[NPM3D Project] - Point Transformer

Adrien Golebiewski

Contents

Introduction

Point clouds processing has become more and more popular as new applications arise: autonomous driving,video-games, town planning, or augmented reality to name a few of them. However, the irregular domain and lack of ordering make it challenging to design deep neural networks suited for point clouds.

In the recent years, standard neural architectures, such as convolutional neural networks (CNN), have shown encouraging results for structured data in different types of application. In this context, several pointset processing approaches attempt to transform the points into regular representations such as voxel grids or rendered views of the point clouds. However, the transformation of the point sets leads to loss of shape information and suffer from high computational complexity. On the other hand, some architectures act directly on the point set. This is in this particular context that Qi et al. introduces differents types of networks with PointNet and PointNet++.

Inspired by the recent progress of Transformer-based architectures in image vision, Hengshuang Zhao et al. propose, thought an article, to extend its use to point cloud processing. Transformers originated from NLP where attention mechanisms related words based on the importance of a scalar value. Since the output attention feature of each word is related to all input features, the network can learn the global context.
With 631 scientific quotations, the success of this paper was real in the Point cloud community in 2021. And it is still taken up by many studies to this day.

The key idea of the presented paper is to use the fact that Transformers, unlike the convolution, are fundamentally set operators, rather than a sequence operator. They are permutation invariant and do not attach stationary weights to specific locations, which makes them very well suited for point clouds processing. The method presented here uses a different kind of self-attention mechanism which consists in applying self-attention locally and to use vector attention rather than scalar attention.

As there is no official implementation or pretrained models yet, I first decided to focus on implementing the method on PyTorch, on Google Colab. Then I tried to reproduce the results for shape classification on ModelNet40. To sum up, I followed this strategy below :
• Implementation of Point Transformer for shape classification on PyTorch.

• Test of the implementation on the ModelNet40 classification task and try to obtain the same scores as Zhao et al. by finetuning the model.

• Study of the influence of some parameters namely the role of the Transformer dimension and the optimizer.

• Writing the report and discussion of the results.

1 Working bases

1.1 Point Cloud Processing

Point clouds are unordered sets of points with variable cardinality. They are grid-structured unlike images. Thus, it is no relevant to apply neural networks architectures on 3D data. Nevertheless, several approaches have been studied in the past few years, that could be categorized as follows :

Voxel-based approaches : Some approaches rely on transforming the point sets into an ordered representation, such as voxel grids. Then, 3D CNNs can be applied to the voxel-based representation and imply a reduction of the resolution.The available information since multiple points can be combined into one single voxel which can damage important spatial relations. Furthermore, transforming the point cloud into voxels increases memory requirements due to the sparsity of the points.

To remedy this, we have, **View-based approaches**. Some research has been conducted on rendering point clouds into 2D images. Since shape information can be occluded by rendering point clouds from a specific view-point, multi-view approaches have been proposed that render multiple images

from different angles. Even though images are rendered from different views, the model still fails to capture all geometric and spatial relations.

The last approach is a **Point-based approach**. The goal is to process each point individually with a multi-layer perceptron (MLP) and then fuse the representation to a vector of fixed size with a set pooling operation over a latent feature space. The motivation is that pooling is a symmetric function that is permutation invariant. We have the PointNet++ which differs from PointNet from the fact that features are pooled locally.

1.2 Attention and Transformers

Attention mechanism comes from natural language processing research. First, recurrent neural networks (RNN) were used for machine translation applications, where the last hidden state is used as the context vector for the decoder to sequentially produce the output. The problem is that dependencies between distant inputs are difficult to model using sequential processing. Therefore, Bahdanauet al. introduced the attention mechanism that takes the whole input sequence into account by taking the weighted sum of all hidden states and additionally, models the relative importance between words.The key equation below describes this attention mechanism :

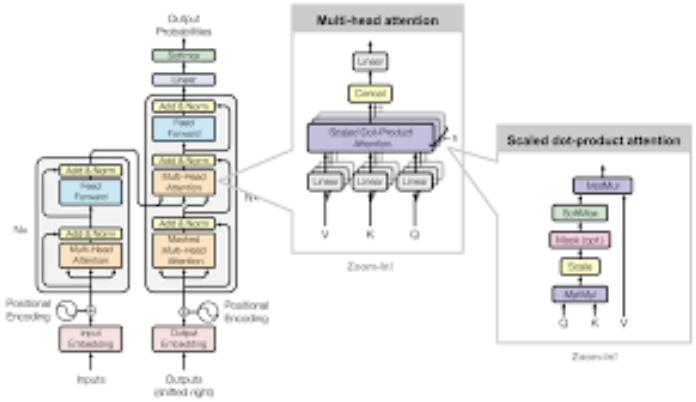

Figure 1: Full model architecture of Transformer

This mechanism is improved by introducing multi-head attention and proposing an encoder-decoder structure that solely relies on attention instead of RNNs or convolutions (see 1). In the work of Hengshuang Zhao et al., multi-head attention is **the basis for Point Transformer**.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

This architecture in particular has been part of the heart of the neural network research in the last few years, from NLP to computer vision.

