual information.

Grid-based approaches [19], [42] compute local features using 3D convolutions on a voxel grid. Octree-based approaches [30], [41] alleviate the large memory footprint of high-resolution voxel grids by using a hierarchical representation to efficiently represent free space. Our approach operates directly on the point cloud to avoid discretization errors and excessive memory usage.

For the extraction of local features, we use the convolutions proposed by Thomas et al. [38]. They define a continuous convolution for unordered point clouds using kernel points. A convolutional kernel learns a weight for each position in a set of kernel points. The 3D continuous weight space is an interpolation of these discrete positions. They design the network architecture for segmentation and classification which allows them to use skip connections to recover the points which get lost in the downsampling process. This is not feasible for compression, since this would require storing intermediate results of the encoder, which we want to avoid to efficiently compress the point clouds. Therefore, we define a continuous 3D deconvolution to recover the points based on their learned feature.

Yu et al. [44] propose a network to upsample point clouds using local feature matrices. In contrast to their hierarchical purely point-based approach, our deconvolution integrates and propagates features. We use multiple deconvolution layers which allows for more flexible upsampling rates at inference time.

Point Cloud Compression. Many works in the field of robotics and computer vision also focus on compressing point cloud data. Octrees [20] can efficiently store three-dimensional data and can be used as an efficient map representation which is commonly used in robotics [12], [6]. This already compact representation can be further compressed by using additional geometric heuristics [16], [34]. Huang et al. [14] use neural networks to learn a conditional probability model, followed by entropy coding to compress frequent symbols with fewer bits than rarer symbols. Octrees efficiently store three-dimensional coordinates but require additional methods to compress attributes [45]. Other approaches focus on an iterative prediction of neighboring points using spanning trees [9], [22]. Tree structures are very memory efficient in most real-world scenarios but do not exploit the full potential of recurring common objects.

The correlation between point clouds acquired from LiDAR or depth camera streams offers a further possibility to save memory. A range-image-based representation allows for the usage of image or video compression algorithms [24], [37], [40]. However, such a projective representation is not suitable for large dense maps captured from many locations, since it only represents the environment of the actual viewpoint well, but is not very generalizable.

Golla et al. [7] exploit standard image compression for static point clouds by storing oriented compressed height and occupancy images for local patches. Deep convolutional autoencoders can use a rate-distortion loss [28], [29] to ensure a good trade-off between quality and memory consumption. These approaches use grids and differentiable quantization instead of a combination of points and features. Similar to Huang et al. [15], we also propose an end-to-end learnable point cloud compression autoencoder network. In contrast to their approach to embed the information of the whole point cloud into a single feature, we store a set of local feature descriptors together with positions from which we can then restore the point cloud.

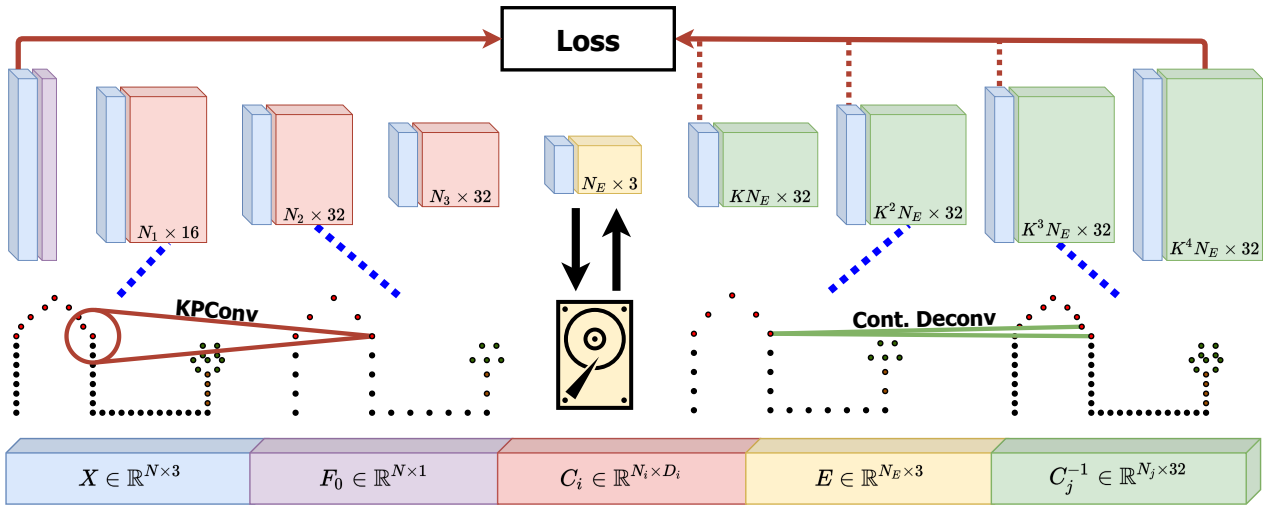

Fig. 2: Schematic overview of our proposed network. The encoder takes as input a point cloud $\mathcal{P} = \{\mathcal{X}, \mathcal{F}\}$, $|\mathcal{P}| = N_0$ and computes subsequentially for each subset a feature descriptor $C_i, i \in \{0, 1, 2\}$ (red) using kernel point convolutions. The last feature descriptor will be mapped (using an MLP) to the compact embedding space (yellow) to enable a memory-efficient representation $E \in \mathbb{R}^{N_E\times 3}$. This embedding E with its associated points can be used for storage or transmission and will later be used by the decoder to decompress the point cloud. The decoder consists of four deconvolutional layers which subsequentially upsample the point cloud (green). We use the feature of the last layer as a refinement of the last coordinates. The loss enforces a similar appearance of input and output. Additionally, we have a regularization term for each upsampled cloud (denoted by the dotted line) to be lower-resolution versions of the input.

