Explanation, Critique, Implementation and Improvement of "Cloud Transformers: A Universal Approach To Point Cloud Processing Tasks"

Josh Borrett
MASH
joshua.borrett@dauphine.eu

Killian Coustoulin
MASH
killian.coustoulin@dauphine.eu

Abstract

The aim of our project is to demonstrate a thorough understanding of the article and related material, to implement the relevant code and finally to propose improvements or extensions to the method, thus fulfilling the entirety of the marking criteria set out. This work has been carried out equally between the authors

1 Comprehension

1.1 Explanation

In this first section we discuss and demonstrate an exhaustive understanding of the proposed method.

1.1.1 Introduction and Summary

The overall main aim of the paper "Cloud Transformers: A Universal Approach To Point Cloud Processing Tasks" is to propose a novel architecture to apply to point cloud recognition and generation tasks. An example of a generation task might be generating a 3D cloud from a 2D image, whereas recognition tasks are often semantic segmentation which is the task of associating to each point in the cloud the object to which that point belongs (similar to TP5), or correctly labeling objects in that cloud. The architecture proposed is based upon transformers and their multi headed attention variants as described in Vaswani et al. [2017] Cordonnier et al. [2021] The authors develop "Cloud Transform Blocks" which they show can be added sequentially, in a similar fashion to Recurent NN's or other deep learning frameworks, and are highly adaptable to both generation and classification. They test this method against the current state of the art (at the time) and show an improvement in both performance, beating the state of the art over the benchmark and in adaptability, being applicable to both generation and classification tasks which was not previously the case.

1.2 Exploration of Related Work

In order to be able to understand fully the advancements made by this paper it is important to be aware and understand the related literature so as to be able to make a fair comparative evaluation of the proposal. Taking inspiration from its effectiveness on 2D Image Analysis tasks, derivatives of Convolutional NN's, O'Shea and Nash [2015] make up the majority of the state of the art for point cloud processing tasks. Here we briefly explain the methodology and results of some of these methods. One of the challenges CNN based methods meet in point cloud problems is sparsity because compared to images, even dense point clouds are sparse and irregular. One method to attempt to fix this is to transform this sparse point cloud into a voxel grid as in Maturana and Scherer [2015] and then we can apply 3D convolutions to learn features to help in classification tasks. One issue encountered in this method is that computing convolutions over dense voxel grids can be computationally intensive and difficult to scale. The other avenue to counter the issues faced by CNN's to point clouds is to build local graphs in a neighborhood of each point and then apply a transformation function to learn features of the point cloud. This has the advantage of being able to take the point cloud directly as an input. However the issues are firstly that the learning method often requires the use of Multi-Layer Perceptrons which

Foundation Models for Science Workshop,38th Conference on Neural Information Processing Systems (NeurIPS 2024).

themselves can be difficult to tune, and secondly, due to the construction of the local graphs these methods are very sensitive to the point cloud density. The 2 major distinctions between this article and previous research is the ability to perform both generative and classification tasks, as well as the integration of Transformers to these tasks. Indeed, this is not the only paper to have made contributions to the integration of Transformers to point cloud problems as Zhao et al. [2021] also did the same, however Point Transformer outperforms this article on semantic segmentation benchmarks, there has been little in way to adapting it to generation tasks, highlighting the importance of this article.

1.3 Cloud Transformer Architecture

In the following subsections we will explain the building block nature of the Cloud Transformer, using a lower level language compared to the original paper as well as drawing parallels and comparisons to demonstrate understanding.

The Cloud Transformer architecture, while relatively complex and drawing elements from several existing architectures, is quite intuitive. We can think about this architecture as being made up of several building blocks where each progressive block directly builds on the former. We will explain and give visualisations of the architecture further on, but so that the overall terminology is clear we have the smallest "block" which encompass the core operation of Cloud Transformers which are called *Cloud Transforms*. These serve as the fundamental unit of the processing pipeline. Multiple Cloud Transforms can be grouped together and processed in parallel, forming what we refer to as *multi-head processing blocks*. These blocks can then be stacked sequentially to create larger structures, known as *cascaded multi-head processing blocks*, these adjust spatial resolution and the number of feature channels at different stages. Finally, we use these cascaded blocks to construct full architectures, called *cloud transformers*, designed to handle a variety of point cloud processing tasks.

