A Survey on Point Cloud Compression Using Deep Learning Approaches

Reetu Hooda, Alexandre Zaghetto, Danillo B Graziosi and Ali Tabatabai
Sony Corporation Of America
San Jose Lab, California

***Abstract*—Deep learning have been widely used for solving several data processing tasks and recently drawn attention into data compression domains as well, notably for point cloud (PC). Compression techniques based on deep learning (DL) methods such as convolutional neural network (CNN) has given flexibility of exploiting higher dimensional correlations for improved performance. The most common DL based choice for point cloud compression (PCC) is an autoencoder. Although there are few implementation that uses recurrent neural network (RNN) and fully connected neural network. This paper covers the on-going research on PCC using DL approaches. The benchmark datasets with performance metrics are also included. It also discusses recently introduced machine learning (ML) techniques to solve PC related problems such as shape classification, object detection and tracking, cloud segmentation etc. The results of the methods included shows performance comparable to conventional coding paradigm thus offering promising improvements in the future.**

***Index Terms*—Point cloud compression, Deep Learning, Autoencoders, Convolutional neural network.**

I. Introduction

A Point cloud is a collection of unorganized set of points in 3D space typically represented with geometry positions as 3D coordinates (x, y, z). The coordinates can also have associated attribute values such as color information or normal vectors. Point cloud is considered as one the powerful 3D visual representation of the data that provides very realistic and interactive experience to the users. They provide multiple viewpoints of an object and are extensively used in diverse applications such as autonomous driving, geographical information system, robotic applications, cultural heritage to preserve historical architecture etc.

To efficiently represent the details entailing a 3D scene for instance, enormous amount of information is required. As the precision of the PC increases (i.e., increment in the no. of points and multiple attributes associated with it), the amount of memory to store and the bandwidth requirements for transmission also increases. Therefore, to enable practical usage of PC in aforementioned applications, efficient PCC techniques holds utmost importance.

In the last decade, DL have become an emerging field offering solutions to image and video data related problems including compression with encouraging results that are comparable to traditional methods. The non-unoformity of PC data scattered in space makes it difficult to compress. Hence, the current most relevant methods convert 3D data into 2D format for efficient compression. Although, DL provides the flexibility of exploiting the 3D correlations for potential improvements. In this context, the main goal of this paper is to present the recent advancements in PCC using DL approaches. It also includes the benchmark datasets with performance metrics used for evaluation of a contribution. We also present ML based solutions to solve other PC related problems.

Two fundamental problems of PCC are categorized as follows:

1) Geometry Compression: It focuses on compressing the locations of the points.
2) Attribute Compression: Aims at exploiting the redundancy in attribute values given the locations of the points.

Among these two, geometry compression is actively researched that are adopting DL approaches and most recent ones will be discussed in this paper.

The taxonomy of schemes covered in this paper is shown in Fig 1.

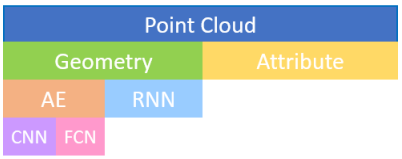


Fig. 1. Taxonomy of the proposed schemes

The rest of the paper is organized as follows: Section II discusses all the various autoencoder implementations for compression of point cloud and their merits and demerits will be evaluated. Fully connected neural network approach is presented in section III and recurrent neural network (RNN) based approach is addresed in section IV. Section V briefly assess the ML approaches for PC centric tasks and section VI concludes the paper.

II. Autoencoders

CNNs are known to be one of the most effective neural network (NN) in extracting useful features in uniform structure sharing a pattern/correlation. These charcteristics are mostly to be found in images and video resulting in wide usage of CNNs. And hence, most recently proposed DL based PC geometry compression techniques adopt autoencoder (AE) neural network design involving convolutional layers. This section describes some of the recently proposed PC geometry coding schemes using CNN based autoencoders.

A. Point Cloud Geometry Autoencoder (PCG-AE)

PCG-AE proposes a basic implementation of DL approach for PCC to highlight the potential improvements of AE PC coding paradigm. The general 3D coordinate representation of PC is changed to set of binary 3D blocks format where occupied voxels are signalled by a '1' and '0' otherwise. Since points in PC are generally arranged in disorderly fashion, coding them using NN can be very challenging. Then this type of formatting introduces some correlation similar to images and video to be exploited by a CNN based AE in 3D domain. The underlying idea is to use an adaptive forward and inverse transform instead of using a fixed predefined transform such as DCT [1].

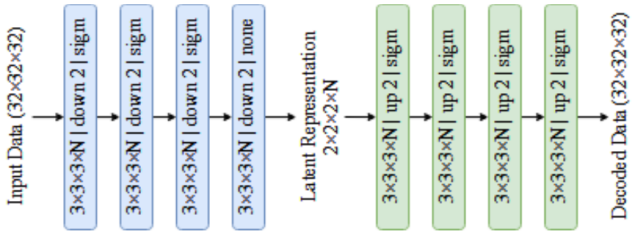


Fig. 2. CNN based AE (adaptive transform)

The details of the structure of transform is shown in Fig 2. Fist, the input PC geometry is divided into 3D binary voxeled blocks of size $32 \times 32 \times 32$ and the position of the 3D sub-blocks in original PC are sent as index in the final bitstream. Then AE is used to map PC geometry into feature space and is addressed as latent representation as shown in Fig 3. So, the original 3D geometry is now represented by a latent representation at the end of the encoding process which will have reduced dimentionality. Finally, it is quantized and entropy coded to generate the final bitstream.