III. Our Approach

Our goal is to learn for a small subset of points useful features from which we can recover the original point cloud. To this end, we propose an autoencoder structure for point cloud compression comprised of two parts. First, the encoder learns, from a given input data, a reduced and often more abstract representation – an embedding or code. The embedding is the input for the second part, the decoder that tries to reconstruct the original data using this compressed representation. By comparing the reconstruction with the original point cloud, the network can learn self-supervised parameters via backpropagation [33]. Compression is achieved as long as the embedding is smaller than the original point cloud.

Encoder Blocks. In our case, the input of the encoder is a dense 3D point cloud $\mathcal{P} = \{(\boldsymbol{x}_i, \boldsymbol{f}_i)\}$, $|\mathcal{P}| = N$, which consists of coordinates $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^3\}$ and point features $\mathcal{F} = \{\boldsymbol{f}_i \in \mathbb{R}^D\}$. The encoder uses multiple convolutional layers to learn local geometric features for each point. In Fig. 2 the coordinates $\mathcal{X}$ are denoted in blue and the learned descriptors $\mathcal{F}$ in red. Each layer reduces the number of points while increasing the receptive field. We use kernel point convolutions (KPConvs) [38], which directly operate on the features of the points themselves to avoid discretization effects caused by using grid-based representations such as voxel grids or octrees. The convolution of F at a point $\boldsymbol{x}$ with a convolutional kernel g is defined as a weighted sum of its neighboring features as

$$(F * g)(\boldsymbol{x}) = \sum_{(\boldsymbol{x}_i, \boldsymbol{f}_i)\in\mathcal{N}(\boldsymbol{x})} g(\boldsymbol{x}_i - \boldsymbol{x})\boldsymbol{f}_i, \qquad (1)$$

where $\mathcal{N}(\boldsymbol{x}) = \{(\boldsymbol{x}_i, \boldsymbol{f}_i) \in \mathcal{P} \mid \|\boldsymbol{x}_i - \boldsymbol{x}\| < r\}$ are the neighbors of $\boldsymbol{x}$ within radius r. The convolutional weights $\boldsymbol{W}$ are defined on M kernel points from which the weight of the neighbors $\mathcal{N}(\boldsymbol{x})$ are interpolated using first-order splines of size σ

$$g(\boldsymbol{y}_i) = \sum_{m<M} \max\left(0, 1 - \frac{\|\boldsymbol{y}_i - \hat{\boldsymbol{x}}_m\|}{\sigma}\right) \boldsymbol{W}_m, \qquad (2)$$

with the relative coordinate $\boldsymbol{y}_i = \boldsymbol{x}_i - \boldsymbol{x}$ and the weight $\boldsymbol{W}_m \in \mathbb{R}^{D_{\text{in}}\times D_{\text{out}}}$ of the m-th kernel point $\hat{\boldsymbol{x}}_m$. In each encoding block, we downsample the previous point cloud using grid-based subsampling. This returns for each occupied voxel the point which is the nearest to its center. It provides us with a homogenous point distribution without the risk of losing all points in a certain sparser area (as random sampling) or incur long sampling times (as for furthest point sampling). We use the same ResNet-like blocks as Hugues et al. [38]. The identity shortcuts in ResNet blocks enable a more direct gradient flow to earlier layers to reduce the risk of encountering instabilities due to vanishing gradients [11]. In contrast to skip connections between encoder and decoder, no additional information needs to be stored for the decompression. The last layer consists of an MLP to compress the features of size $\mathbb{R}^{N\times D_{\text{out}}}$ to the desired dimension $\mathbb{R}^{N\times D_{\text{emb}}}$.

Decoder Blocks. The task of the decoder is to reconstruct the original data from the embedding. Most encoder-decoder networks use skip connections [31], [23] from the encoder to the decoder to keep the high-frequency information. For compression, skip connections cannot be used, since it would require storing additional data from the encoder blocks for usage in the decoder. Hence, the whole signal must be encoded in the embedding to achieve effective compression. Therefore, we present a decoder block that does not depend on any skip connections but rather estimates the lost coordinates themselves. For a given point $(\boldsymbol{x}_i, \boldsymbol{f}_i)$, we

