NPM3D Project - MVA 2021
Point Transformer, by Hengshuang Zhao et al. (2020)

Pierre Fernandez
pierre.fernandez@polytechnique.edu

Abstract

In this work, I present Point Transformer [24], a novel deep learning technique for 3D data. It uses an attention mechanism similar to the one of the Transformer architecture, to construct networks specially suited for point clouds processing. The method is said to achieve state-of-the-art in different tasks such as shape classification or semantic scene segmentation. Here I try to reproduce the results of the authors and propose several studies and extensions of their research.

1. Introduction

Point clouds processing has become more and more popular as new applications arise: autonomous driving, town planning, video-games 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 (and 3D data in general).

Standard neural architectures, such as convolutional neural networks (CNN), have shown promising results for structured data [10] [9] and have been developed a lot in different types of application in the recent years. For that reason, several pointset processing approaches attempt to transform the points into regular representations such as voxel grids [25] [14] or rendered views [15] of the point clouds. However, transforming the point sets leads to loss of shape information. Furthermore, these methods suffer from high computational complexity due to the sparsity of the 3D points.

On the other hand, some architectures act directly on the point set. Qi et al. paved the way for these types of networks with PointNet [2] and PointNet++ [13]. They are designed for feature learning on point clouds by using multilayer perceptrons (MLPs), max-pooling and rigid transformations.

One last type of architecture [20] acts on graphs dynamically computed in each layer of the network. It allows to capture local geometric structure while maintaining permutation invariance.

Inspired by the recent progress of Transformer-based architectures in image vision tasks [12] [5] [18] [23], Hengshuang Zhao et al. propose to extend its use to point cloud processing. Transformers originated from NLP research [19] 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 is capable of learning the global context. 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 that has been developed in [23], which consists in applying self-attention locally and to use vector attention rather than scalar attention.

According to the authors, the Point Transformer achieves state-of-the art on large-scale semantic segmentation on the S3DIS dataset (70.4% mIoU on Area 5), shape classification on Model-Net40 (93.7% overall accuracy), and object part segmentation on ShapeNetPart (86.6% instance mIoU).

As there is no official implementation or pretrained models yet, I first decided to focus on implementing the method on PyTorch, inspiring me from parts of pre-existing codes found on GitHub. Then I tried to reproduce the results for shape classification on ModelNet40 and obtained an overall accuracy of 90.6% (as compared to the 93.7% presented in the paper). I also propose a variant of their architecture by incorporating convolutions in some modules of the network and achieved an overall accuracy of 91.3%. To sum up, my contributions in the project are the following:

- I implemented Point Transformer for shape classification on PyTorch.
- I tested my implementation on the ModelNet40 classification task and tried to obtain the same scores as Zhao et al. by finetuning my model.
- I proposed to study the influence of some parameters namely the role of the Transformer dimension, the op-

timizer. I also proposed to add convolutional layers instead of some linear ones in the network.

- I wrote down the report and discuss my results in the light of what I have done during the project.

2. Related Work

2.1. Point Cloud Processing

Point clouds are unordered sets of points with variable cardinality, and, unlike images that are grid-structured, they can be very irregular. Consequently it is hard to apply standard 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. The metric space is discretized into small regions called voxels, which are labeled as occupied if a point lies inside the voxel. Then, 3D CNNs can be applied to the voxel-based representation [21] [25] [14]. This preprocessing, however, reduces the resolution and 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 the memory requirements and is computationally heavy, due to the sparsity of the points in the space. To address these limitations, multiple extensions leverage the sparsity of 3D data, such as [3].

View-based approaches In contrast to building voxel grids, a lot of research has been conducted on rendering point clouds into 2D images. Then, working with traditional CNNs is possible. Since shape information can be occluded by rendering point clouds from a specific viewpoint, multi-view approaches have been proposed that render multiple images from different angles [15] [8]. Even though images are rendered from different views, the model still fails to capture all geometric and spatial relations. To this day, multi-view approaches achieve impressive results on standard 3D benchmarks. However, the transformation from sparse 3D points into images increases computational complexity as well as required memory.