2 The Point Transformer

In this section, I quickly present the main information learned from the paper about the point Transformer :

2.1 Point Transformer Layer

The key component of a Point Transformer network is the **Point Transformer layer** which takes as input a set of points with their associated features, and to return the same set of points but with new features produced by the attention mechanism. As mentioned above, this attention mechanism is not exactly the same as shared previously. The equation can be rewritten like this :

$$\mathbf{y_i} = \sum_{\mathbf{x_j} \in \mathcal{X}} \rho \left(\varphi \left(\mathbf{x_i} \right)^T \psi \left(\mathbf{x_j} \right) + \delta_{i,j} \right) \alpha \left(\mathbf{x_j} \right)$$

where $\mathcal{X} = \{\mathbf{x_i}\}_i$ is the set of input features, $\mathbf{y_i}$ is the i^{th} output feature. Here, φ, ψ and α respectively play the role of query, key and value of the equation 1, at which was added a positional encoding δ. It allows to specify geometric information in the attention layer and to adapt to local structures. Here $\delta_{i,j}$ is defined as $\theta \left(\mathbf{p_i} - \mathbf{p_j} \right)$ where $\mathbf{p_i}$ is the 3D coordinates of point i.ρ is a normalization function, usually softmax that allows to get coefficients in $[0, 1]$ for scalar attention.

From that, the intuition is to use vector values for the attention weights : they modulate the input features on a finer scale. The following equation describes they called "vector attention".

$$\mathbf{y_i} = \sum_{\mathbf{x_j} \in \mathcal{X}} \rho \left(\gamma \left(\varphi \left(\mathbf{x_i} \right) - \psi \left(\mathbf{x_j} \right) + \delta_{i,j} \right) \right) \odot \left(\alpha \left(\mathbf{x_j} \right) + \delta_{i,j} \right)$$

where γ produces attention vectors and $\odot$ denotes the Hadamard product of the 2 vectors.

Finally, since the attention mechanism is an operator that acts on sets, one can choose to use a smaller set of features rather than the whole set. The authors decided to apply the attention locally, meaning that only the k nearest neighbors of $\mathbf{x_i}$ on the 3D space are taken into account in the computation of the output features.

The mappings are linear transformations, while $\delta \approx \theta$ and γ are MLPs, this way everything can be learnt through back-propagation. The attention mechanism is a combination of linear projections that reduce the computational cost of propagation and back-propagation through the network, and a residual connection that has proved to be useful in several deep-learning tasks throughout the years.

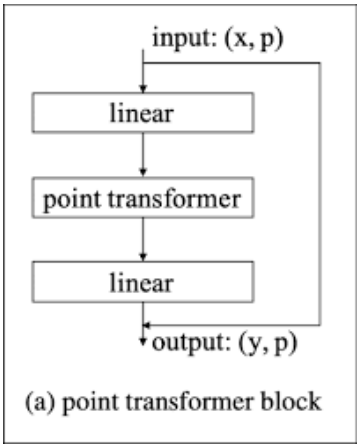

Figure 2: Point Transformer layer

2.2 Transition Down layer

An another important component of the architecture is the Transition Down layer which reduce the cardinality of the point set to a smaller one, allowing to decrease the complexity of the neural networks throughout the network. If $\mathcal{P}_1$ is the input set of points, the goal of this layer is to retrieve a new subset of points $\mathcal{P}_2 \subset \mathcal{P}_1$, while keeping accurate features for points in $\mathcal{P}_2$. First, a subset $\mathcal{P}_2$ is extracted from $\mathcal{P}_1$ using Farthest Point Sampling (FPS), reducing the cardinality from N to $N/2$ for instance. Then for each point in $\mathcal{P}_2$, we retrieve its k nearest neighbors in the set $\mathcal{P}_1$.

