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.

Index Terms—Mapping, Deep Learning Methods

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 [21] or kD-trees [4], utilize a hierarchical data representation that allows for efficient storage and fast neighbor queries. Voxel grids [11] 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 [19], [41] 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 [37], but such predefined geometries may not fully exploit the available potential of recurring structures. Therefore, learning-based representations often try to capture common characteristics of the scene for further compression [34]. Recent deep neural autoencoder networks [13], [15], [31] 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. [22]. 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 [28] 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++ [29] 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. [45] extend this idea by a multi-level feature aggregation to increase the contextual information.

Grid-based approaches [20], [44] compute local features using 3D convolutions on a voxel grid. Octree-based approaches [32], [43] 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. [40]. 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. [46] propose a network to upsampling 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 [21] 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], [36]. 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 [47]. Other approaches focus on an iterative prediction of neighboring points using spanning trees [9], [24]. 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 [23], [26], [17], [9], [22]. 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 [30], [31] 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.

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 due to vanishing gradients [11]. In contrast to skip connections but rather estimates the lost coordinates themselves. For a given point $x_i$ with a convolutional kernel $g$ is defined as a weighted sum of its neighboring features as

$$ (F * g)(x) = \sum_{(x, f_i) \in \mathcal{N}(x)} g(x_i - x) f_i, \tag{1} $$

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

$$ g(y_i) = \sum_{m < M} \max \left(0, 1 - \frac{|y_i - \hat{x}_m|}{\sigma}\right) W_m. \tag{2} $$

with the relative coordinate $y_i = x_i - x$ and the weight $W_m \in \mathbb{R}^{D_{\text{out}}}$ of the $m$-th kernel point $\hat{x}_m$. 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. [40]. 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 $\mathbb{R}^{N \times D_{\text{out}}}$. 

Decoder Blocks. The task of the decoder is to reconstruct the original data from the embedding. Most encoder-decoder networks use skip connections [33], [25] 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 $(x_i, f_i)$, we 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 $\{(x^k_i, f^k_i), k \in \{0, \ldots, K - 1\}$
The radius of the convolution is equal to the resolution. The absence of skip connections between the encoder and decoder. The encoder architecture leads to a lower decompression quality. We use 3 encoding blocks (KPConv) upsamples the point cloud by a factor of 4. The offset block \(\Delta_k\) to the new position. The upsampling factor \(K\) 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 \(d^2\) of each point to its nearest neighbor in the other point cloud

\[
D_{CD}(P_m, P_{out}) = \frac{d^2(P_m, P_{out})}{2} + \frac{d^2(P_{out}, P_m)}{2},
\]

where \(P_m\) denotes the input point cloud before compression and \(P_{out}\) 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 \(\hat{P}\) of all deconvolutions as a regularization term to ensure valid intermediate results. The loss \(\mathcal{L}\) is then given by:

\[
\mathcal{L} = D_{CD}(P_m, \hat{P}_{out}) + \beta \sum_j D_{CD}(P_m, \hat{P}_j),
\]

where \(\beta\) 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 [3] and the octree-based compression algorithm by Mekuria et al. [22], which we will refer to as “MPEG anchor”. Additionally, we will show the results for a plain Octree [21] 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 dense point clouds, we aggregate all scans using the ground truth poses and divide the map into patches of size \(40 \times 40 \times 40\).
15 m³. 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 cm³. 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 $D_\text{d}$ are widely used for measuring the quality of the point cloud reconstruction. These metrics consist of two parts: the distance $D_\text{d}$ from the ground truth point cloud $P_\text{in}$ to the reconstruction $P_\text{out}$ and vice versa:

\[
D_\text{d}(P_\text{in}, P_\text{out}) = \frac{D_\text{d}(P_\text{in}, P_\text{out})}{2} + \frac{D_\text{d}(P_\text{out}, P_\text{in})}{2},
\]

\[
D_\text{d}(P_i, P_j) = \frac{1}{|P_i|} \sum_{x_i \in P_i} \min_{x_j \in P_j} d(x_i - x_j).
\]

Thereby, the metric is sensitive to false positives (reconstructing points in unoccupied areas) and false negatives (leaving out occupied areas). The euclidean distance $D_\text{e} = D_\text{d}(P_\text{in}, P_\text{out})$ with $d = \|x_i - x_j\|_2$ 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 $D_\perp = D_\text{d}(P_\text{in}, P_\text{out})$ with $d = |n^T(x_i - x_j)|$. Where $n \in \mathbb{R}^3$ 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:

\[
\text{IoU} = \frac{|G_\text{in} \cap G_\text{out}|}{|G_\text{in} \cup G_\text{out}|}.
\]

The occupancy grids $G_\text{in}$ and $G_\text{out}$ have a resolution of $20 \times 20 \times 10$ cm³ 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 [27] and trained on a GeForce RTX 2080 SUPER and with an Intel CPU with 3.5 GHz. We use the Adam optimizer [18] and the one cycle learning rate schedule proposed by Smith et al. [38] 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 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 \cite{Zhang2018} for visualization.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig6}
\caption{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.}
\end{figure}

\section{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 $D_{ld}$, 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 \cite{Sun2017} compared to the plain occupancy octree is due to some overhead for attribute compression and the ability to further compress tele-immersive data streams.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig7}
\caption{We estimate the dependency between the subsampling rate $\alpha$ 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 $b$.}
\end{figure}

\section{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 \cite{Caesar2020}, which not only has a different sensor setup (different height, 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.

\section{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.
A result of a voluminous, planar, or linear distribution, respectively. The resolution of the last encoding layer.

Fig. 7. The best fitted function is given by

\[
A(r_s) = \frac{a}{r_s^b} \propto \rho(r_s).
\]

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 \(K_i = [A(r_s)_{\text{c}}]^{-1/1}\), where \(I\) is the number of deconvolutional blocks and \(r_c\) is the sub-sampling rate 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.

<table>
<thead>
<tr>
<th>experiment</th>
<th>(D_e)</th>
<th>(D_\perp)</th>
<th>IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>without regularization</td>
<td>0.110</td>
<td>0.062</td>
<td>0.327</td>
</tr>
<tr>
<td>with regularization</td>
<td>0.077</td>
<td>0.039</td>
<td>0.332</td>
</tr>
</tbody>
</table>

**TABLE I: Quantitative regularization impact**

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 [46]. The quantitative results in Tab. 1 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.

### 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 data. 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.