DEEP LEARNING-BASED POINT CLOUD CODING: A BEHAVIOR AND PERFORMANCE STUDY

André F. R. Guarda[1], *Nuno M. M. Rodrigues*[2], *Fernando Pereira*[1]

[1]Instituto Superior Técnico, Instituto de Telecomunicações, Lisbon, Portugal
[2]ESTG, Instituto Politécnico de Leiria, Instituto de Telecomunicações, Leiria, Portugal

ABSTRACT

Point clouds are an emerging 3D visual representation model for immersive and interactive multimedia applications, in particular for virtual and augmented reality. The huge amount of data associated to point clouds critically asks for efficient point cloud coding technology. While there are already some point cloud coding paradigms in the literature, notably octree, patch and graph-based for geometry data, very recently deep learning emerged in this research domain, offering very promising performances for image coding. While deep learning-based methods often provide interesting results, the understanding of this type of coding solutions is essential to improve their design in order to be used effectively. In this context, this paper presents a study and analysis on the behavior and performance of a deep learning-based point cloud coding solution based on an autoencoder network using only convolutional layers. Beside a promising RD performance, other findings should allow making solid steps in understanding this emerging coding paradigm.

Index Terms— Point cloud coding, deep learning, convolutional neural network

1. INTRODUCTION

Point cloud (PC) coding is an increasingly relevant research domain, with standards currently under development by MPEG [1] and JPEG [2]. Due to the rise in interest and availability of virtual and augmented reality multimedia applications, e.g. for entertainment, geographical information and communication purposes, point clouds (PCs) are becoming a fundamental 3D visual representation format for many future multimedia applications [3]. PCs offer a versatile representation format, providing easy access to multiple viewpoints and interaction with objects, which enables immersive and interactive user experiences. Their main advantage is that, unlike meshes, PCs do not have connectivity constraints, making them lighter and better suited for real-time applications [4]. A PC consists in an unordered set of points in the 3D space, which are represented by their 3D coordinates, x, y, z, i.e. the geometry, and optionally their associated RGB colors or other attributes. However, this paper focuses only on static geometry coding. Due to the nature of PCs, they can contain millions of points, and thus a huge amount of data, making efficient PC coding a must. There are several PC coding paradigms available in the literature, notably octree [5], graph [6] and patch-based [7] coding solutions. In this paper, a new PC coding paradigm is explored, notably based on deep learning.

In recent years, deep learning has become very popular for image and video processing [8]. More recently, deep learning has started to make its way into the coding arena, in particular for image coding [9], [10]. While still in its infancy, deep learning-based methods have already achieved remarkable results in image coding [10], managing to compete with state-of-the-art HEVC [11] Intra coding [12]. Thus, progressing to PC coding is a natural step for deep learning coding solutions.

In this context, the purpose of this paper is to study the behavior and performance of a deep learning-based point cloud geometry coding solution. The adopted solution uses a convolutional autoencoder (AE) to transform the PC data into a latent representation with lower dimensionality, thus easier to code, while also allowing the inverse transform to faithfully reconstruct the PC data. The AE behavior will be studied in detail, with multiple experiments performed to evaluate the impact of some key parameters in the final rate-distortion (RD) performance. It is also vital to analyze the latent representations, to understand how the AE captures and represents the input data features. This information is useful so that more sophisticated deep learning-based PC coding techniques can be further designed, taking advantage of this knowledge and findings to improve the overall performance.

The paper is organized as follows: Section 2 briefly reviews the background work on PC coding. Section 3 describes the adopted deep learning-based PC geometry coding solution. Section 4 presents the experimental results, with a detailed analysis and discussion of the codec behavior and performance. Finally, Section 5 concludes the paper.