The extracted features go through an MLP layer and are max-pooled to obtain the feature vector of the point in $\mathcal{P}_2$. The Transition Down block is schematized below :

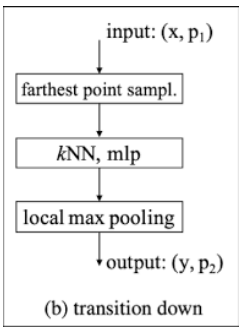

Figure 3: Transition Down layer

Similarly, for dense prediction tasks such as cloud segmentation, it could be useful to increase the cardinality of the point set. Zhao et al. define a Transition Up layer. As I focused on shape classification, this layer has not been implemented in my work.

2.3 Network Architecture for Shape Classification

Shown in the newt figure (figure 4 below), the network architecture for shape classification implies a succession of Point Transformer and Transition Down blocks.

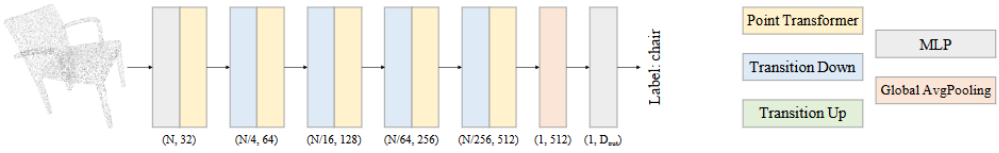

Figure 4: Point Transformer architecture for shape classification

First, we input the point cloud of size N, taking as features for each point the 3D coordinates. An MLP transforms the features into a vector of size 32 , which are again transformed by the first Point Transformer block. Then, the features go successively through a Transition Down layer that reduces the cardinality of the point set and increases by 2-fold the features' size. This operation is repeated 4 times. To finish, the network performs global average pooling over the features to get a global feature vector of size 512 for the whole point set.

It is in this case this deep network that I implemented. The next part of this report presents the strategy I adopted.

3 Implementation and experimentation

In this part, I present my works and how I implemented the network and how I trained it for the ModelNet40 classification task. Then, I realize some experimentations about the influence of some parameters, both of the model architecture and its optimization, on the training procedure and the test accuracy.

I attempted to implemented myself all the network and tried to keep as close as possible to the original one.

3.1 Data Processing on ModelNet40

The ModelNet40 dataset contains 9,843 models CAD models in the training split and 2468 in the testing split. There are 40 object categories. Since they are CAD models, one must first extract point clouds from it.

In our context, I perform farthest-point-sampling (FPS) to get an uniform sampling of all the nodes and retrieve a smaller point cloud made of 1024 points. We have the need to do FPS at each time we draw a new batch in the data loader, and substantially reduces the computational complexity.

3.2 Network Implementation setup

Below, we have the details of what I used for them, as suggested by the original paper.

- φ, ψ, α : Linear (d, d) (the weights are not shared between the 3 functions)
- δ : Linear $(3, d) \rightarrow$ ReLU $\rightarrow$ Linear (d, d)
- γ : Linear $(d, d) \rightarrow$ ReLU $\rightarrow$ Linear (d, d)
- Linear projection of the Point Transformer block: Linear (d_{feature}, d)
- MLP of the Transition Down layer: Conv $1d\,(d_{\text{feature}}, 2d_{\text{feature}}) \rightarrow$ BatchNorm $\rightarrow$ ReLU
- First MLP of the full model:

  Linear $(3, 32) \rightarrow$ ReLU $\rightarrow$ Linear $(32, 32)$
- Last MLP of the full model:

  Linear $(512, 256) \rightarrow$ ReLU $\rightarrow$ Linear $(256, 64) \rightarrow$ Re$LU \rightarrow$ Linear $(64, 40)$

where $d_{\text{feature}} = 32, 64, 128$ or 256 depending on the depth of the block choosen in the network. Besides, I decided to study the parameter d ,the Transformer dimension. This experimental aspect is not presented in the original paper.

As suggested in the paper, I used $k = 16$ for the number of neighbors to be considered both in the local self-attention and the Transition Down layer. I also decided to go further by testing several values of k.

As we are in a classification framework,the CrossEntropy loss was used for training. The optimizer that is used in the original paper is the SGD one with momentum set to 0.9 and weight-decay set to 0.0001 . A study about the influence of the optimizer is later on the report.

3.3 First Results with the Vanilla Implementation

Firstly, I used the same parameters as given in the paper for the optimization, with a transformer dimension $d = 4$, but obtained quite bad results (see figure below).

This is probably due to the authors having used different parameters for the network architecture.

When training for 200 epochs with the Adam optimizer with learning rate 0.001 dropped 10-fold at step 60 and 80 , the maximum accuracy that I obtained on the test dataset was barely better with 89.74%.