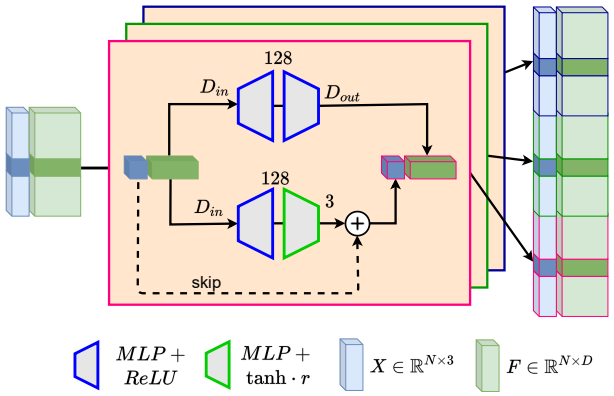

Fig. 3: Overview of our proposed deconvolution. Our deconvolutional kernel consists of two small MLP networks which serve as 1D convolutions. The first MLP (upper part) transforms the old feature $\boldsymbol{F}$ into a new feature space. The second MLP (lower part) predicts the offset to the new position $\boldsymbol{X}$ within the radius r. Depending on the number of deconvolutional kernels, in this example $K = 3$, we upsample the point cloud by a factor of K.

define the deconvolution C^{-1} by a set of K MLP layers, as depicted in Fig. 3. For each point, we obtain K new points $\{(\boldsymbol{x}_i^k, \boldsymbol{f}_i^k)\}, k \in \{0, \dots K-1\}$ within a given radius r by

$$\boldsymbol{x}_i^k = \boldsymbol{x}_i + r \cdot \Delta_k(\boldsymbol{f}_i), \tag{3}$$

$$\boldsymbol{f}_i^k = \Phi_k(\boldsymbol{f}_i). \tag{4}$$

Let us call Δ_k an offset block, which consists of an MLP with one hidden layer which uses a ReLU activation after the first layer and $\tanh$ after the second. The offset block Δ_k determines a coordinate increment by a nonlinear mapping of the feature space into the coordinate frame of the kernel $\Delta_k : \mathbb{R}^{D_{\text{in}}} \mapsto [-1, 1]^3$. The feature block $\Phi_k : \mathbb{R}^{D_{\text{in}}} \mapsto \mathbb{R}^{D_{\text{out}}}$ computes new features based on the old descriptor. Similar to the offset block Δ_k, the feature block Φ_k is also an MLP with one hidden layer but utilizes two ReLU activations instead. K offset and feature blocks form the proposed deconvolution $C^{-1} : \{\mathbb{R}^{N\times 3}, \mathbb{R}^{N\times D_{\text{in}}}\} \mapsto \{\mathbb{R}^{KN\times 3}, \mathbb{R}^{KN\times D_{\text{out}}}\}$ which will be applied to each point $\{(\boldsymbol{x}_i, \boldsymbol{f}_i)\}$ of the current layer. This upsamples the point cloud by a factor of K.

Architecture. Fig. 2 provides an overview of our proposed network architecture. We use 3 encoding blocks (KPConv) with grid-based subsampling followed by one MLP layer for compressing the embedding. Experiments show that a deeper encoder architecture leads to a lower decompression quality. We explain this behavior by vanishing gradients due to the absence of skip connections between the encoder and decoder. The radius of the convolution is equal to the resolution r_s of the sampling grid. This ensures that all points are part of at least one convolutional operation.

We estimate the relation between the grid resolution r_s and the sub-sampling rate $A(r_s)$ to upsample the point clouds to their original size

$$A(r_s) = \frac{a}{r_s^b}. \tag{5}$$

Further derivation and reasoning of the chosen power function (with the parameters a and b) are provided in Sec. IV.

We use the grid resolution r_s to control the compression rate. The sparser we sample, the higher the compression will be and the more points we need to upsample. Each KPConv has $3^3 = 27$ kernel points arranged in a grid with an influence radius of $\sigma = r/2$. The feature dimension (the red blocks in Fig. 2) for the KPConvs is $16, 32, 32$ respectively, and 3 for the embedding (yellow block). We saw that these relative compact feature descriptors are sufficient to store the geometric information about the neighborhood. If the original point cloud has no feature, we then use the occupancy value ($\boldsymbol{f}_i = 1$) as advocated by Thomas et al. [38]. The decoder consists of 4 deconvolutions to upsample the embedding to its original size. We use 32-dimensional point features and 128-dimensional hidden layer spaces in the deconvolution (expressed through the green blocks in Fig. 2). The upsampling factor K_i of the deconvolutional kernels is adapted to the sampling rate (see Sec. IV-E). Since the feature of the last layer is unused otherwise, we feed it to an MLP ($\mathbb{R}^{32} \mapsto \mathbb{R}^3$) to refine the coordinates.