1.4 A Recap Of Rasterisation

We have seen and used rasterisation in Cloud Compare during the course and here we give reminders of this process to help discuss the cloud transform process in the following subsection. Rasterisation is the process of transferring an unordered point cloud onto a 2D or 3D (Voxel) grid. Each point in a point cloud consists of two key components:

1. **Position** $p_i \in \mathbb{R}^3$: This represents the spatial coordinates of the point in 3D space.
2. **Feature Value** $v_i \in \mathbb{R}^N$: This contains additional information about the point, which will be rasterised onto a structured grid. We have often see this during the course as containing the Normals.

Rasterisation is simply the process of projecting the Point Cloud onto a 2D or 3D grid, and representing the feature values on this grid.

1.4.1 Cloud Transform

Now we can look more in depth at each section of the framework, starting with the Cloud Transform we try to explain these methods heuristically without all the mathematical details so as to demonstrate understanding and steer clear of plagiarism. The overall aim of the Cloud Transform is to apply a convolution to the point cloud, and output a point cloud of the same cardinality, but different feature dimensions. However as discussed in the related literature section, in particular Wu et al. [2020] there are issues applying these point clouds. The Cloud Transform (CT) implements a 3 step method to overcome these issues, the CT rasterises, then applies a convolution and then finally de-rasterises the point cloud.

To process the point cloud efficiently, the transform first projects it onto a structured grid using a projection that will later be sufficiently regular to be learnable by a single layer perceptron:

- In **2D**, this results in a feature map of size $w \times w$ with c_{in} channels.
- In **3D**, it produces a volumetric representation of size $w \times w \times w$.

This step ensures that the originally unordered point cloud is mapped to a regular grid where further processing can be applied using convolutional techniques.

For the rasterisation step, we need some method of projecting the point cloud position noted p_i onto the grid. To determine where a point should be placed in the grid, a function predicts a transformation of the original spatial positions p_i. Instead of using a direct matrix transformation, the model **learns residual adjustments** to each point's position

and applies a learnable transformation T. This ensures that the projected key positions k_i capture fine-grained spatial information.

For a 2D setting, these positions are projected onto a plane, and a sigmoid activation is applied to constrain them within a normalized range $[0, 1]$, ensuring they fit within the grid. In the paper these positions are called keys, noted k_i, and represent a transformation of the position vectors

Once the key positions k_i are determined, the corresponding feature values v_i are placed onto the structured grid. Instead of mapping them to a single grid cell, the values are distributed across multiple neighboring cells using a technique called bilinear interpolation.

Each key position falls within a specific grid cell, but instead of snapping to the closest grid point, it spreads its value across four nearby pixels (or eight in 3D) via a weighting system. The process works as follows:

- We compute weights based on how close the key position is to each neighboring grid point.
- These weights determine how much of the feature value should be assigned to each pixel.
- This ensures a **smooth assignment** of values across the grid, preventing loss of detail.

To handle cases where multiple points are projected into the same grid region, the method uses a maximum aggregation approach. Instead of summing or averaging overlapping values, it takes the maximum value in each grid cell. Although this is a surprising method, since it loses information about all points but the most "intense" results show that it improves performance significantly, likely by preserving the most important features at each location.

1.4.2 Applying Convolutions

Once rasterized, the feature map undergoes convolutional processing, either using a single convolution or a more complex combination of layers. The processed feature map $\tilde{I}$ has the same spatial dimensions as the original but may have a different number of channels, allowing it to extract richer feature representations. This layer seems to work in the same way as in a classic CNN.

1.4.3 De-Rasterization: Converting the Grid Back to Points

After feature extraction via the convolution, the final step is de-rasterisation, which converts the structured feature map back into a transformed set of values V_e. This step ensures that information flows between points that were projected close to each other in the grid.

This mechanism resembles a self-attention mechanism but operates adaptively based on spatial proximity. Instead of attending to every point globally, it focuses on those that map to similar positions in the rasterized grid, making it an efficient and sparse attention mechanism.