Point-based approaches Another type of architectures followed from PointNet [2] and PointNet++ [13]. The main idea 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 behind the pooling operation is that pooling is a symmetric function that is permutation invariant. PointNet++ differs from PointNet from the fact that features are pooled locally. First, centroids of local regions are sampled using hand-crafted algorithms such as farthest-point-sampling (FPS), then local features are encoded to the centroids by exploring the local neighborhood with query ball or k-nearest-neighbors (KNN) grouping. Besides, since point sets are unstructured, an alternative active research area is the definition of convolution operations that can operate on irregular 3D point sets such as KPConv [16] or PointCNN [11].

2.2. Attention and Transformers

Attention 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. [1] 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. [19] improved the attention mechanism 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.

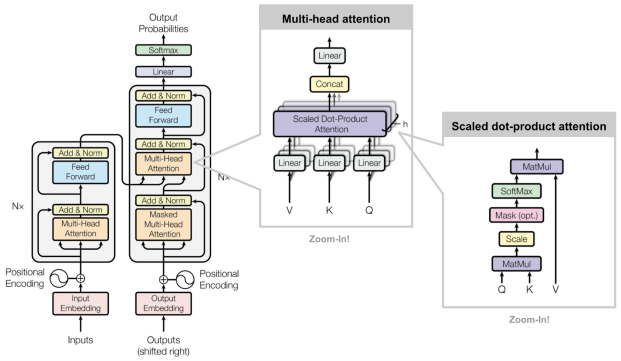

Figure 1. Full model architecture of Transformer (Image source Fig 1 & 2 from [19])

The key equation 1 describes the attention mechanism.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

It is important to note that this architecture has been at the heart of the neural network research in the last few years, from NLP to object recognition and computer vision [12] [5] [17] [18]. Many papers adapted it for their given tasks. For instance, the paper actually uses a variant of this equation developed in [23] that will be described in the next part of the report.

3. Presentation of the Point Transformer Method

3.1. Point Transformer Layer

As expected from the name of the article, the key component of a Point Transformer network is the Point Transformer layer. The goal of this layer is to take as input a set of points with their associated features, and to return the same set of points but with new features that were produced by the attention mechanism. As mentioned above, this attention mechanism is not exactly the same as 1. Indeed, first we can rewrite equation 1 as:

$$\mathbf{y_i} = \sum_{\mathbf{x_j} \in \mathcal{X}} \rho\left(\varphi(\mathbf{x_i})^T \psi(\mathbf{x_j}) + \delta_{i,j}\right) \alpha(\mathbf{x_j}) \tag{2}$$

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 that allows to specify geometric information in the attention layer and to adapt to local structures. Here $\delta_{i,j}$ is defined as $\theta(\mathbf{p_i} - \mathbf{p_j})$ 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 of [23] (by the same authors than the paper studied here) is to use vector values for the attention weights, rather than scalar ones. The authors argue that they are able to modulate the input features on a finer scale. The following equation describes they called "vector attention":

$$\mathbf{y_i} = \sum_{\mathbf{x_j} \in \mathcal{X}} \rho\big(\gamma\left(\varphi(\mathbf{x_i}) - \psi(\mathbf{x_j}) + \delta_{i,j}\right)\big) \odot \left(\alpha(\mathbf{x_j}) + \delta_{i,j}\right) \tag{3}$$

where γ is a function that produces attention vectors and $\odot$ denotes the Hadamard product of the 2 vectors. This attention mechanism is illustrated in 2

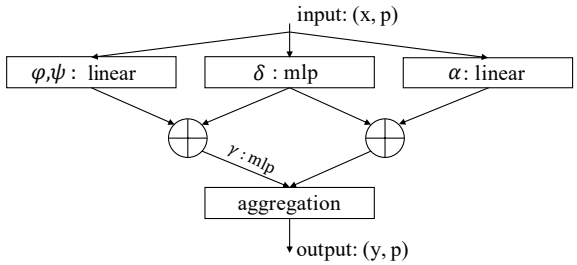

Figure 2. Point Transformer layer (Image source Fig. 2 from [24])

Finally, since the attention mechanism is essentially an operator that acts on sets, one can choose to use a smaller set of features rather than the whole set. The authors chose 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. This boils down to considering $\mathcal{X}_i$ (and not $\mathcal{X}$) in equation 3.