2. BACKGROUND WORK

For a better understanding of the PC coding field, a short review of the state-of-the-art in static PC geometry coding is presented in this section. Most PC geometry coding methods in the literature can be divided in two main paradigms: octree-based [5], [13] and patch-based [7], [14] methods. In the former, an octree is built to represent the PC, by recursively

978-1-7281-4496-2/19/$31.00 ©2019 IEEE

dividing the PC into octants, until reaching a chosen depth [5]. This is the working principle of the Point Cloud Library (PCL) [15] codec, which became the reference solution in PC geometry coding, due to its conceptual simplicity and the publicly available software. As for patch-based methods, they usually partition the PC into smaller clusters, project each cluster into a 2D map, obtaining depth and occupation maps, which may be then encoded with available image or video coding standards [7], [14], such as JPEG [16] and HEVC [11]. More recently, a graph-based PC geometry coding solution has been presented in [6]. While graph-based methods have been used before for attribute coding, this was the first time this approach was applied to geometry coding. The proposed coding solution still uses an octree to code a coarser version of the PC, and then the graph-based method is used to code additional detail.

Currently, MPEG is in the process of developing two PC coding standards. The Geometry-based Point Cloud Compression (G-PCC) standard [1], focused on static PCs, is an octree-based coding method which encodes an octree until a given depth, and then adds detail to each voxel by estimating the surface passing through it. As for the Video-based Point Cloud Compression (V-PCC) standard [17], it targets dynamic PCs, and uses a patch-based approach where the PC geometry, as well as attributes, are projected into 2D maps which are coded with a video codec, e.g. HEVC [11].

3. DEEP LEARNING-BASED POINT CLOUD GEOMETRY CODING SOLUTION

In this section, the adopted deep learning-based PC coding solution is described. This solution uses a neural network, namely an autoencoder (AE), to transform every PC coding unit into a set of coefficients, here called *latents*, which are quantized and entropy coded for storage or transmission. Unlike the traditional, fixed transforms (e.g. DCT, DST), this deep learning-based solution is able to learn a transform adapted after training with enough representative data [10].

Fig. 1 presents the overall architecture of the adopted PC geometry coding solution using an autoencoder design, from here on referred to as Point Cloud Geometry-AE (PCG-AE). Just as for image coding, the full PC is initially divided into coding units, smaller than the full PC, designated as *3D blocks*. Each 3D block consists on binary voxels, where a '1' signals a filled point at the corresponding coordinate, and a '0' represents an empty space. The 3D blocks are transformed with the AE encoder, resulting into a set of latents, or coefficients, which may then be quantized and entropy coded.

The AE used in the PCG-AE coding solution is presented in Fig. 2. This type of AE is generally characterized by three components: an encoder, its complementary decoder, and a bottleneck associated to the latents exchanged between them [18]. The AE encoder learns a function to transform the input data into a set of coefficients, the so-called *latent representation*. The AE decoder, often symmetric of the AE encoder, is tasked with reconstructing an approximate version

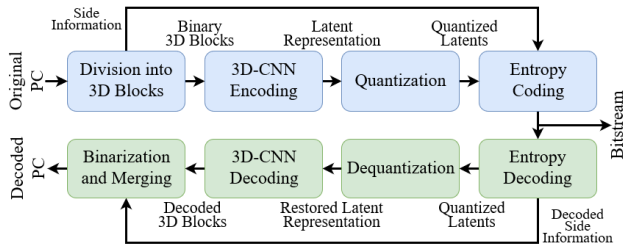


Fig. 1. Overall architecture of PCG-AE coding solution.

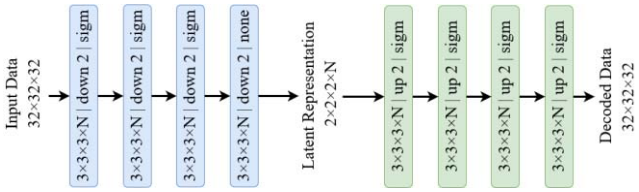


Fig. 2. 3D-CNN AE architecture. All layers are convolutional with the following hyper-parameters: filter support width × height × depth × number of filters (N) | sampling stride | activation function.

of the input coding unit, using the latent representation and the learned inverse transform. The latent representation is constrained to have a lower dimensionality than the input data, forcing the AE to learn the most relevant features so that the input PC can be reconstructed as faithfully as possible.

The adopted AE consists of 3D convolutional layers only, which are the direct extension of their 2D counterpart used for 2D images [19], in order to code 3D blocks. Each convolutional layer applies a set of N filters to extract N feature maps, using a filter support of 3×3×3, with a stride of 2. This means that the encoder feature maps are successively down-sampled in half, at each layer, so that the latent representation has a much lower dimensionality, thus increasing the compression factor. At the decoder side, the feature maps are up-sampled so that the decoded data has the same resolution as the input data. After each layer in this 3D-CNN, a sigmoid activation function is used, with the exception of the last encoder layer. This sigmoid function provides a decoded value between zero and one for each decoded voxel, that can be easily used by the loss function.