Loss Function. We want to restore a point cloud, which is as similar as possible to the input. We use the Chamfer distance D_{CD} as a measure of similarity in our loss function $\mathcal{L}$. It is the average symmetric squared distance $\bar{d}^2$ of each point to its nearest neighbor in the other point cloud

$$D_{\text{CD}}(\mathcal{P}_{\text{in}}, \mathcal{P}_{\text{out}}) = \frac{\bar{d}^2(\mathcal{P}_{\text{in}}, \mathcal{P}_{\text{out}})}{2} + \frac{\bar{d}^2(\mathcal{P}_{\text{out}}, \mathcal{P}_{\text{in}})}{2}, \tag{6}$$

$$\bar{d}^2(\mathcal{P}_i, \mathcal{P}_j) = \frac{1}{|\mathcal{P}_i|} \sum_{\boldsymbol{x}_i \in \mathcal{P}_i} \min_{\boldsymbol{x}_j \in \mathcal{P}_j} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2, \tag{7}$$

where $\mathcal{P}_{\text{in}}$ denotes the input point cloud before compression and $\mathcal{P}_{\text{out}}$ the decompressed point cloud. Using the symmetric distance prevents the network from flooding the whole space with points but also from leaving out parts that have been present in the original point cloud. We add the Chamfer distances between the input point cloud and the output points $\hat{\mathcal{P}}$ of all deconvolutions as a regularization term to ensure valid intermediate results. The loss $\mathcal{L}$ is then given by:

$$\mathcal{L} = D_{\text{CD}}(\mathcal{P}_{\text{in}}, \mathcal{P}_{\text{out}}) + \beta \sum_j D_{\text{CD}}(\mathcal{P}_{\text{in}}, \hat{\mathcal{P}}_j), \tag{8}$$

where β is a weight to control the impact of the regularization term and we use $\beta = 0.2$ in all of our experiments. For a more detailed analysis of the regularization, see Sec. IV-E.

IV. Experimental Evaluation

In this section, we validate that our proposed algorithm is able to compress point cloud data efficiently. We compare our method to Draco [8] and the octree-based compression algorithm by Mekuria et al. [21], which we will refer to as "MPEG anchor". Additionally, we will show the results for a plain Octree [20] using binary flags to indicate if a leaf octant is occupied.

A. Experimental Setup

Our network is designed for compressing large-scale dense point clouds. We evaluate our method on the SemanticKITTI [2] dataset. For obtaining high accurate and

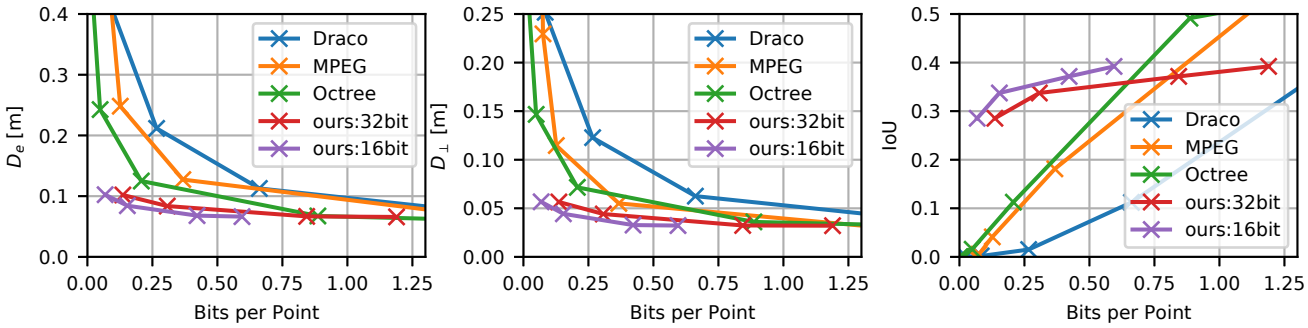

Fig. 4: Compression results on the test sequence 08 of the KITTI Vision Benchmark dataset. We use Draco [8], the MPEG Anchor from Mekuria et al. [21] and an own binary Octree implementation as baselines. Our approach can reconstruct the point cloud for the same amount of memory at a higher quality level.

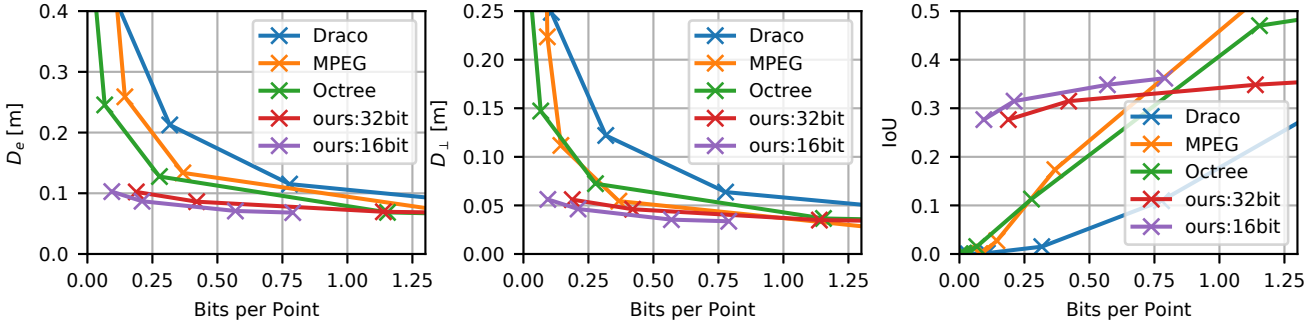

Fig. 5: Compression results on the nuScenes dataset. The model was still trained on SemanticKITTI to show the generalizability of our approach. The errors are slightly higher compared to the validation set of KITTI, but we are still able to outperform the baselines.

dense point clouds, we aggregate all scans using the ground truth poses and divide the map into patches of size $40 \times 40 \times 15\,\mathrm{m}^3$. The labels have been used to remove the dynamic objects which otherwise will lead to artifacts in the map. To remove redundant points in the original point cloud, we filter it using a voxel grid with a resolution of $10\,\mathrm{cm}^3$. We use sequence 00 to 10 (except 08) for training (see Fig. 6 first column). A small subset of the training data serves as a validation set and the complete sequence 08 is used for testing and comparison with the baselines.