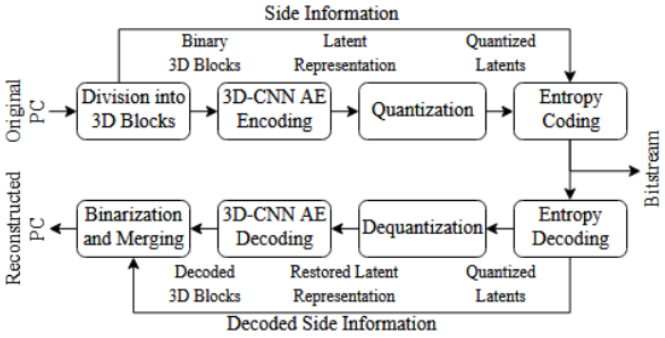


Fig. 3. Overall architecture of PCG-AE

At the decoder, restored latent representation is fed to the AE with an aim of reconstructing the geometry as close as possible to the original geometry. The output of AE decoder are binarized using threshold of 0.5 to reconstruct the 3D block. And the obtained 3D block are placed in their respective positions using the indexes in side information to reconstruct the entire PC.

Adaptive Transform: This implementation uses a very basic 3D CNN design which is an extension of the one typically used for 2D CNN image processing. It has 4 convolutional layers at the encoder and 4 at the decoder each with width, depth and height of 3 and a stride of 2 to downsample and reduce the dimensionality of the latent representation. N represents the no. of filters which is a hyperparameter and four models were trained for values: 32, 64, 96 and 128. N decides the reconstruction quality with the associated bitrate and hence effects the RD performance. N is tuned experimentally. All CNN layers use sigmoid as the activation function. The decoder has a symmetric structure with downsampling replaced with upsampling by a factor of 2.

The training of encoder and decoder is performed simultaneously. Point clouds from MPEG datasets were used for training and testing. Point cloud library (PCL) codec was set as the benchmark to evaluate the performance of the proposed scheme. RD performance with ratio of reconstructed points and original points were used as the performance metrics. It was concluded that using more filters (N) achieves higher quality as expected. Although for lower bitrates, PCL performs better than PCG-AE but in case of medium and higher bitrates, PCG-AE outperforms PCL by significant margin.

B. RD Control Through Implicit and Explicit Quantization

The extension of PCG-AE have been very recently proposed to provide flexible RD control functionality. Usually RD loss function (more specifically Lagrangian multiplier λ) defined in 1 is used to control RD trade-offs in DL-based coding solutions [2].

$$Loss = dist + \lambda \times rate \tag{1}$$

where $dist$ is the distortion between the original PC and reconstructed PC. $rate$ is the entropy of the quantized latent representation. RD control is achieved using the following:

1) The tuning between entropy of latent representation discussed in the previous section and reconstruction error through λ , this approach works as an implicit quantization.
2) Traditional transform-based coding methods use a parameter Quantization step (QS) to achieve different RD points and operates as explicit quantization.

Although The implicit quantization approach requires training of multiple DL models for each desired RD point but performs substantially better than explicit quantization. The scheme in [2] suggests to use combination of implicit and explicit quantization which reduces the training complexity and memory requirement to reach multiple RD points when compared to implicit-only quantization coding. Therefore, RD tradeoffs is achieved using small number of trained DL models.

DL model depicted in Fig. 4 uses loss function defined in 1 for training. Models trained for different value of λ allows us to achieve various RD points without using explicit quantization. The new PC uses the trained model for compression.