In a second time, I decided to change the optimizer and the learning rate to improve this results.

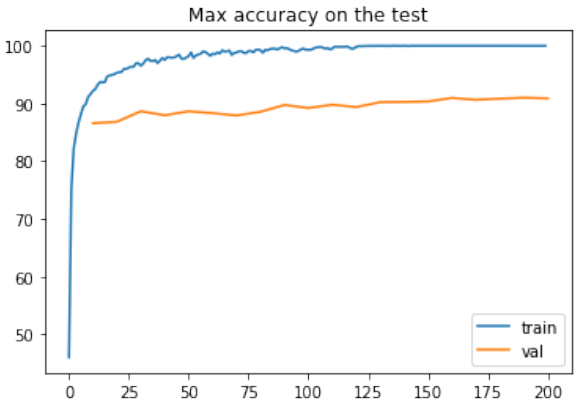

Figure 5: Accuracy on training and testing dataset for 200 epochs using SGD optimizer and $d = 64$

3.4 Impact of the Optimizer and the number of iterations

As mentioned in the previous section, the optimizer with same parameters as in the paper did not give good results in term of metrics. In this context, I decided to try out different optimizers with fixed learning rate $lr = 0.001$ and with a transformer dimension of 64 to see if there was any difference. The results are presented bellow in figure 6, 7 and in table 1

Optimizer	Adam	SGD	RMSProp
Maximum test accuracy	90.23	89.89	68.90

Table 1. Influence of the optimizer in the final accuracy of the model

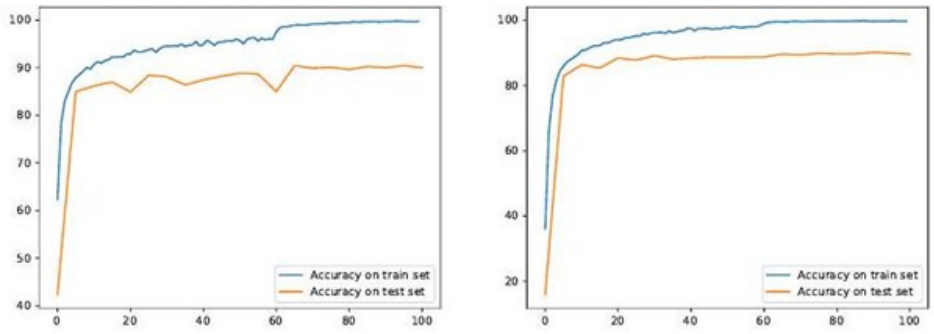

Figure 6: Accuracy on training and testing dataset for 100 epochs using different optimizers and a transformer dimension $d = 64$

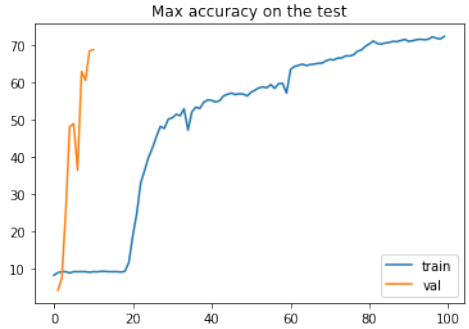

Figure 7: Accuracy on training and testing dataset for 100 epochs using RMSProp Optimizer $d = 64$

We can see that there is a difference between the behavior of the optimizers. Adam achieves better overall score than SGD but is less stable. RmsPropr seems to be much less efficient with an accuracy of the model not exceeding 70 percent.

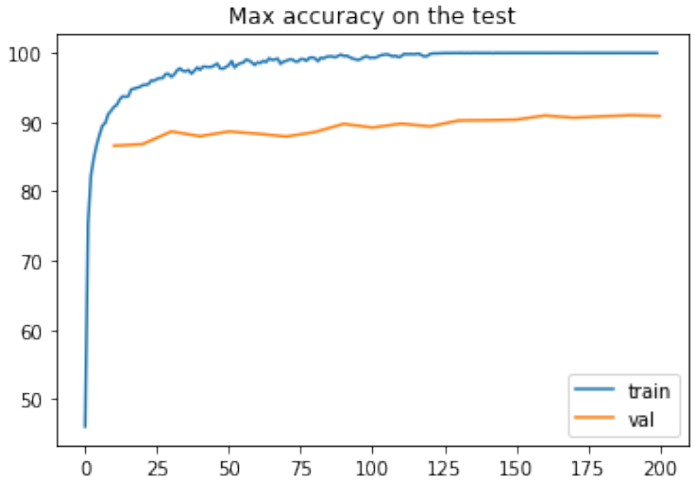

Figure 8: Accuracy on training and testing dataset for 200 epochs using SGD Optimizer and a transformer dimension $d = 32$

With a larger number of iterations (from 100 to 200), the test accuracy increases slightly (in the order of a tenth) as we can see in the figure above. The memory and computation time being very important with 200 epochs on Google Colab, the rest of the experiments were carried out on 100 epochs.