As for all deep learning methods, the AE needs to be trained beforehand to determine the best neural network parameters, thus defining the most appropriate transforms for this type of data. During training, the loss function measures the 3D block reconstruction error, in this case using the so-called *weighted binary cross-entropy* (wBCE) [20]. The wBCE measures the average error in terms of logarithmic loss between each original voxel value and the corresponding AE decoder's predicted values. Since the number of 0s in the input 3D block usually largely outnumbers the number of 1s, a weight w is used to boost the error impact of false negatives, so that its importance is not diluted.

In order to increase the codec compression ratio, the floating-point latent values are quantized into integers using a uniform quantization function, using the same quantization step for all latents. The quantization process is regulated by the number of bits used to represent each latent value, *nb*. The

quantized latent values are finally entropy coded into a single file (bitstream) using 7z [21]; this file also includes the side information conveying the positioning on the 3D blocks in the full PC, which is required to appropriately reconstruct it.

The PCG-AE decoder mirrors the PCG-AE encoder operations by performing entropy decoding, dequantizing the latent values, and AE decoding the 3D blocks from the latent values. The AE decoded floating point values are classified as 0 or 1 by using a binarization threshold, *BTh.* Finally, all binarized 3D blocks are merged to reconstruct the full PC.

4. PERFORMANCE ANALYSIS AND DISCUSSION

In this section, the adopted PCG-AE codec is studied by analyzing its behavior and performance through a series of experiments where some key parameters vary within a meaningful range. The objective is to gain a deeper insight on this new coding paradigm, so that strategies to improve the codec's performance may be designed in the future. This type of exercise is also fundamental to avoid the impression that deep-learning based solutions are essentially a 'magic black box'. Before presenting the results, the conditions, setup and methodology for the experiments are described.

4.1. Experimental Conditions and Setup

PC Data: MPEG dataset [22], as used for the MPEG PCC standardization process. PCs were split into training and testing sets, with four PCs selected for testing (*Statue Klimt*, *Queen*, *House without Roof* and *Longdress*, see Fig. 3), and the remaining PCs used to train the AE models. All PCs were divided into 3D blocks of size 32×32×32 voxels.

Training data: Blocks with a point count lower than 500 points were discarded, since they contain very little relevant geometry information to be learned; naturally, this did not occur for testing, where all 3D blocks were coded. Out of the remaining blocks, 6000 were chosen randomly for training. When appropriate, PCs with lower point density were down-sampled to a precision of 9 or 10 bits to reduce sparsity.

Training process: Three 3D-CNN AE models were trained using different number of filters *N*, notably 32, 64 and 96 filters. Models were trained with stochastic gradient descent using the Adam algorithm [23] with a learning rate of 10^{-4}, and minibatches of 8 3D blocks at a time. The adopted weight of the wBCE loss function was 30 as it was found to offer a good balance between false negatives and false positives.

Quantization: To obtain different coding bitrates, *nb* was varied between 2 and 16 bits per latent value.

Performance Metrics: Bitrate is measured as bits-per-point (bpp) of the original PC; geometry quality is assessed with the PSNR associated to the point-to-point distance metric, referred to as D1 in the MPEG PCC test conditions [22].

Benchmarking: As for MPEG PCC, the natural benchmark for the PCG-AE codec is the popular PCL codec [15] since the MPEG G-PCC [1] reference software is still not publicly available to code the test PCs adopted for this study.