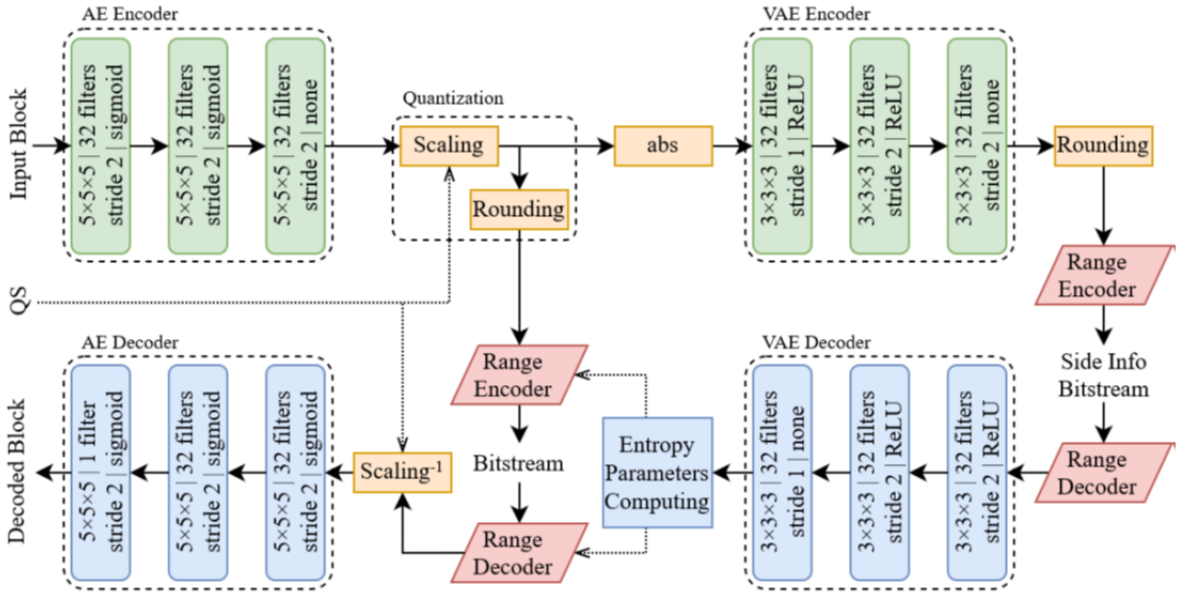


Fig. 4. DL-based PC coding architecture using explicit quantization using QS

Similar to the approach mentioned in previous section, latent representation is entropy coded after quantization. But to have more efficient entropy coding VAE is used to adapt the entropy model to current latent data.

When model shown in Fig. 4 is trained for $QS = 1$,it corresponds to implicit quantization even though we are explicitly specifying the amount of quantization we desire. RD control can be achieved using higher value of QS ($QS > 1$ i.e., explicit quantization) to generate multiple RD points using the same DL model. The $Scaling$ block down-scales the latent representation by QS which is further entropy coded. At the decoder, the $Scaling^{-1}$ block up-scales the latent values by the same QS. And the entropy model adapts to a certain value of QS.

Now, to reach multiple RD points, single trained model can be used with

1) QS = 1 (implicit quantization)
2) QS >1 (explicit quantization)

In addition, implicit-explicit balance is to be made since the model was initially trained for QS = 1 and higher QS may incorporate some distortion. This happens because there is a certain limit to the no. of RD points generated using single trained DL model.

Performance Evaluation: In this approach PCs were divided into blocks of size $64 \times 64 \times 64$ unlike PCG-AE. The QS value was varied from 1 to 20. And for implicit-only scheme, λ was varied from 50 to 20000. It was observed that single model was unable to reach lower bitrate (as QS increases) when trained for higher bitrate and was unable to reach higher bitrate (as QS decreases) when trained for lower bitrate. Hence, implicit-explicit quantization scheme was not able to cover wide range of quality and rates unlike multiple DL models trained using implicit-only quantization. Therefore, to address this issue 3 models were used instead of a single model to cover wide range of bitrates. Implicit-explicit quantization with fairly fewer models ($\lambda = 100, 1000, 10000$) models could achieve similar performance (for some cases slightly better) as that of using 10 trained models ($\lambda = 100, 250, 500, 600, 750, 1000, 2000, 3000, 10000, 20000$) with implicit-only quantization.

C. Learned Convolutional Transform (LCT)

LCT approach uses 3D convolutional autoencoder which directly operates on voxels. The decoding is interpreted as a classification problem to predict/infer the occupancy of the voxel by defining the geometry as a binary signal across the voxel grid. The architecture of the model is depicted in Fig. 5, where f_a and f_s are the analysis and synthesis transform respectively. N specifies the no. of filters (which is 32 in the proposed scheme) following the notation s^3 that specifies the filter size and strides in each direction [3].

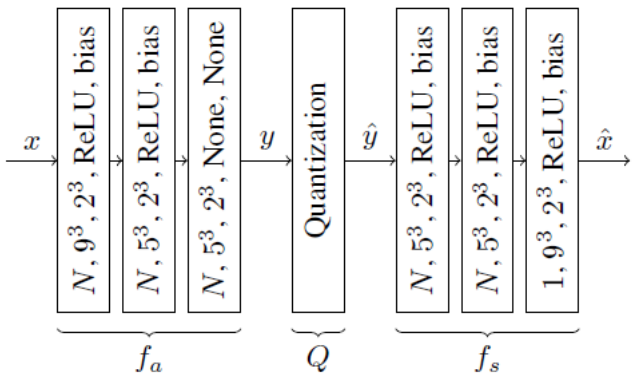
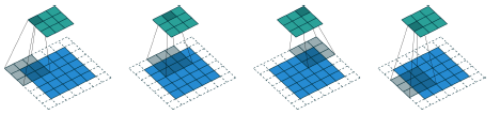
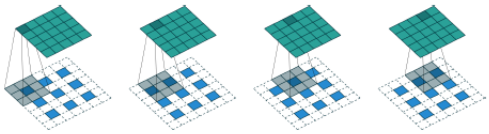


Fig. 5. AE architecture (no. of filters, filter size, strides, activation, bias)

The latent values are represented as $y = f_a(x)$ which is further quantized using unifrom quantization to obtain $\hat{y} = Q(y)$ and is decompressed to get $\hat{x} = f_s(\hat{y})$. f_a and f_s are learned during training process. The encoder and the decoder uses convolutions and transpose convolutions respectively with $same$ padding and stride shown in Fig. 6.



(a) Strided convolution on a 5^2 input with a 3^2 filter, 2^2 strides and same padding. The shape of the output is 3^3.



(b) Strided transpose convolution on a 3^2 input with a 3^2 filter, 2^2 strides and same padding. The shape of the output is 5^2.

Fig. 6. Strided convolution and strided transpose convolution operations.