The mappings φ, ψ, α are linear transformations, while $\delta \approx \theta$ and γ are MLPs, this way everything can be learnt through back-propagation. The attention mechanism is combined with 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. The full Point Transformer block is schematized in 3.

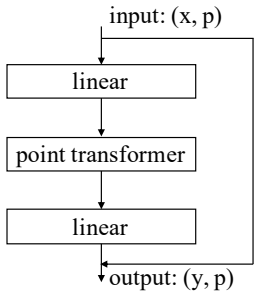

Figure 3. Point Transformer layer (Image source Fig. 3 from [24])

3.2. Transition Down layer

The Transition Down layer is another important component of the architecture that will be used later on. Its purpose is to 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 $\mathcal{P}_2$. To do so, the layer proceeds as Qi et al. in [13]. 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 in 4.

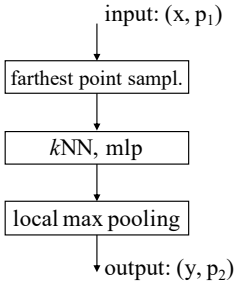

Figure 4. Transition Down layer (Image source Fig. 3 from [24])

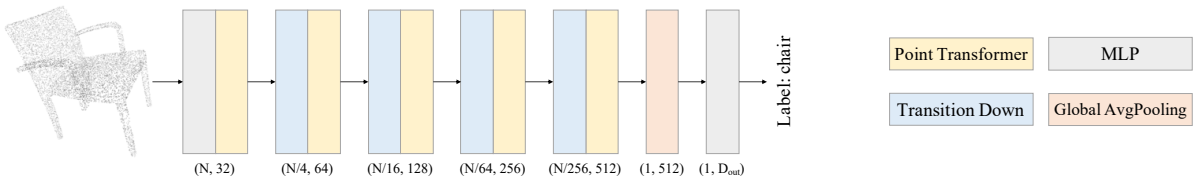

Figure 5. Point Transformer architecture for shape classification (Image source Fig. 4. from [24])

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. Since I focused on shape classification, this layer has not been implemented and is less relevant to the work done in the project so I refer the interested reader to Section 3.5 of [24] for further details.

3.3. Network Architecture for Shape Classification

The network architecture for shape classification is shown in 5. It consists in the succession of Point Transformer and Transition Down blocks. First, we input the point cloud of size N, taking as features for each point the 3D coordinates (and the normals if they exist). An MLP transform 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 by four-fold the cardinality of the point set and increases by 2-fold the features' size. This operation is repeated 4 times. Finally the network perform global average pooling over the features to get a global feature vector of size 512 for the whole point set. This global feature is passed through an MLP to get the global classification logits.

4. Implementation Details and Experiments

In this section I present the details of how I implemented the network and how I trained it for the ModelNet40 classification task. Then I present the influence of some parameters, both of the model architecture and its optimization, on the training procedure and the final accuracy.

Although the authors of the paper mention the fact that the code and the models will be released to the community, it has not yet been done. Therefore, I inspired myself from some part of the following repositories. https://github.com/yanx27/Pointnet_Pointnet2_pytorch [22]: a PyTorch implementation of PointNet and PointNet++ that has been useful for data preparation. https://github.com/qq456cvb/Point-Transformers, https://github.com/yzheng97/Point-Transformer-Cls and https://github.com/lucidrains/point-transformer-pytorch are unofficial implementation of the studied paper. They were useful for designing the Point Transformer layer and the final architecture. However, there are parts of their code I have not fully understood and that are far away from what is presented in the paper, so I implemented myself all the network and tried to keep as close as possible to the original one, while being, in my opinion, more straigthforward.

4.1. Data Pre-Processing on ModelNet40

The ModelNet40 dataset [21] 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. Qi et al. [13] created a new version of the dataset where each model constitutes a point cloud of 10.000 points with normals computed from the original faces. Here, we perform FPS to get an uniform sampling of all the nodes and retrieve a smaller point cloud made of 1024 points. The data processing implementation of [22] presents the advantage of pre-computing and saving which points are sampled for each point cloud. This removes the need to do FPS at each time we draw a new batch in the data loader, and substantially reduces the computational complexity. That is why I used this implementation.