(a) (b) (c) (d)

Fig. 3. Test PCs: (a) *Statue Klimt*; (b) *Queen*; (c) *Longdress*; (d) *House without Roof.*

4.2. Number of Reconstructed Points

The number of reconstructed points is a critical metric in terms of subjective quality. Therefore, a study has been performed on the overall number of reconstructed points, and its dependence on the number of 3D-CNN filters, *N*. Fig. 4 compares the number of original and reconstructed points per 3D block for varying *N*. Blocks (in *x* axis) are presented in ascending order of number of points (in *y* axis). It can be observed that the number of reconstructed points tends to be higher than the number of original points. In addition, a lower *N* results in an excessive number of reconstructed points, up to twice the original number in some cases, with possible effects (not necessarily negative) on the quality. This behavior indicates that a 3D-CNN with less filters has more difficulty to accurately represent sparser 3D blocks. For *Queen* and *Longdress*, the number of reconstructed points follows the input number more closely than the other two PCs, which can be explained by their significantly higher point density, thus fewer sparse 3D blocks to be coded.

The number of reconstructed PC points is directly affected by two parameters: i) the wBCE weight, *w,* used during training (30 as default); and ii) the binarization threshold, *BTh,* used to round the AE decoded values to 0 or 1 (0.5 as default). In theory, increasing *BTh* should reduce the number of reconstructed points, especially for lower *N*. With this in mind, the histogram of the AE decoded voxel values, before binarization, is shown in Fig. 5. The results show that most AE decoded voxel values are very close to 0 or 1, whereas the histogram is nearly flat between 0.1 and 0.9. Contrary to the expectations, the conclusion is that it is not critical where the binarization threshold, *BTh,* is precisely placed for the final reconstructed PCs, as changing around 0.5 it would yield almost no change in the number of points, unless going to very extreme values.

4.3. Global-level Latents Distribution

The latents produced by the AE learned transform are the basis for reconstructing a faithful replica of the original PC. Unlike traditional transforms, e.g. the DCT, it is not possible to easily analyze the AE adaptive transform to determine its behavior or its mathematical formulation. However, the latent representation may be numerically studied, and this is the main target of this section. The PC latent representation consists in *N* 2×2×2 blocks of feature maps; for *N=32* and *N=64*, 256 and 512 latents are produced, respectively.

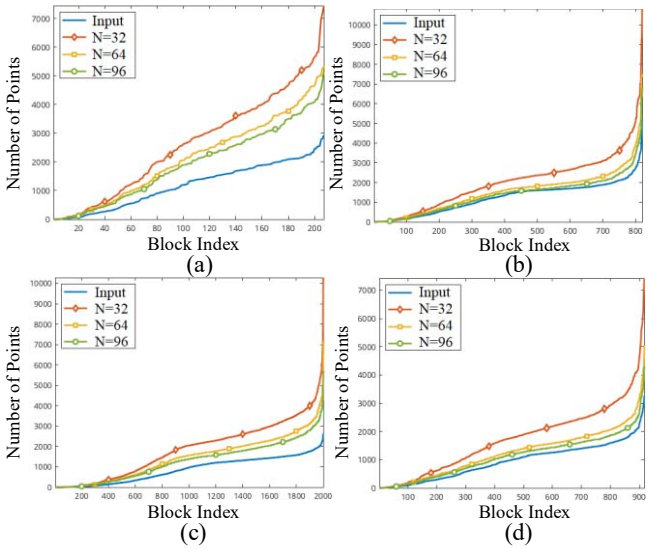


Fig. 4. Number of original and reconstructed points with PCG-AE for different number of filters *N*. (a) *Statue Klimt*; (b) *Queen*; (c) *House without Roof*; (d) *Longdress*.

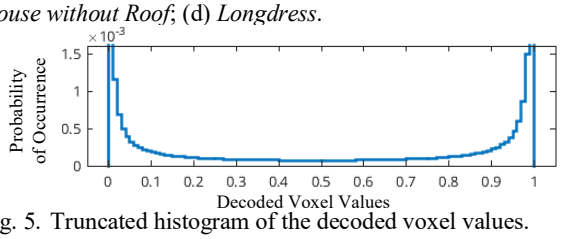


Fig. 5. Truncated histogram of the decoded voxel values.

Fig. 6 shows some statistics for the latent values of all test PCs, and their variation across the *N*=32 or *N*=64 feature maps, namely the maximum, minimum and mean values, as well as their variance and dynamic range (maximum minus minimum). The histogram of all latent values, for all PCs, is presented in Fig. 7, for *N*=32 and *N*=64. The first observation is that most latent values are concentrated around zero, with a global dynamic range from -7 to 8, approximately. However, the dynamic range varies between feature maps, hinting that quantization may be more efficient if different steps are used for each feature map. The latents statistics also show that the mean is around 0, which suggests that the negative and positive values are well balanced; moreover, the latents variance is low and close to 0, thus indicating that the latent values do not deviate much from their mean.