The scheme does not use any entropy coder, instead uses a combination of LZ77 and Huffman coding to compress $Q(y)$. while considering decoding as a classification problem, due to the sparsity of the point cloud, an imbalance is created between the filled and empty voxels which is addressed using α-balanced focal loss. But the final loss is same as defined in 1. To have differentiable loss during training, rate is calculated differently while training and testing:

TABLE I
RATE CALCULATION

Training	Testing
To allow for differentiation uniform noise is added in place of discretization	Rate is no. of bits of final final compressed data since data is quantized
Q(y) has continuous probability distribution	Q(y) has discrete probability distribution
Continuous (differential) entropies	Discrete entopies

Differential entropy are used as approaximations od discrete entropy.The results showed that the model was able to generalize well even though it was trained using ModelNet40 mesh dataset (to cover wide range of variety and quantity) and tested on completely different dataset (MVUB-Microsoft Voxelized Upper Bodies dataset). The proposed scheme outperforms MPEG anchor for all the sequence and at all bitrates. For the similar bitrates, it achieves lower distortion and reconstructs more points compared to MPEG anchor, resulting in better quality. Different λ values $(10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6})$ were used to generate RD points and different octree depths for MPEG anchor. It improved the Bjontegaard-delta bitrate (BDBR) by 51.5% on average for the testing data.

D. Learned Point Cloud Geometry Compression (LPCGC)

Soon after LCT was proposed, it became relevant literature to other new approaches. One of them was Learned PCGC that uses DNN based variational autoencoder. The model architecture is illustrated in Fig. 8.

Preprocessing: This module involves three step, Voxelization, Scaling and partition. *Voxelization* is an optional step and is skipped if PC is already in 3D volumetric presentation. In a 3D Cartesian coordinate system, a voxel is set to 1 if it is occupied and 0 otherwise. This induces the inter-voxel correlation which can be exploited by 3D convolutions to generate very compact latent representation [4].

Similar to image downsampling used in image and video compression, it was extended to PCC in order to maintain the quality at lower bitrates. *Scaling* is incoporated to reduce the sparsity and hence increase the density of the PC. It is shown to improve the efficiency by considerable amount, especially for sparser PC such as category 3 of MPEG dataset. The scaling is performed as follows:

$$\hat{X_n} = round(X_n \times s) \tag{2}$$

where $(X_n \times s) = (i_n \times s, j_n \times s, k_n \times s)$ and X_n is the original PC geomtery for $n = 1.....N$. X_n is scaled by s where $s < 1$. Then the PC is partitioned into 3D non-overlapping cubes of size $W \times W \times W$. The respective position of the valid cube (at least one occupied voxel) with the no. of occupied voxels in each cube is signaled explicitly which adds a very small amount of overhead (metadata).

The 3D cubes are fed to the Analysis transform comprising of 3D stacked CNNs to generate a compact latent representation of size *(channel, length, width, height)* defined as $y = f_e(x; \theta_e)$ where θ_e are convolutional weights. For decoding, Synthesis transform is designed that uses quantized latent values $\hat{y}$ to reconstruct the cubes, formulated as $\tilde{x} = f_d(\hat{y}; \phi_d)$ where ϕ_d are the parameters.

The basic 3D convolutional unit in the transforms used is Voxeption-Resnet (VRN) because of its efficiency obtained due to to residual and inception network. Its structure is depicted in Fig. 7

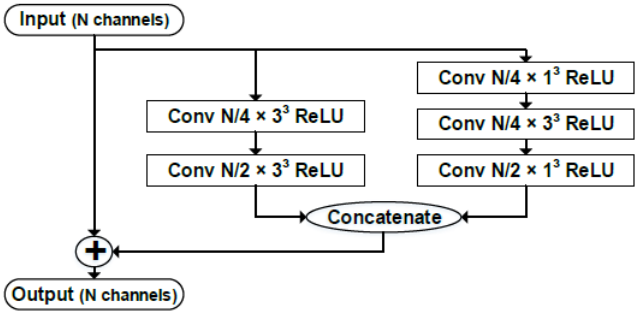


Fig. 7. Voxeption-ResNet Blocks (VRN)