1.4.4 To what extent is this similar to self-attention

The Cloud Transform shares key similarities with self-attention by adaptively aggregating information between points based on learned key positions. Instead of computing pairwise attention scores, each point is projected onto a structured grid via a learnable transformation, and interactions occur between nearby projected locations. This introduces a form of adaptive sparsity, similar to sparse attention mechanisms, where only spatially close points exchange information. Unlike traditional self-attention, which uses direct query-key matching, the Cloud Transform relies on grid-based feature aggregation through bilinear interpolation, followed by convolutional processing. This structured approach enables efficient and flexible point cloud processing while maintaining the principles of selective information exchange seen in attention models.

1.4.5 Managing exploding gradients

Similar to the issue in Recurrent Neural Nets, the authors found that sequentially stacking Cloud Transform blocks can cause exploding gradients during the learning step with back propagation. The authors coined what they call the "Gradient Balancing trick", where they divide the partial derivative in the back propagation by the dimension of the grid, w, leading to:

$$\frac{\partial L}{\partial k_i} \leftarrow \frac{1}{w} \frac{\partial L}{\partial k_i}$$

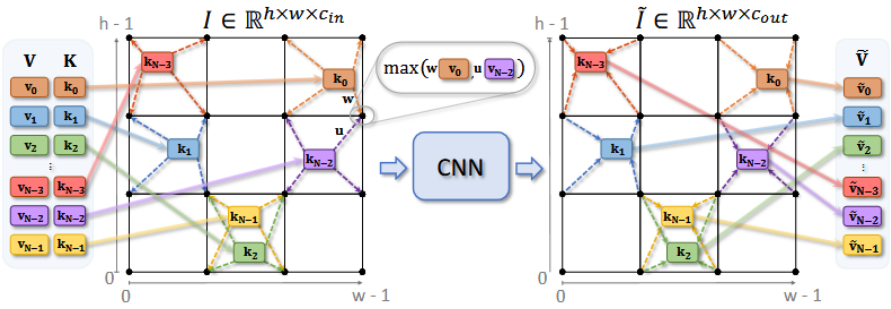

Figure 1: A visualisation of the entire CT workflow. Rasterisation and bilinear interpolation (Left), showing the max operation when 2 points projected onto the same node. Transform via the convolutional layer (Middle) and derasterisation (Right)

1.5 Multi-Headed Cloud Transform Block

A key challenge in processing point clouds is that when points are projected onto a structured grid, some level of information loss is inevitable. This is due to the finite resolution of the grid used for rasterization, which limits the number of available positions for representing data. In our approach, we typically use a resolution of up to $w = 128$ for 2D grids and $w = 32$ for 3D grids. Since finer details may be lost due to these discretisation constraints, we mitigate this by introducing a multi-head processing strategy, inspired by self-attention mechanisms in Transformer networks and multi-view convolutional networks.

Each head in this multi-headed setup operates as an independent cloud transform module, predicting its own key positions and feature values separately. These heads can vary in spatial resolution and even dimensionality. For example, some heads may operate on a 2D grid, while others function on a 3D grid. This flexibility allows the model to capture features at different scales and levels of granularity.

To form the final output of the multi-head cloud transform (MHCT) block, the contributions from all parallel heads are summed together for each input point. This ensures that each input feature vector x_i is transformed into a new feature vector y_i of the same size while incorporating diverse perspectives from multiple grid resolutions.

After summation, the MHCT block applies a normalisation layer followed by a ReLU activation function to improve stability and non-linearity. Additionally, a residual skip connection is introduced, allowing the input features to be directly passed to the output and facilitating gradient flow during training. This residual connection is crucial in preventing vanishing gradient issues, especially in deeper architectures.

Structurally, the MHCT block shares similarities with two well-known architectures:

1. **Inception Blocks** – These use parallel convolutional paths of varying kernel sizes, similar to how MHCT integrates multiple spatial resolutions.

2. **ResNeXt Networks** – These use grouped convolutions with multiple small independent processing units, resembling our use of multiple independent cloud transform heads.