4.4. Block-level Latents Distribution

The statistics in Fig. 6 and Fig. 7 refer to all 3D blocks, which means they offer global indicators but do not allow to extract conclusions at the 3D block level, the level where coding is performed. As such, Fig. 8 presents three example 3D blocks with distinct characteristics for a more detailed study: (a) simple shape with low number of points (23, 0.07 % of the 3D block), only occupying a small region of the full 3D block, i.e. two octants; (b) complex shape with high number of points (3722, 11 %), generally occupying the whole block; and (c) simple shape (planar) with average number of points (1559, 4.8%), occupying only the top half of the block, i.e. four octants. Fig. 9 shows the latent values obtained for these

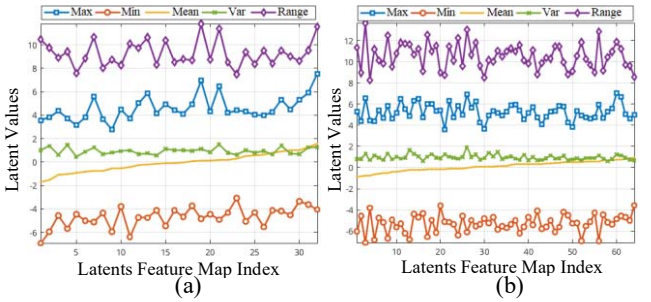


Fig. 6. Statistics of the latent values, for each feature map, for all 3D blocks in all PCs, namely maximum, minimum, mean, variance and dynamic range: (a) *N*=32 and (b) *N*=64.

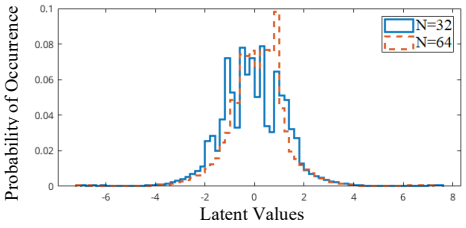


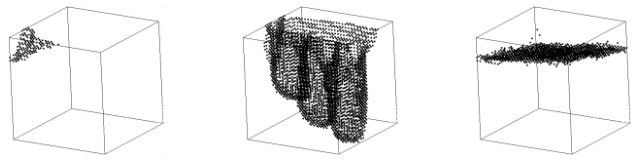
Fig. 7. Histogram of latent values for all PCs, for *N*=32 and *N*=64.

3D blocks, using *N*=32 and *N*=64. The latent values are presented as a 2D gray image of 8×*N*, where the 2×2×2 feature maps are displayed linearly following a raster scan order, with the minimum (negative) value as black, the maximum as white, and zero is gray in the middle of the scale.

In Fig. 9, it is possible to visualize the effects of each 3D block's characteristics on the latent representation. For instance, Block 1 takes, for any given feature map, the same latent values for all positions except two (position 5 and 7), which correspond to the two occupied octants in the original block. This is an effect of the feature maps down-sampling performed at each convolutional layer. However, when considering different feature maps, the corresponding latent values are different, depending on the learned bias parameter for each filter in the convolutional layer [19]. Block 2 presents the largest variations inside each feature map, as its points are spread in the full 3D block. Finally, Block 3 also shows four constant values in each feature map, corresponding to the empty octants. Overall, each filter seems to have learned to obtain good different features, since the feature maps are mostly different for each 3D block.

Fig. 10 shows the energy of the latent values for each feature map, for the three example blocks, computed as the sum of the squared latent values within each feature map. The obtained energies are in accordance to the previously observed results, where Block 1 has a low number of points, thus a much lower energy than the others, whereas the more complex Block 2 shows the highest energy. Fig. 10 also demonstrates that the feature maps have rather different energy, indicating that some filters detect more important features than others. The feature maps with lower energy should correspond to features that are not present in the current 3D block, thus having a smaller impact for decoding.