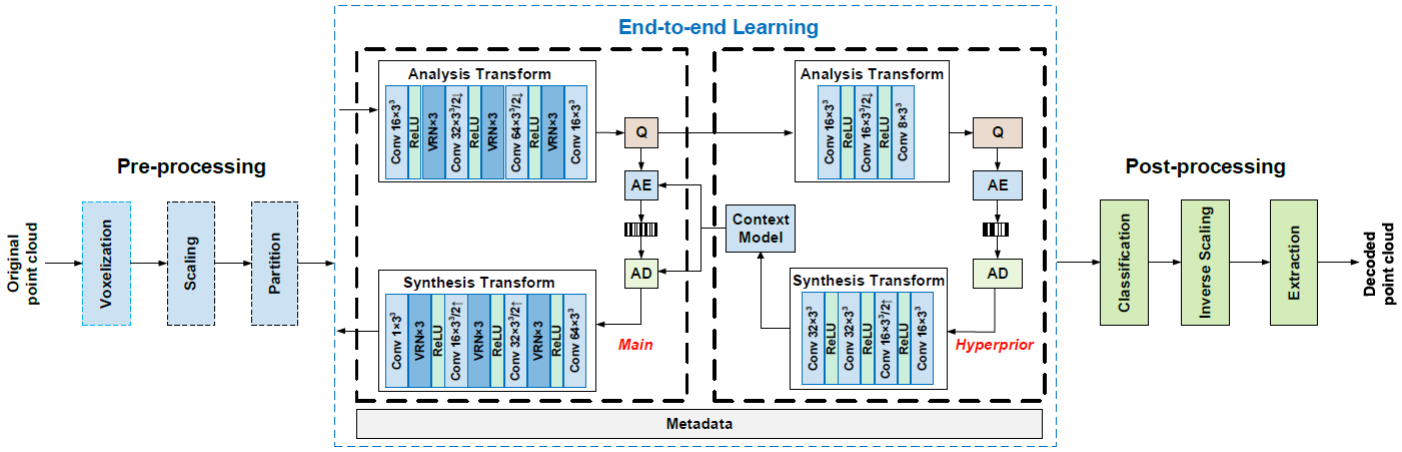


Fig. 8. Overall architecture of Learned PCGC

The scheme uses fairly smaller kernels $(3 \times 3 \times 3)$ offering lower complexity. Hyperprior is used for entropy modeling similar to the LCT approach.

Post-processing: comprises of classification, inverse scaling and extraction. Even though the median value of threshold is 0.5 but is shown to be not optimum and hence, the *Classification* step uses adaptive threshold to infer whether the voxels in the reconstructed cube $\tilde{x}$ is present or not. The classification of the voxels in the cube is also influenced by the no. of occupied points in the original cube. This information sent as the metadata during encoding. *Inverse scaling* is implemented as the next step which factors the reconstructed geometry by $1/s$ for rendering and display. And *Extraction* is an optional step that changes the volumetric representation to raw file format such as ply or ASCII.

For model generalization, the scheme used ShapeNet as the training dataset and MPEG GPCC and JPEG pleno standardization datasets were used for testing. The models are trained from end to end for each bitrate by changing λ (Lagrangian) and scaling factor s. Simulation results show that learned PCGC outperforms GPCC (octree) by 77% & 69%, GPCC (trisoup) by 67% & 62%, and it offers 88% & 82% gain for D1 (point to point) and D2 (point to plane) respectively.

III. Fully Connected Neural Network (FCN)

A. Deep AE-based PCGC

This scheme focuses on design of a codec that adapts to the features of PC for improved compression efficiency. The architecture of the proposed network is shown in Fig. 9 . It comprises of four main modules: a pointnet based encoder that acts as an analysis transform, a uniform quantizer, an entropy estimation block in the middle and the decoder that acts as a non-linear synthesis transform. The geometry of a PC is represented as 3D points consisting of let's say n points. These n points are downsampled first, to reduce the sparsity of original PC. Sampling layer will result in duplicate points which are merged together and results in lesser no. of points (m) which are the subset of original sampled n points [5].

Next, the m points are fed to the encoder which comprises of five 1D convolutional layers with kernel size of 1, followed by a $ReLU$ and batch normalization layer. The filters in each layer is 64, 128, 128, 256 and k respectively where k is decided based on the no. of input points.Then there is a max-pooling layer used to produce k-dimensional latent code z. The latent code is quantized and encoded by the entropy coder.

The decoder uses a FCN which decodes the latent code z using 3 fully-connected layers with hidden layers of 256, 256 and $n \times 3$ neurons to generate $n \times 3$ reconstructed PC. This whole process can be summarized using the following equation:

$$z = D(Q(E(S(x)))) \tag{3}$$

where S, E, Q and D are sampling layer, encoder, quantization and decoder respectively. And analysis and synthesis transform are formulated as $z = f_e(x; \theta_e)$ and $\hat{x} = f_d(y; \phi_d)$ respectively, where x is the original PC, z is the compressive representation and $\hat{x}$ is the reconstructed PC. $\theta_e \& \phi_d$ are the trainable parameters.

Quantization using rounding function makes the derivative 0 or undefined resulting in the loss to be non-differentiable. Therefore, the proposed scheme replaced quantization by additve uniform noise.

$$[f(x)] \approx f(x) + u \tag{4}$$

where u is the random noise. The distortion is computed using chamfer distance defined as follows:

$$d_{CH}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2 + \sum_{y \in S_2} \min_{x \in S_1} ||x - y||_2^2 \tag{5}$$

where S_1 is the original PC with n points and S_2 is the reconstructed PC with m points. The proposed scheme uses