Evaluation metrics. The quality of a compression algorithm is a trade-off between compression ratio and reconstruction error. For the compression ratio, we use the average bits per point (BPP) required for storing the encoding of the point cloud. We use 3 metrics for measuring the reconstruction error. Symmetric point cloud distances D_d are widely used for measuring the quality of the point cloud reconstruction. These metrics consist of two parts: the distance $\bar{D}_d$ from the ground truth point cloud $\mathcal{P}_{\text{in}}$ to the reconstruction $\mathcal{P}_{\text{out}}$ and vise versa

$$D_d(\mathcal{P}_{\text{in}}, \mathcal{P}_{\text{out}}) = \frac{\bar{D}_d(\mathcal{P}_{\text{in}}, \mathcal{P}_{\text{out}})}{2} + \frac{\bar{D}_d(\mathcal{P}_{\text{out}}, \mathcal{P}_{\text{in}})}{2}, \tag{9}$$

$$\bar{D}_d(\mathcal{P}_i, \mathcal{P}_j) = \frac{1}{|\mathcal{P}_i|} \sum_{\boldsymbol{x}_i \in \mathcal{P}_i} \min_{\boldsymbol{x}_j \in \mathcal{P}_j} d(\boldsymbol{x}_i - \boldsymbol{x}_j). \tag{10}$$

Thereby, the metric is sensitive to false positives (reconstructing points in unoccupied areas) and false negatives (leaving out occupied areas). The euclidean distance $D_e = D_d(\mathcal{P}_{\text{in}}, \mathcal{P}_{\text{out}})$ with $d = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$ is used as a metric for reconstructing the points itself. However, for some robotics applications (e.g. point-to-plane ICP), it is less important to reconstruct the exact same point than that it lies on the same surface. Therefore, we also show the symmetric plane distance $D_\perp = D_d(\mathcal{P}_{\text{in}}, \mathcal{P}_{\text{out}})$ with $d = |\boldsymbol{n}^T(\boldsymbol{x}_i - \boldsymbol{x}_j)|$. Where $\boldsymbol{n} \in \mathbb{R}^3$ denotes the ground truth normal of that point. The normals have been precomputed using the Eigenvalue decomposition of the covariance matrix from all points within a 50 cm radius around the query point.

The last metric is the intersection-over-union (IoU) between occupancy grids G for both point clouds. The IoU is here defined as

$$IoU = \frac{|G_{\text{in}} \cap G_{\text{out}}|}{|G_{\text{in}} \cup G_{\text{out}}|}. \tag{11}$$

The occupancy grids G_{in} and G_{out} have a resolution of $20 \times 20 \times 10\,\mathrm{cm}^3$ as used by Huang et al. [14].

Implementation details. We use the octree implementation by Behley et al. [3] for an efficient radius neighbor search in the KPConv-blocks. Our model is implemented in PyTorch [25] and trained on a GeForce RTX 2080 SUPER and with an Intel CPU with 3.5 GHz. We use the Adam optimizer [17] and the one cycle learning rate schedule proposed by Smith et al. [36] with a start learning rate of 10^{-6}, which will increase to 10^{-4} in the first 20 epochs and afterward decrease to 10^{-5}. We use the cosine annealing strategy and train for 200 epochs with a batch size of 3. We limit the number of input points to 30,000 points to speed-up the training and reduce the memory footprint while training. Nevertheless, we compare at test time our reconstructed point cloud to all available points. We train four