4.5. RD Performance with Uniform Quantization



(a) (b) (c)

Fig. 8. 3D blocks with different characteristics: (a) Block 1 – simple, low point-count shape; (b) Block 2 – complex, high point-count shape; (c) Block 3 – planar, average point-count shape.

Naturally, the RD performance is the most critical assessment criterion for a codec which targets compression efficiency. In this context, Fig. 11 shows the PCG-AE RD performance using a uniform quantization for the latent values, for a varying number of filters, *N*. The results are compared with the PCL benchmark. For each *N*, the value of *nb* was varied in order to obtain multiple RD points. From Fig. 11, it may be concluded that for the medium and high bitrates, PCG-AE consistently outperforms PCL; however, the opposite happens for the lower rates where PCG-AE cannot reach bitrates as low as PCL. As the number of filters decreases, the quality and bitrate also decrease, as expected, since *N* is directly related to the number of latent values. For lower bitrates, it is preferable to reduce the number of filters than simply applying stronger quantization. The best performance is achieved with *N*=32 for medium bitrates and *N*=64 for higher bitrates. Raising *N* above 64 severely increases the bitrate with minimal quality gain.

4.6. RD Performance with Energy Filtering

As shown in Fig. 7, many latent values are close to zero, raising the question of how important are the smaller latents for the final RD performance. To understand the impact of eliminating the smaller latents, an energy filtering method is applied: after the quantization, a given percentage of the energy is removed, starting from the latent values with lowest energy. To this end, for each 3D block the total energy of its latent representation (including all *N* feature maps) is determined, and the set of smallest latent values is selected, corresponding to a target percentage of the total block energy. Then, the selected latent values are set to zero. While this process certainly reduces the rate, its effects on the reconstructed PC quality must be studied. This energy filtering is performed after quantization to be sure that an effective energy reduction is applied beyond the quantization; the alternative solution (energy filtering before quantization) could simply lead to removing the same latents as the uniform quantizer dead zone, thus making this study ineffective.

In this experiment, latents were removed up to different accumulated energy percentages, namely 10, 20 and 30 percent, for the AE trained models with *N*=32 and *N*=64 filters. The corresponding RD performance is shown in Fig. 12, where the RD curves are labelled as *AE-N-EF* where *N* refers to the number of filters and *EF* to the percentage of filtered energy. As expected, there is a bitrate reduction with this energy filtering, which grows as more latents are set to zero. For *N*=64, in spite of a small quality reduction for the

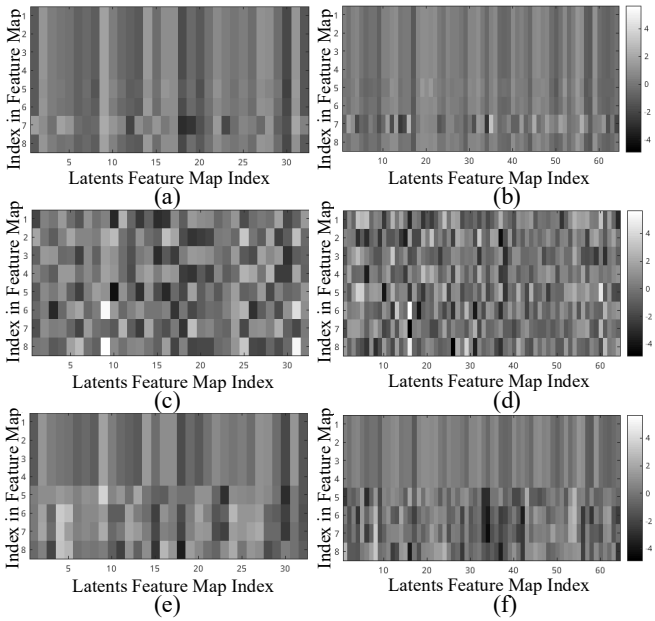


Fig. 9. Latents values for (top to bottom) block 1, block 2, block 3; left obtained with *N*=32, right obtained with *N*=64.