For the implementation in the article, they employed 16 two-dimensional heads and 16 three-dimensional heads, ensuring a balanced mix of feature extraction across different spatial configurations. This combination effectively enhances the model's ability to process point cloud data while minimizing the limitations imposed by grid discretisation. However due to our computational constraints, we unfortunately are forced to carry out our implementation on a smaller scale that we will discuss in the next section.

1.6 Cascaded Blocks and Cloud Transformers

In short Cascaded blocks are made of 3 MHCT blocks stacked sequentially. These CMHCT are then in turn stacked sequentially to for the Cloud Transformers. We can stack and join CMHCT blocks in different ways to build architectures that are appropriate for slightly different tasks. In the interest of having appropriate space to discuss our implementation and proposals, we will skip the details of all the different architectures and we will discuss the particular architectures we implement in the following section.

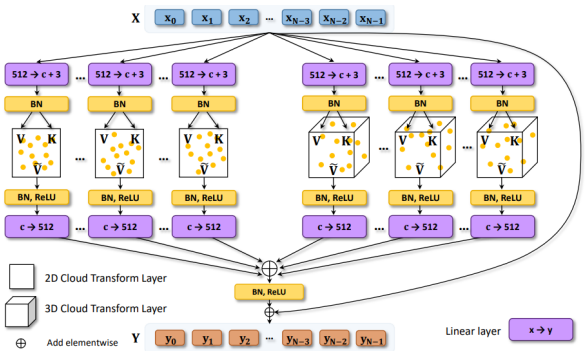

Figure 2: Visualisation of the multi-headed nature of the MHCT block. Each head is made of a Cloud Transform as described above

2 Implementation and Results

2.1 Foreword

This work focuses on two tasks—classification (ModelNet40, ScanObjectNN) and cloud completion (ShapeNet)—using a single RTX 3080Ti (12GB VRAM). The original Cloud Transformer architecture is large and was trained across multiple GPUs; reproducing its full-scale results on a single GPU would require days of training. Instead, we will use smaller models that still aproaches the results showcased in the paper.

2.2 Implementation

For the implementation, we build on the official Cloud Transformers repository[1], rewriting key modules (MultiHead_Union, MultiHead_Pool, MultiHead_AdaIN, classification_model, completion_model) for clarity and adding gradient checkpointing to lower peak memory usage. Dataset loaders (in `data_processing`) apply online augmentations—random rotation about the up axis, clipped Gaussian noise, and normalization—to improve generalization. Training scripts use Adam with a ReduceLROnPlateau scheduler, and mixed precision for ShapeNet; all model components and optional modifications are configurable via command-line flags. The experiments can be found inside the project jupyter notebook.

2.3 Classification

The classification model uses a streamlined Cloud Transformer backbone consisting of four sequential Cascaded Multi-Head Cloud Transform (CMHCT) layers followed by a multi-headed pooling layer that aggregates per-point features into a single global representation. A secondary segmentation branch predicts a foreground/background mask, critical for ScanObjectNN's noisy bounding-box inputs so the network learns to ignore background points (see Figure 3).

We evaluate on ModelNet40 (12,311 objects, 40 classes) and ScanObjectNN (2,902 real-world objects with background noise, same split as the original paper). By halving CMHCT depth (2 layers), dimension of the tensor (256 instead of 512) and number of heads (16 to 8) to cut compute and VRAM usage, our small implementation achieves 89.34% overall accuracy on ModelNet40 and 82.27% on ScanObjectNN (versus 93% and 85% reported in the paper with a full-sized model) .

2.4 Cloud completion

The goal of cloud completion is to take an incomplete, sparse partial scan of an object and predict a dense, high-resolution reconstruction of its full 3D shape, expanding from less than two thousand input points to a complete cloud of 16,384 points, by learning to infer and synthesize the missing geometry based on learned shape priors.