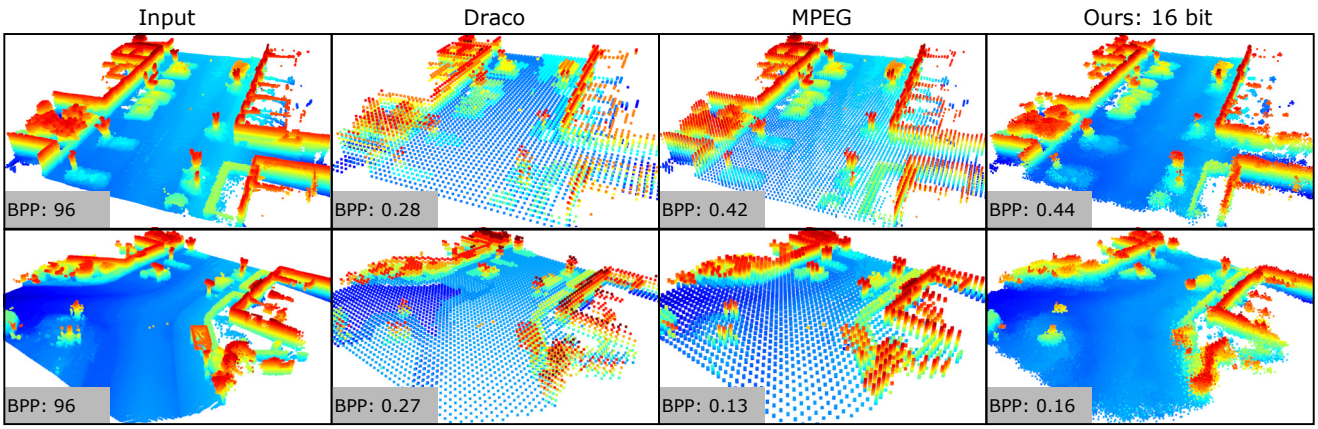

Fig. 6: Qualitative results of our proposed method and the baselines for different bits per point (BPP) on two example point clouds. The points are colored according to their height. Our approach is able to recover dense point clouds also when targeting low bit rates.

network architectures with varying sub-sampling resolutions $r_s \in \{3.0\,\text{m}, 2.0\,\text{m}, 1.2\,\text{m}, 1.0\,\text{m}\}$ for different compression levels. The encoder needs approximately 0.27 MB and the decoder between 0.30 MB and 0.62 MB memory storage (depending on the upsampling rates), which is a one-time investment and not depending on the number of compressed maps or points. We use Open3D [46] for visualization.

B. Compression Results

In this first experiment, we compare our compression results to the baselines to quantify the compression performance of our proposed method. Our approach stores the output of the encoder, namely a set of features and points, as the compressed representation. We will show the results for storing the embedding as 32 and 16 bit floating-point values. The compression results of our approach and the baselines on sequence 08 are presented in Fig. 4. Our proposed method outperforms the baselines in the distance-based metrics D_e and $D_\perp$, as well as in the IoU for the 16 bit representation. The small quality gain ($< 2\%$) of the 32 bit representation is disproportional concerning doubling the memory demand. The reduced density of the baselines leads to substantially higher errors than our approach which can reconstruct for each bit rate the same number of points. Our approach achieves over 2.4 times lower reconstruction errors for bit rates under 0.1 BPP compared to the baselines. We think that the worse performance of the MPEG Anchor [21] compared to the plain occupancy octree is due to some overhead for attribute compression and the ability to further compress tele-immersive data streams.

C. Generalization Capability

Learning-based methods often degrade when applied in a different environment due to over-fitting to the specific characteristics of the training set. We claim that our method generalizes well by learning the local geometries rather than remembering the global shape. To support this claim, we test our approach on a completely different dataset without retraining the model. Here, we use the nuScenes dataset [5], which not only has a different sensor setup (different height,

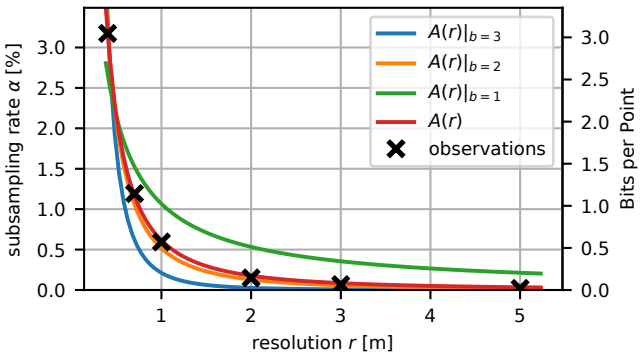

Fig. 7: We estimate the dependency between the subsampling rate α and the subsampling resolution r to predict the bit rate and to estimate the necessary upsampling rate for the deconvolution. We use least squares fitting to estimate the parameters of a power function $A(r) = a \cdot r^{-b}$ with parameters a and b.

field-of-view, and the usage of 32 beams instead of a 64-beam LiDAR), it is also recorded on a different continent which usually changes the appearance of the scenes quite substantially. Note that there are no labels for the nuScenes dataset available so that the dynamics will not be removed from the map, which makes it more challenging due to new artifacts it has never seen before. Fig. 5 shows that we still outperform the baselines for the 16 bit representation, even though the margin to the baselines got smaller.

D. Qualitative Analysis

In this part, we will analyze the decompression results qualitatively. In Fig. 6, we show the decompressed maps of the baselines and our approach. For the baselines, we have chosen the quality levels with the closest bit rates to our approach. The points are colored according to their height. As we can see, our approach is able to recover comparably dense point clouds even for varying compression rates. The point clouds of the baselines are sparser, especially when targeting low bit rates (see Fig. 6 second row). When reducing the bit rate, the decompressed maps of our approach get noisier while the baselines show larger quantization errors. Structures like the curbs of the streets are only visible in our denser maps.