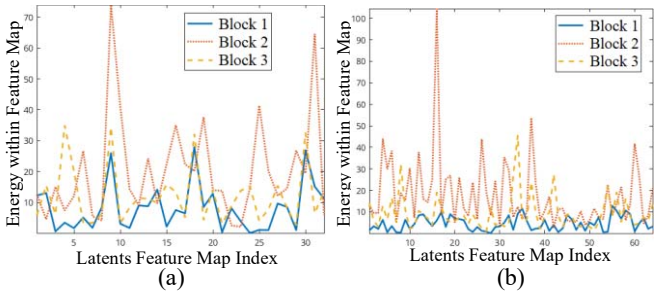


Fig. 10. Energy of latent values within each feature map for the example 3D blocks in Fig. 8: (a) with *N*=32 and (b) with *N*=64.

higher rates, the overall RD performance improves for higher *EF* percentages, especially at lower bitrates. For *N*=32, the loss of quality for the higher rates seems to be more significant, which in turn causes the RD performance gains to be smaller and the best RD performance is obtained for EF equal to 20%. Overall, this behavior implies that filtering energy when less latents are used, i.e. lower *N*, is more critical than filtering the same energy percentage for a larger number of latents. This seems to indicate that, for a smaller number of latents, there is less redundancy, and thus they are individually more important, even the lower energy latents.

5. CONCLUSIONS

This paper studies the behavior and performance of a deep learning-based approach to (lossy) static PC geometry coding solution designated as PCG-AE. The PCG-AE RD performance is encouraging since it is able to overcome the PCL RD performance for a good range of bitrates, although this deep learning-based solution is far from optimal. The analysis of the latents behavior should provide useful hints to optimize the AE architectural parameters as well as the quantization and entropy coding processes. This type of

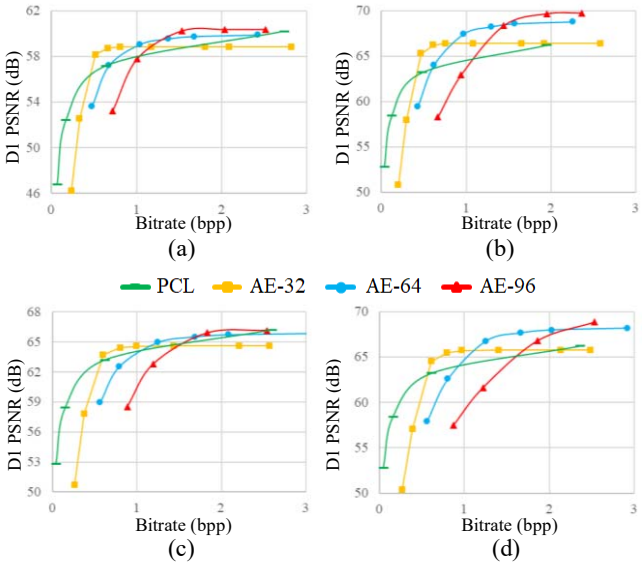


Fig. 11. RD performance for PCL and PCG-AE for different *N*. (a) *Statue Klimt*; (b) *Queen*; (c) *House without Roof*; (d) *Longdress*.

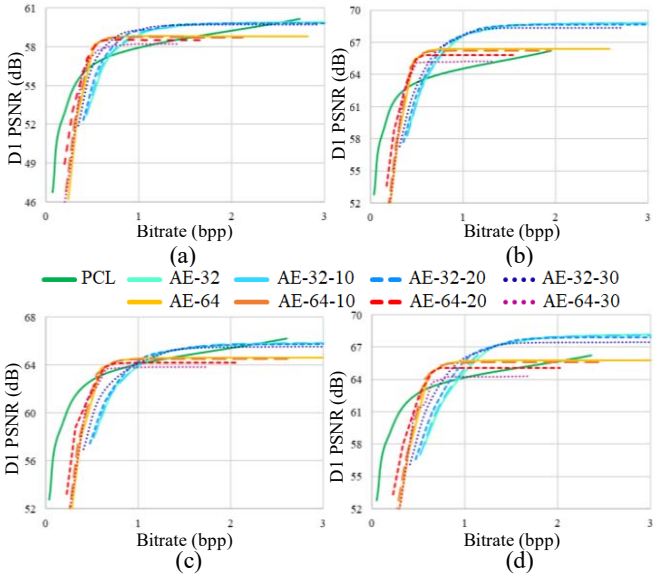


Fig. 12. RD performance for PCL and PCG-AE with quantization followed by energy filtering *EF*, for *N*=32 and 64 (*AE-N-EF*). (a) *Statue Klimt*; (b) *Queen*; (c) *House without Roof*; (d) *Longdress*.

analysis is crucial for the design of more sophisticated and better performing deep learning-based PC coding solutions.