The cloud-completion model is an encoder-decoder Cloud Transformer: its encoder is identical to the classification backbone (a point-wise MLP plus four cascaded multi-headed Cloud Transform blocks), making the overall

[1] `https://github.com/SamsungLabs/cloud_transformers`

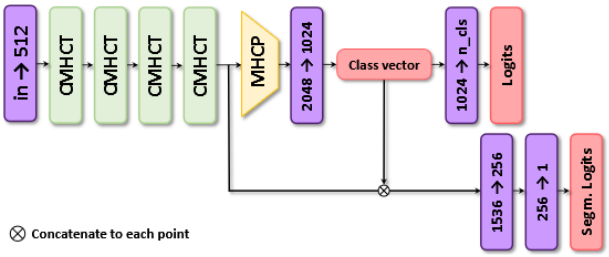

Figure 3: Classification Architecture

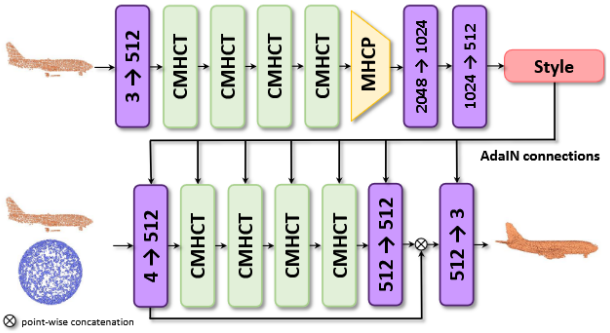

Figure 4: Cloud Completion Architecture

network twice as large as the classification model. The decoder conditions on the 512-dimensional style vector (via AdaIN), ingests both the partial ShapeNet scan and randomly sampled points from a unit sphere (each tagged by source), processes them through four CMHCT blocks, and outputs a fixed 16,384 point completed cloud via a final MLP (see Figure 4).

It is trained on ShapeNet's eight categories (~30000 objects, ~240000 partial views) using a combined Earth Mover's Distance and Chamfer loss.

Training with the same parameters and dataset as the paper would take actual days to complete. Instead, we made the choice of only keeping the Airplane class, with about 30000 partial views, to get results in a timely manner, while using a very small network (Tensor of size 512, 12 Heads, 1 Layer, half size grids). Furthermore, the article reconstruct cloud of 16384 points, while our reconstruction only have 8192 points.

With this setup, we achieve an average Chamfer Distance of 6.749 and an average F1@1% score of 0.8478 over the airplane class. The paper achieves an average Chamfer distance of 3.39 and an average F1@1% score of 0.752 over the full Shapenet dataset with a full-sized model. You can find some examples of reconstruction with our model in the following (see Figure 5).

3 Modifications of the original Cloud Transformers

In this section, we propose several modifications to the original model with an implementation (via flags). Since training the Cloud completion model is very long, we focus on classification to evaluate our ideas.

3.1 Soft-Max Weighted Aggregation

Rather than taking a hard element-wise maximum across heads, we apply a learnable softmax weighting during rasterization. This produces a smooth, differentiable aggregation of head outputs, letting the network dynamically assign importance to each head's features at every grid cell. Something similar is done in the ablation study of the research paper, but instead they try Mean aggregation and Sum aggregation. In both case it gave a worse result than the max aggregation. unfortunately, our method also seems to give slightly worse results on ModelNet10 with a small model with a final accuracy of 91.08% vs 92.40% (see Figure 6).