Besides, the Adam optimizer was chosen for the rest of the experiments.

3.5 Impact of the Transformer Dimension

The final study that I share is the impact of the Transformer Dimension d. The differents accuracies obtained for the training are presented in the figures below. Again, for the results to have been more statistically relevant, I should have repeated them over several iterations. However, a bigger dimension for the Point Transformer layer, i.e. 64, seems to give better result (the best I obtained).

Neverthless, if we increase this parameter, it significantly increases the time needed to train the network.

3.6 Perspectives and personnal comments

In addition to the implementation of this architecture from the paper, it seems relevant to me to define some points of perspectives and proposals for experiments and additional improvements.

Firstly, throughout the studies, I used $k = 16$ for the number of neighbors to be considered both in the local self-attention and the Transition Down layers as suggested by Zhao et al.

It would have been interesting to test other values of k. According to the paper, When the neighborhood is larger (in our case with k = 64), each self-attention layer is provided with a large number of datapoints, many of which may be farther and less relevant. This excessive noise into the processing lower the model's accuracy.

Secondly, it would have been interesting to try another type of layer for the MLP of the Transition Down block. For example, we could have tested 2D convolutions and changed this MLP to Conv $2d\,(d_{\text{feature}}\,, 2d_{\text{feature}}\,) \rightarrow$ BatchNorm $2d \rightarrow$ ReLU

The network is able to weight differently each point belonging to the neighborhood from which we retrieve the feature, whereas Conv $1d$ was equivalent to a linear layer whose weights were shared by all the points in the neighborhood.

This paper is nowadays a real reference for Transformer applied to 3D point clouds. From a personal point of view, study this particular paper was really interesting allowing me to discover how a "transformer" type architecture could be applied outside the classical use cases in NLP and computer vision. The ideas structure is really clear and I quite easy manage to follow the authors intuition.

However, I remark that some details could have been added in the paper to allow us to have a better understanding. For instance, the paper argues that vector attention outperforms scalar attention (see Section 4.4 of the paper), but they never share the dimension of the attention weights, although the study in 4.6 showed this parameter has crucial importance. Secondly, In my opinion, I lacked interpretation compared to the analyses done in the segmentation part that I found more complete.

4 Conclusion

The paper presented by Zhao et al. presents an adaptation of Transformers and self-attention to 3D point cloud processing. This method is one of the most recent technique about semantic scene segmentation, object part segmentation, and object classification point clouds.

In this project, I focused on trying to implement and reproduce the results on the ModelNet 40 classification task. I obtained 90.33% accuracy on the test dataset at most, which are far from 93.7% of the authors' implementation. I believe that with more hyper-parameter tuning and with bigger Point Transformer blocks (higher dimensional vector attention), it should be possible to improve my best model and to reduce the gap between my implementation'results and theirs.

However, with the memory, ram constraints, I could not increase this parameter without having code launch problems.

Consequently, it was hard to optimize well the hyperparameters, and very difficult to reproduce the training procedures used in this type of papers.

As a conclusion, further work would consist in comparing this paper to the more recent paper about **Fast Point transformer** in particular. This paper also introduces Transformers for Point Cloud processing but it consists of a new lightweight self-attention layer. The approach encodes continuous 3D coordinates, and the voxel hashing-based architecture boosts computational efficiency.

It could be very interesting to implement this other method and to understand the difference between this two types of Point Transformer.

5 References

[1] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3 d classification and segmentation. In $CVPR$, 2017.1, 2

[2] Nico Engel, Vasileios Belagiannis, and Klaus Dietmayer. Point transformer, 2020. 7

[3] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R. Martin, and Shi-Min Hu. Pct: Point cloud transformer, 2021.

[4] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Classification of point cloud scenes with multiscale voxel deep network. 04 2018. 1, 2

[5] Zhirong Wu, Shuran Song, A. Khosla, F. Yu, Linguang Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. CVPR, 2015. 2, 4

[6] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer, 2020. 1, 3, 4, 5, 6, 7

[7] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3 d object detection. In $CVPR$, 2018.1, 2

[8] Chunghyun Park, Yoonwoo Jeong, Minsu Cho, Jaesik Park. Fast Point Transformer.