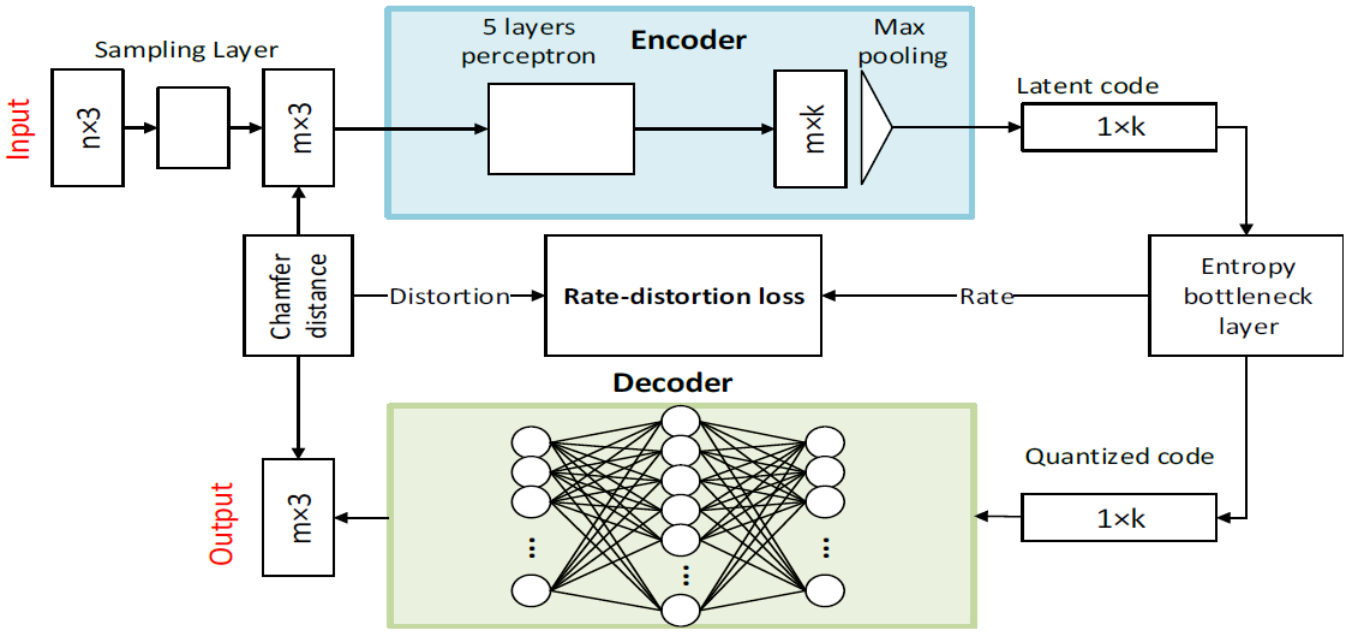


Fig. 9. Deep AE-based architecture

ShapeNet as the training and testing swt from a single class with 90% and 10% split. The results are compared with MPEG anchor (TMC13). The model uses data from four categories: chair (train: 6101, test: 678), (train: 6101, test: 678) airplane (train: 3640, test: 405), table (train: 7658, test: 851) and car (train: 6747, test: 750). Deep AE-based PCGC achieves BD rate gain og 73.15% on average over TMC13.

B. FoldingNet: PC AE via deep grid deformation

A CNN requires its neighbors at a certain distance sharing fixed spatial relation. Although PC does not exhibit such characteristics in raw format, voxelization is performed to make convolution operation on PC more meaningful. But voxelization sacrifices the details of the original representation of the PC. There are very approaches that work on the original original PC without voxelization. One of them is FoldingNet which is recently proposed. It is a MLP based AE which uses a 2D grid structure to construct a PC through Folding operation [6].

Fig. 11 shows the FoldingNet architecture. The encoder comprises of MLP sand graph based max-pooling layers. The geometry of the PC $n \times 3$ (where n is the no. of points) is the input to the encoder. Then the local covariance matrix of size 3×3 for each point v is computed using the coordinates that are one hop neighbor of v and vectorized to 1×9. The covariances ($n \times 9$) and geometry ($n \times 3$) are concatenated to a matrix of size $n \times 12$ and fed to the 3 layer perceptron. The output of the perceptron is fed to two consecutive graph layers to eventually generate a codeword (1×512) which maps a high-dimensional embedding of an input PC. The codeword is replicated m times and concatenated with fixed 2D grid

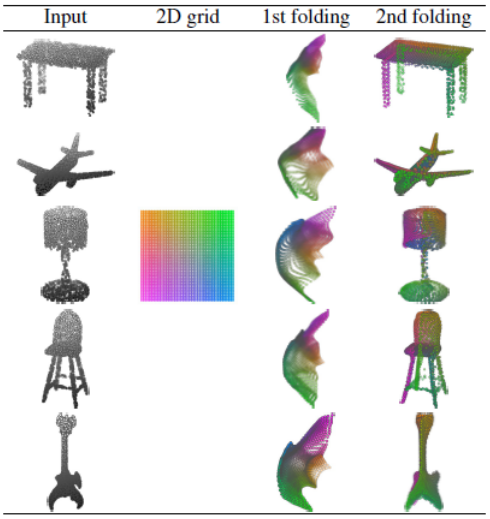


Fig. 10. Illustration of two-step-pfolding decoding

points of size $m \times 2$ to generate a matrix of $m \times 514$. This matrix is the input to the 3 layer perceptron and generates the output of the first folding operation ($m \times 3$). Then again, it is concatenated with replicated codewords and becomes the input to the second folding operation which results in reconstructed PC. Two folding operation can be perceived as a 2D to 3D transformation/mapping that produces elaborating surfaces. The first one folds the 2D grid to 3D space and the second one folds inside the 3D space. The no. of output points m is not necessarily same as n. The distortion error between the input and the reconstructed PC is calculated using chamfer