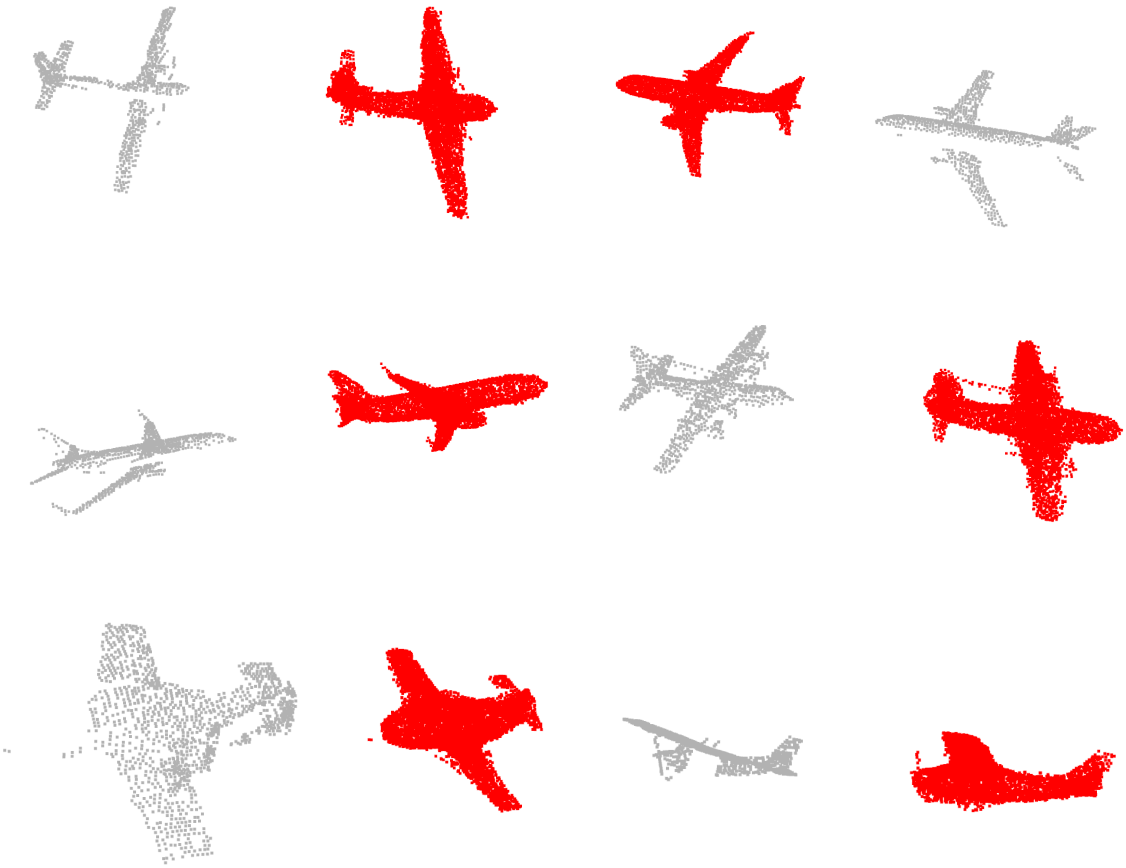

Figure 5: Cloud completion examples from our model

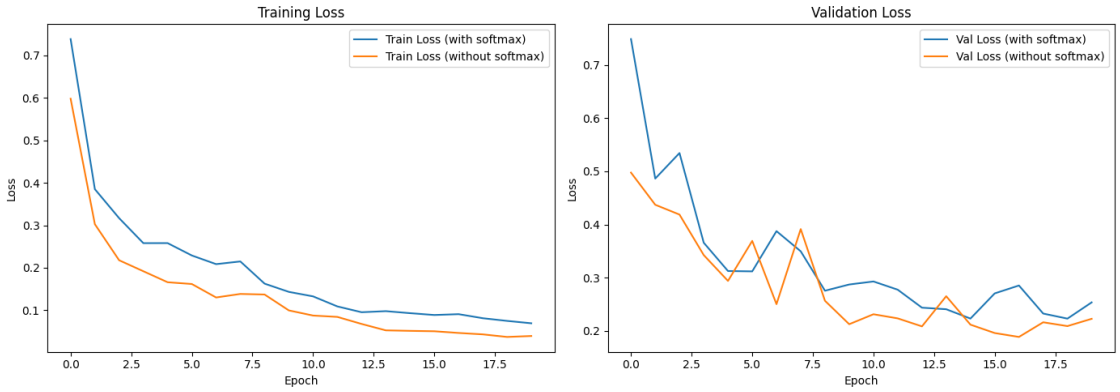

Figure 6: Comparison between Soft-max aggregation and max aggregation

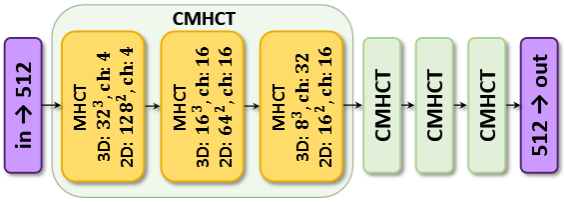

Figure 7: CMHCT block composition

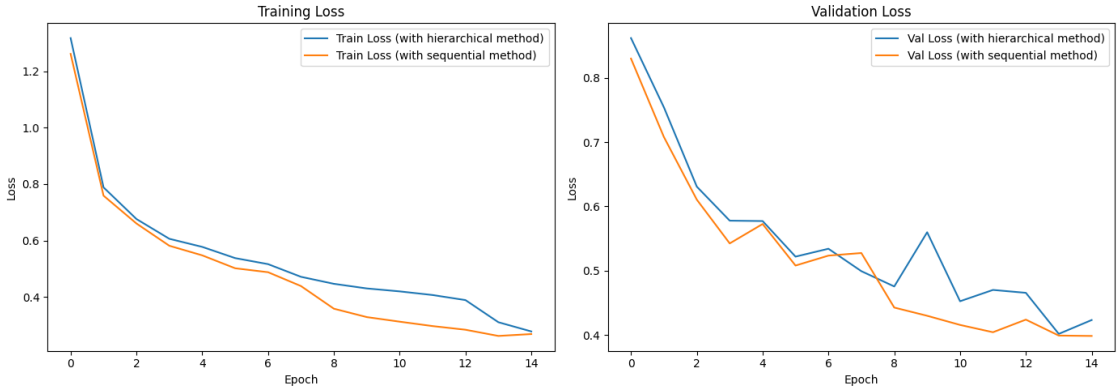

Figure 8: Comparison between Hierarchical and Sequential CMHCT blocks

3.2 Layer Norm in CMHCT Blocks

We add channel-wise LayerNorm between each CMHCT block. By normalizing across feature channels for each point, training becomes more stable—especially with small batch sizes on a single GPU—and less sensitive to batch statistics. The evaluation of this method was not conclusive, but we did a bigger training with this method and overral, loss were much more stable. However, it is not clear if it improves final accuracy as it might make the model a bit less expressive. We could also consider replacing the batch norm used in the different layers and replace it with a layer norm. It might be very beneficial since in the paper's original code, they usually use very small batches (less than 8).

3.3 Hierarchical Attention instead of Sequential

Instead of stacking MHCT blocks within the CMHCT layer one after another sequentially (see Figure 7), we use a single hierarchical attention module that attends at four spatial resolutions in parallel (instead of 3 normally) and fuses them via learnable gates. This multi-scale fusion should capture both local detail and global context more efficiently while preserving expressiveness. Basically, instead of putting the input sequentially through 3 MHCT, we instead compute 4 MHCT with different grid sizes in parallel before fusing them. With this method, we achieve a similar (or slightly worse) score as the original model on ModelNet40 on a small model : 87.84% vs 88.45% (see Figure 8).

4 Critical analysis

4.1 Information Loss in Rasterization