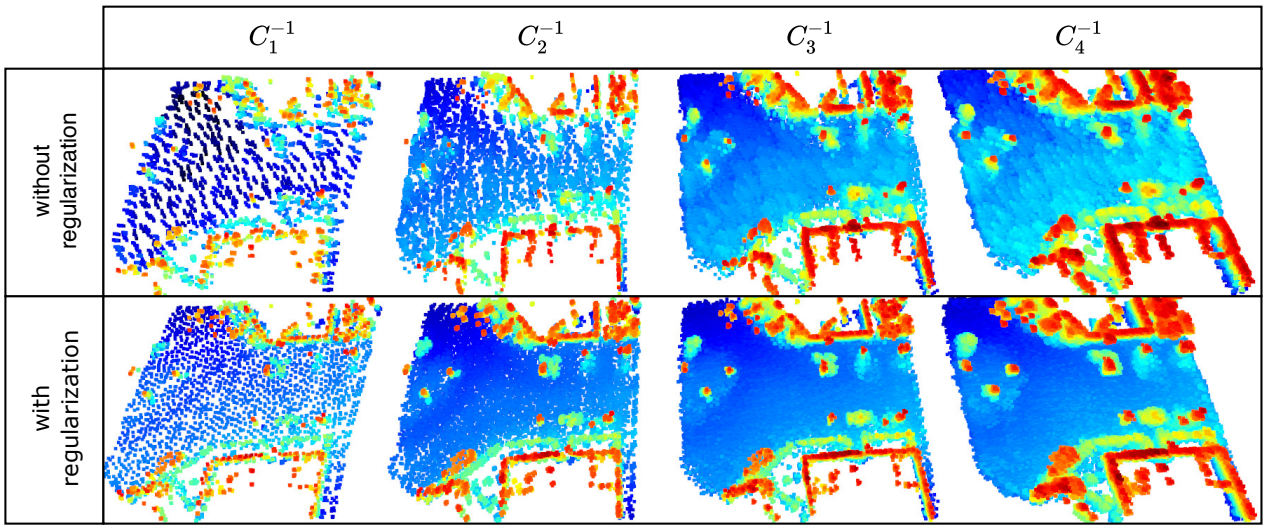

Fig. 8: Intermediate point clouds after each deconvolutional layer $C_j^{-1}, j \in \{1, 2, 3, 4\}$ for the same networks but with different loss functions. The network of the top is trained without the regularization term, whereas the bottom row is learned with the regularization term. The lack of the regularization leads to less homogeneous and more noisy point clouds after each deconvolution. When trained with the regularization, the intermediate point clouds can be used as lower resolution model.

E. Ablation Studies

In this part, we provide ablation studies to validate the choices made and to give a deeper understanding of the behavior of the network. We first investigate the adaptive sampling strategy and then the regularization.

Adaptive Sampling. Each encoding block reduces the number of points by grid-based subsampling and thus the number of embedding points depends on the resolution of the sampling grid. In this experiment, we investigate the influence of varying grid resolutions r_s and the subsampling rate α of the points. This enables us to adapt the up-sampling rate to ensure that the input and output point clouds have similar sizes. Additionally, it enables us to predict the bit rate BPP based on the resolution r_s or vise versa. Therefore, we subsample the point clouds of the training set with different grid resolutions r_s and compute the mean subsampling rate α. We fit a power function $A(r_s)$ which is proportional to the density distribution $\rho(r_s)$ of the point cloud

$$A(r_s) = \frac{a}{r_s^b} \propto \rho(r_s). \tag{12}$$

The parameter a denotes the magnitude of the density while the parameter b describes the dimensionality of the point distribution. A fix parameter $b \in \{1, 2, 3\}$ would correspond to a voluminous, planar, or linear distribution, respectively. The result of $A(r_s)$ as well as for fix $b \in \{1, 2, 3\}$ are shown in Fig. 7. The best fitted function is given by $a = 0.006$ and $b = 1.80$. The parameter $b = 1.80$ reflects the huge amount of planar surfaces (streets, walls, meadows...) in outdoor environments. Each deconvolutional block i up-samples the point cloud by the factor $K_i = \lceil A(r_e)^{-1/I} \rceil$, where I is the number of deconvolutional blocks and r_e is the sub-sampling resolution of the last encoding layer.

Impact of regularization. In this experiment, we show the importance of the regularization term. We train the same network two times, first with and second without regularization. In Fig. 8, we can see the qualitative impact on the point clouds after each upsampling step. The distribution of the points is more uniform and less noisy for the point clouds of the regularized network. The regularization term penalizes big point distances between the clouds. This leads implicitly to more homogeneous regions so that we do not see the necessity of an additional repulsion loss as in the PU-Net [44]. The quantitative results in Tab. I support our assumption that more meaningful intermediate point clouds help the network to reconstruct the data. Additionally, the intermediate point clouds can, with regularization, be used as lower-resolution versions of the point cloud.

TABLE I: Quantitative regularization impact

experiment	D_e	$D_\perp$	IoU
without regularization	0.110	0.062	0.327
with regularization	0.077	0.039	0.332

V. Conclusion

In this work, we presented a novel approach for lossy point cloud compression using a neural network. Our approach operates on dense large-scale maps of aggregated point clouds used for autonomous driving. Our method exploits recurring structures to learn a small set of feature descriptors using a deep convolutional autoencoder. The descriptor set serves as the compressed representation, which can be used for efficient storage and transmission, or later by the decoder to reconstruct the point cloud. Furthermore, we propose a 3D deconvolution that directly operates on the points themselves to avoid discretization effects from using voxel grids or memory size problems from skip connections. This allows us to successfully reduce the memory footprint of dense point clouds. We implemented and evaluated our approach on different datasets and provided comparisons to other existing techniques and supported all claims made in this paper.