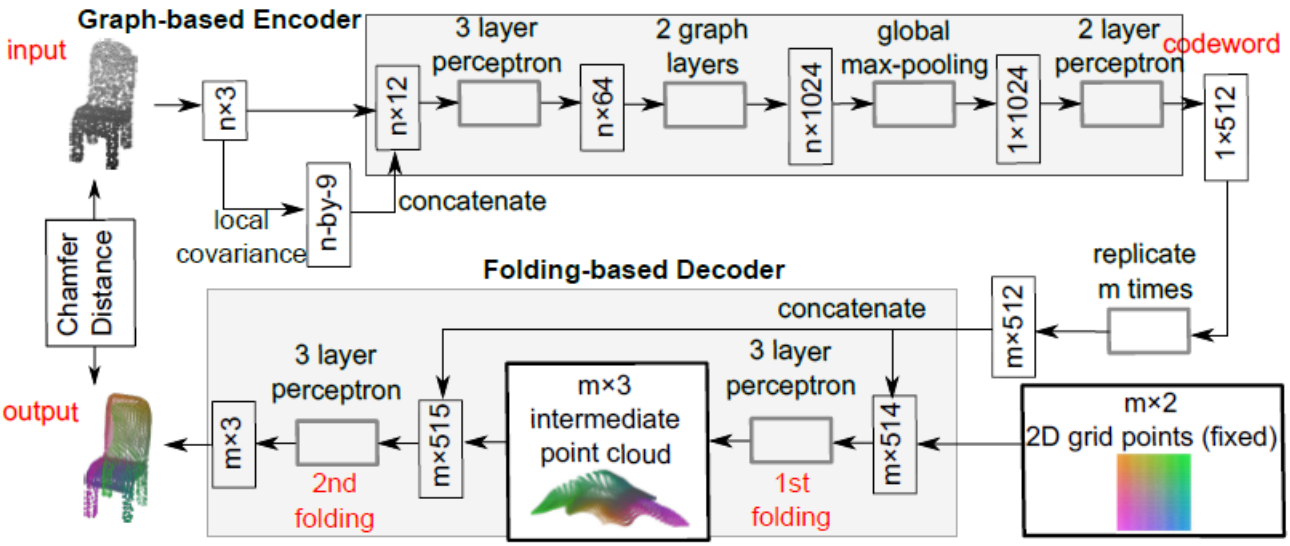


Fig. 11. FoldingNet architecture

distance defined in previous section.

During training, the model starts with a random folding shown in Fig. 10 but eventually turns into a meaningful PC. The model was trained using 16 categories of ShapeNet dataset. One major advantage of Folding based decoder shown in Fig. 11 is that it only uses 7% parameters of a decoder with FC NN and it is more focused towards solving the classification problem rather than compressing the PC. But the idea could be adapted for PCC to validate the performance in comparison to international standardization. As far as classification problem is considered, the codewords generated at the end of the encoder are used to classify using SVM.

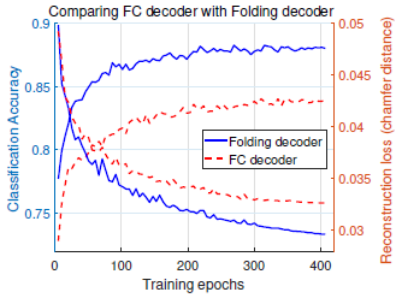


Fig. 12. Comparison between fully connected (FC) and folding decoder

The folding based decoder was compared against the FC decoder and shown in Fig. 12 . During the training process, the reconstruction loss (Chamfer distance) kept decreasing which implies the reconstructed PC is becoming more and more similar to the original PC. And the classification accuracy of the linear SVM which was trained on the codewords improved implying that the codeword becomes more linearly separable while training.

IV. RNN

There are few DNN approaches that uses RNN for point cloud compression. One of them is discussed in this section that focuses on data from a 3D LiDAR sensors. The conventional methods that are part of the standardization are not very successful in reducing the point cloud to a very low volume, especially 3D LiDAR sequences. Therefore, some approaches operate on the raw packet data from a 3D LiDAR which is shown to be highly cost effective approach for compression. Although it is possible to transform raw packet data into 2D matrix losslessly, but they are very irregular in nature and does not share any spatial correlation ,if taken without calibrations. But it reduces the sparsity of the PC which is helpful for the further processing [7].

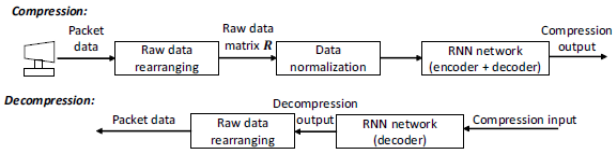


Fig. 13. RNN based PC compression scheme

The proposed scheme targets static PC and introduces a RNN based compression method with residual blocks. The overview of the scheme is shown in Fig. 13. First, 2D matrices are constructed by transforming the frames of raw packet data. Normalization is performed as a pre-processing step before feeding it to the network which comprises of an encoder, a binarizer and decoder. As 2D format can have a much larger range of possible values due to the accuracy and long range detection of LiDAR scanners, the scheme uses a few steps to normalize: The packet data are randomly chosen feom 100 frames to calculate the mean μ and the histogram of distance values are created and the data matrix R is normalized as

follows: $(R-\mu)/\theta$ where threshold θ is set such that 95% of the distances fall under this value.