One of the key limitations of the Cloud Transform architecture is the potential loss of information during the rasterization step. The process of mapping an unordered set of points into a structured grid inherently involves discretization, which can lead to a loss of fine-grained spatial details.

Given an input point cloud:

$$X = \{x_1, x_2, \ldots, x_N\}, \quad x_i \in \mathbb{R}^f,$$

each point x_i is associated with a predicted key position k_i and a feature value v_i. The rasterization function projects these values onto a structured feature map:

$$P_2 : X \to I \in \mathbb{R}^{w \times w \times c_{\text{in}}}$$

in the 2D case, or onto a volumetric grid:

$$P_3 : X \to I \in \mathbb{R}^{w \times w \times w \times c_{\text{in}}}$$

in the 3D case. However, this projection inherently introduces errors due to two primary factors:

- **Discretization Artifacts:** The rasterization step maps continuous spatial positions into a discrete grid with a fixed resolution w. If w is not sufficiently large, fine details in the point cloud may be lost because multiple points may be mapped to the same grid cell, leading to feature aggregation and potential over-smoothing.
- **Sparse-to-Dense Mapping Issues:** In cases where the input point cloud is highly non-uniform, some regions may contain many points while others may be sparsely populated. The rasterization process does not always preserve the local density variations effectively, which could lead to an over-representation of dense regions and under-representation of sparse regions.
- **Aliasing Effects:** Since point positions are continuous but the target grid is discrete, certain points may be mapped to incorrect neighboring grid cells, introducing distortions in feature representation. This issue is exacerbated when using lower grid resolutions.
- **Interpolation Bias:** The use of bilinear interpolation for feature assignment spreads information across neighboring cells, which may introduce a smoothing effect that reduces the model's ability to capture sharp boundaries and fine structural details in the point cloud.