6. ACKNOWLEDGMENTS

This work was funded by Fundação para a Ciência e Tecnologia (FCT), Portugal, Ph.D. Grant SFRH/BD/118218/2016, by FCT/MEC through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2019.

7. REFERENCES

[1] S. Schwarz *et al*., "Emerging MPEG standards for point cloud compression," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 1, pp. 133–148, Mar. 2019.

[2] T. Ebrahimi *et al.,* "JPEG Pleno: Toward an efficient representation of visual reality," IEEE MultiMedia, vol. 23, no. 4, pp. 14-20, Oct.-Dec. 2016.

[3] K. Sugimoto *et al.,* "Trends in efficient representation of 3D point clouds," APSIPA Annual Summit Conf., Kuala Lumpur, Malaysia, Dec. 2017.

[4] S. Orts-Escolano *et al*., "Holoportation: virtual 3D teleportation in real-time," Symp. User Interface Softw. Technol., Tokyo, Japan, Oct. 2016.

[5] R. Schnabel *et al.*, "Octree-based point-cloud compression," IEEE VGTC Conf. Point-Based Graph., Boston, MA, USA, July 2006.

[6] P.Rente *et al.,* "Graph-based static 3D point clouds geometry coding," IEEE T-MM, vol. 21, no. 2, pp. 284-299, Feb. 2019.

[7] T. Ochotta *et al*., "Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields", EG Symp. Point-Based Graph., Zurich, Switzerland, June 2004.

[8] M. Z. Alom *et al*., "The history began from AlexNet: a comprehensive survey on deep learning approaches," arXiv:1803.01164, Sept. 2018.

[9] G. Toderici *et al*., "Variable rate image compression with recurrent neural networks," Int. Conf. on Learning Representations, San Juan, Puerto Rico, May 2016.

[10] J. Ballé *et al*., "Variational image compression with a scale hyperprior," Int. Conf. on Learning Representations, Vancouver, Canada, Apr.2018.

[11] G. J. Sullivan *et al.,* "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE T-CSVT, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[12] F. Bellard, "BPG Image format", 2014. [Online]. Available: https://bellard.org/bpg/. [Accessed: 06- Jul- 2019].

[13] R. Mekuria *et al*., "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," IEEE T-CSVT, vol. 27, no. 4, pp. 828-842, Apr. 2017.

[14] H. Houshiar *et al*., "3D point cloud compression using conventional image compression for efficient data transmission," Int. Conf. Inf. Commun. Automat. Technol., Sarajevo, B&H, Oct. 2015.

[15] R. B. Rusu *et al.*, "3D is here: Point Cloud Library (PCL)," IEEE Int. Conf. Robot. Automat., Shanghai, China, 2011.

[16] G. K. Wallace, "The JPEG still picture compression standard," Commun. ACM, vol. 34, pp. 30-44, Apr. 1991.

[17] E. S. Jang *et al*., "Video-based point-cloud-compression standard in MPEG: from evidence collection to committee draft [Standards in a nutshell]," IEEE Signal Process. Mag., vol. 36, no. 3, pp. 118-123, May 2019.

[18] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 85-117, Jan. 2015.

[19] I. Goodfellow *et al*., "Deep learning," MIT Press, 2016.

[20] T. Lin *et al*., "Focal loss for dense object detection," IEEE Int. Conf. Computer Vision (ICCV), Venice, Italy, Oct. 2017.

[21] I. Pavlov, "7-Zip", 7-zip.org. [Online]. Available: https://www.7-zip.org/. [Accessed: Jul. 2019].

[22] S. Schwarz *et al.,* "Common test conditions for point cloud compression," ISO/IEC JTC1/SC29/WG1 MPEG output document N17345, Gwangju Meeting, Jan. 2018.

[23] D. P. Kingma *et al.*, "Adam: a method for stochastic optimization," Int. Conf. on Learning Representations, San Diego, CA, USA, May 2015.