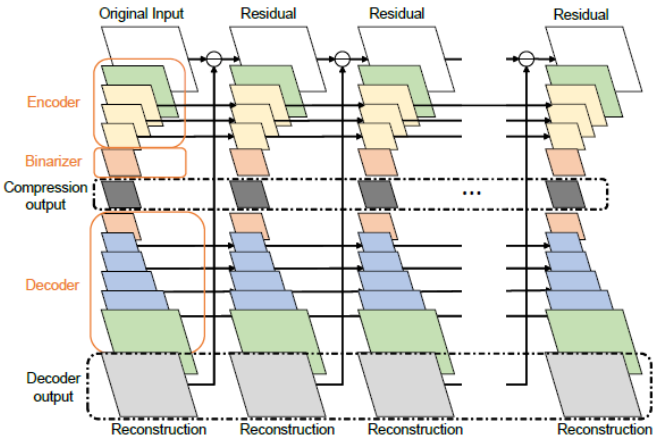


Fig. 14. Overall architecture of RNN based technique

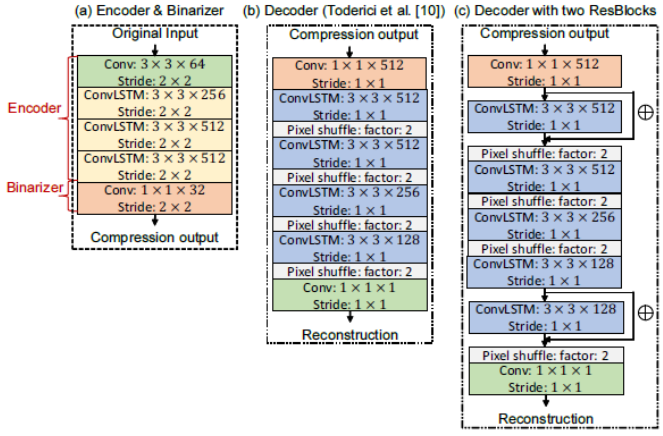


Fig. 15. Network detail

The structure of the network is shown in Fig. 14. The scheme proposes to use a decoder with residual blocks since getting an accurate decompression can be challenging while using such a network. The 2D matrix is encoded first and transforms it into a binary file. The encoder, decoder and decoder with residual blocks comprises convolutional layers and convolutional LSTM layers. the details are given if Fig. 15. The decoder employs the binary file to reconstruct the original input and counted as one iteration. The residual is computed by comparing the reconstructed data and becomes the input for next iteration. this process is repeated until the decompressed data is reasonably accurate. The scheme uses driving data from 11 areas of japan and compared against octree compression and JPEG compression.

Machine learning for solving Point cloud related problems: There are lot of recently proposed Machine learning based methods used to solve point cloud problems as listed [8], [9], [10], [11], [12], [13]

V. Conclusion

The paper focus on emerging research area of PCC using deep learning techniques. It briefly describes some of the most recent developments made and compared against each other to analyze their performances. There are more techniques proposed related to geometry compression and very few focusing attribute compression of point clouds [14] and video compression [15]. Most of the approaches use convolutional based autoencoder for compression, multi-layer perceptron and fully connected deocoders to achieve Point cloud compression. But there are some RNN based techniques introduced in the recent past.

References

[1] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Point cloud coding: Adopting a deep learning-based approach," in *2019 Picture Coding Symposium (PCS)*, 2019, pp. 1–5.

[2] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Deep learning-based point cloud geometry coding: Rd control through implicit and explicit quantization," in *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, 2020, pp. 1–6.

[3] M. Quach, G. Valenzise, and F. Dufaux, "Learning convolutional transforms for lossy point cloud geometry compression," *CoRR*, vol. abs/1903.08548, 2019. [Online]. Available: http://arxiv.org/abs/1903.08548

[4] J. Wang, H. Zhu, Z. Ma, T. Chen, H. Liu, and Q. Shen, "Learned point cloud geometry compression," 2019. [Online]. Available: https://arxiv.org/abs/1909.12037

[5] W. Yan, Y. Shao, S. Liu, T. H. Li, Z. Li, and G. Li, "Deep autoencoder-based lossy geometry compression for point clouds," *CoRR*, vol. abs/1905.03691, 2019. [Online]. Available: http://arxiv.org/abs/1905.03691

[6] Y. Yang, C. Feng, Y. Shen, and D. Tian, "Foldingnet: Interpretable unsupervised learning on 3d point clouds," *CoRR*, vol. abs/1712.07262, 2017. [Online]. Available: http://arxiv.org/abs/1712.07262

[7] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3274–3280.

[8] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *CoRR*, vol. abs/1608.04236, 2016. [Online]. Available: http://arxiv.org/abs/1608.04236

[9] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas, "Representation learning and adversarial generation of 3d point clouds," *CoRR*, vol. abs/1707.02392, 2017. [Online]. Available: http://arxiv.org/abs/1707.02392

[10] A. Dai, C. R. Qi, and M. Nießner, "Shape completion using 3d-encoder-predictor cnns and shape synthesis," *CoRR*, vol. abs/1612.00101, 2016. [Online]. Available: http://arxiv.org/abs/1612.00101

[11] 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*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 40–49. [Online]. Available: http://proceedings.mlr.press/v80/achlioptas18a.html

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

[13] R. Klokov and V. Lempitsky, "Escape from cells: Deep kd-networks for the recognition of 3d point cloud models," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[14] P. A. Chou, M. Koroteev, and M. Krivokuća, "A volumetric approach to point cloud compression—part i: Attribute compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 2203–2216, 2020.

[15] T. Chen, H. Liu, Q. Shen, T. Yue, X. Cao, and Z. Ma, "Deepcoder: A deep neural network based video compression," in *2017 IEEE Visual Communications and Image Processing (VCIP)*, 2017, pp. 1–4.