Although increasing w (the grid resolution) can mitigate some of these issues, doing so significantly increases computational and memory costs. This trade-off between resolution and efficiency is a fundamental limitation of rasterization-based approaches. Additionally, compared to sparse convolutional methods, which operate directly on the raw point cloud without requiring rasterization, the Cloud Transform may struggle to retain intricate details in high-precision point cloud tasks.

4.2 Lack of Explicit Global Context Modelling

One of the primary limitations of the Cloud Transform architecture is its reliance on local feature aggregation, which may hinder its ability to capture long-range dependencies across the entire point cloud. While rasterisation and subsequent convolutions facilitate structured processing, they inherently focus on local regions of the grid, limiting the model's capacity to explicitly encode global context.

- **Local Feature Aggregation:** The Cloud Transform primarily operates by mapping unordered points onto a structured grid and applying convolutional operations. Since convolutions have a fixed receptive field, they can only aggregate information from a limited neighbourhood in each layer. Although stacking multiple layers increases the effective receptive field, the propagation of global information remains indirect and constrained by the depth of the network.
- **Comparison with Transformer-Based Approaches:** Transformer-based architectures, such as Point Transformer and ViTs for point cloud processing, leverage self-attention mechanisms that enable direct interactions between all points, regardless of their spatial proximity. This allows for explicit global context modelling, which is particularly beneficial for capturing long-range dependencies and complex geometric structures within large-scale point clouds.
- **Inefficiency in Capturing Non-Local Dependencies:** Since the Cloud Transform relies on discretising the point cloud into a structured grid, information propagation is dictated by the connectivity of the convolutional network. Unlike self-attention, which computes relationships between all points simultaneously, the Cloud Transform may struggle to effectively capture dependencies between distant regions, leading to potential information loss in tasks that require a holistic understanding of the entire scene.
- **Potential Mitigations:** To address this limitation, hybrid architectures could be explored, incorporating self-attention layers alongside the Cloud Transform to facilitate global feature aggregation. Alternatively, non-local operations or graph-based message passing mechanisms could be integrated to enhance global context modelling while retaining the computational efficiency of rasterisation-based processing.

While the Cloud Transform provides an efficient mechanism for structured feature extraction, its reliance on local operations may limit its effectiveness in scenarios requiring detailed global reasoning. Future research directions could explore ways to incorporate global attention mechanisms while maintaining the efficiency advantages of the Cloud Transform.

5 Conclusion

During this project in Section 1, we have demonstrated a thorough understanding of the target material. In Section 2, we have done our own implementation to obtain similar results as the paper, while using models that are on average 8 times smaller. Furthermore we have both **suggested and implemented variants of this method not considered by the original paper**, notably considering a soft-max weighted aggregation, the introduction of a LayerNorm between each CMHCT block and introduction Hierarchical attentation instead of sequential. Finally in Section 3, we have discussed some critical analysis and limitations of the method, fulfilling the sections of the marking criteria.

References

J.-B. Cordonnier, A. Loukas, and M. Jaggi. Multi-head attention: Collaborate instead of concatenate, 2021. URL `https://arxiv.org/abs/2006.16362`.

D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, 2015. doi: 10.1109/IROS.2015.7353481.

K. O'Shea and R. Nash. An introduction to convolutional neural networks, 2015. URL `https://arxiv.org/abs/1511.08458`.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds, 2020. URL `https://arxiv.org/abs/1811.07246`.

H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun. Point transformer, 2021. URL `https://arxiv.org/abs/2012.09164`.