REFERENCES

[1] T. Akenine-Möller, E. Haines, and N. Hoffmann. *Real-time Rendering*. A K Peters/CRC Press, 4th edition, 2018.

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[3] J. Behley, V. Steinhage, and A.B. Cremers. Efficient radius neighbor search in three-dimensional point clouds. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3625–3630, 2015.

[4] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] H. Caesar, V. Bankiti, A.H. Lang, S. Vora, V.E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A multimodal dataset for autonomous driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[6] J. Elseberg, D. Borrmann, and A. Nüchter. One billion points in the cloud – an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 76:76–88, 2013.

[7] T. Golla and R. Klein. Real-time point cloud compression. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 5087–5092, 2015.

[8] Google. Draco 3d data compression. *https://github.com/google/draco*, 2017. (26.08.2020).

[9] S. Gumhold, Z. Kami, M .Isenburg, and H-P. Seidel. Predictive point-cloud compression. In *ACM SIGGRAPH Sketches*, page 137. 2005.

[10] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 2020.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 34:189–206, 2013.

[13] Y. Hu, W. Yang, Z. Ma, and J. Liu. Learning End-to-End Lossy Image Compression: A Benchmark. *arXiv preprint arXiv:2002.03711*, 2020.

[14] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun. OctSqueeze: Octree-Structured Entropy Model for LiDAR Compression. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1313–1323, 2020.

[15] T. Huang and Y. Liu. 3D point cloud geometry compression on deep learning. In *Proc. of the ACM Intl. Conf. on Multimedia*, pages 890–898, 2019.

[16] Y. Huang, J. Peng, C-C. Kuo, and M. Gopi. Octree-Based Progressive Geometry Coding of Point Clouds. *In Proc. of the Symp. on Point-Based Graphics (PBG)*, 6:103–110, 2006.

[17] D.P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2016.

[18] I. Kweono, M. Hebert, E. Krotkov, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 997–1002, 1989.

[19] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[20] D. Meagher. Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer. *Rensselaer Polytechnic Institute Image Processing Laboratory, Technical Report*, (IPL-TR-80-111), 1980.

[21] R. Mekuria, K. Blom, and P. Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Trans. on Circuits and Systems for Video Technology (TCSVT)*, pages 828–842, 2016.

[22] B. Merry, P. Marais, and J. Gain. Compression of dense and regular point clouds. In *Proc. of the Intl. Conf. on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 15–20, 2006.

[23] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[24] F. Nenci, L. Spinello, and C. Stachniss. Effective Compression of Range Data Streams for Remote Robot Operations using H.264. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, Natalia N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 8026–8037. 2019.

[26] C.R. Qi, H. Su, K. Mo, and L.J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[27] C.R. Qi, K. Yi, H. Su, and L.J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[28] M. Quach, G. Valenzise, and F. Dufaux. Learning convolutional transforms for lossy point cloud geometry compression. In *Proc. of the IEEE Intl. Conf. on Image Processing (ICIP)*, pages 4320–4324, 2019.

[29] M. Quach, G. Valenzise, and F. Dufaux. Improved Deep Point Cloud Geometry Compression. *Proc. of the IEEE Intl. Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2020.

[30] G. Riegler, A. Ulusoy, and A. Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[31] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[32] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of compact 3d models based on the detection of recurrent structures. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2137–2142, 2010.

[33] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning Representations by Back-Propagating Errors*, pages 696–699. 1988.

[34] R. Schnabel and R. Klein. Octree-based Point-Cloud Compression. In *Proc. of the Symposium on Point-Based Graphics*, 2006.

[35] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. In *Computer Graphics Forum*, pages 214–226, 2007.

[36] L. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, 2019.

[37] X. Sun, H. Ma, Y. Sun, and M. Liu. A novel point cloud compression algorithm based on clustering. *IEEE Robotics and Automation Letters (RA-L)*, 4(2):2132–2139, 2019.

[38] H. Thomas, C R. Qi, J. Deschaud, B. Marcotegui, F. Goulette, and L.J. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[39] R. Triebel, P. Pfaff, and W. Burgard. Multi-level Surface Maps for Outdoor Terrain Mapping and Loop Closing. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[40] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda. Point cloud compression for 3D LiDAR sensor using recurrent neural network with residual blocks. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3274–3280, 2019.

[41] P. Wang, Y. Liu, Y. Guo, C. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Trans. on Graphics (TOG)*, 36(4):1–11, 2017.

[42] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[43] L. Yongcheng, F. Bin, M. Gaofeng, L. Jiwen, X. Shiming, and P. Chunhong. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, pages 5239–5248, 2019.

[44] L. Yu, X. Li, C. Fu, D. Cohen-Or, and P. Heng. Pu-net: Point cloud upsampling network. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2799, 2018.

[45] C. Zhang, D. Florencio, and C. Loop. Point cloud attribute compression with graph transform. In *Proc. of the IEEE Intl. Conf. on Image Processing (ICIP)*, pages 2066–2070, 2014.

[46] Q. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.