4.2. Network Implementation

As mentioned in the previous section, many functions are either linear transformations of MLPs. Here are 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: $Conv1d(d_{feature}, 2d_{feature}) \rightarrow BatchNorm \rightarrow ReLU$
- First MLP of the full model: $Linear(3, 32) \rightarrow ReLU \rightarrow Linear(32, 32)$

- Last MLP of the full model:
  $Linear(512, 256) \rightarrow ReLU \rightarrow Linear(256, 64) \rightarrow ReLU \rightarrow Linear(64, 40)$

where $d_{feature}$ = 32, 64, 128 or 256 depending on the depth of the block in the network. d is the Transformer dimension: a parameter that is studied in my work (it was not presented in the original paper however).

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 layer, that is the number suggested by Zhao et al. For the interested reader, the authors present an ablation study of this parameter in Table 5 of [24].

Since the task is a classification one, I used the Cross-Entropy loss 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. The initial learning-rate is 0.05 and dropped by ten-fold at steps 120 and 160 when training on 200 epochs. I studied the influence of the optimizer later on the report.

4.3. First Results with the Vanilla Implementation and Data Augmentation

At first I used the same parameters as given in the paper for the optimization, with a transformer dimension $d = 4$, but obtained poor results (see figure 6). This is probably due to the authors having used different parameters for the network architecture. Be that as it may, I decided to change the optimizer and the learning rate.

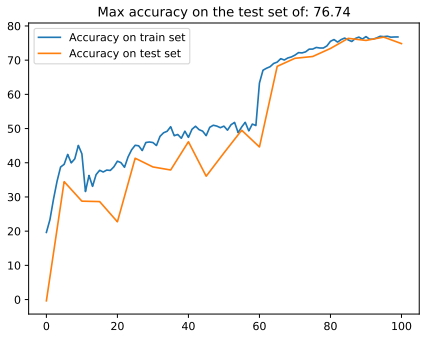

Figure 6. Accuracy on training and testing dataset for 100 epochs using SGD optimizer and a transformer dimension $d = 4$

When training for 100 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 89.74%. I did the same test using some data augmentation transformations, namely random rotation along the z-axis and random shuffle, but the resulting maximum accuracy decreased to 88.89%. However, we can observe in figure 7 that the training process seems more stable, with validation accuracies that are less fluctuating and with variations that look more like the training accuracies. The training process took around 1h30 and more than 2h when using data augmentations. One explanation in the drop of performance may be the fact that more iterations should be needed when using data augmentation. Here, since I do not have enough computing power for data augmentation to be really useful, I decided not use any in the following studies of the report.

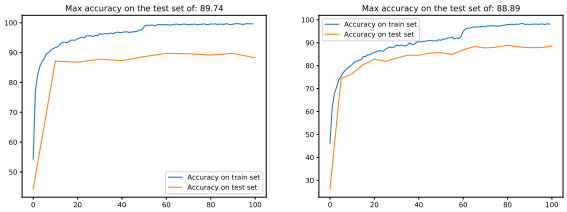

Figure 7. Accuracy on training and testing dataset for 100 epochs using Adam optimizer and a transformer dimension $d = 4$ (left: without data augmentation, right: with data augmentation)

4.4. Using Convolutions in Transition Down Blocks

Then, I tried another type of layer for the MLP of the Transition Down block. The idea comes from [22], where they use 2D convolutions when implementing the PointNetSetAbstraction class (file 'model/pointnet2_utils'). Therefore I changed this MLP to

$$Conv2d(d_{feature}, 2d_{feature}) \rightarrow BatchNorm2d \rightarrow ReLU$$

This results in the network being able to weight differently each point belonging to the neighborhood from which we retrieve the feature, while $Conv1d$ was equivalent to a linear layer whose weights were shared by all the points in the neighborhood. The results improved a little, see 8. For the rest of the report, we now consider that the layer is made of 2D convolutions.

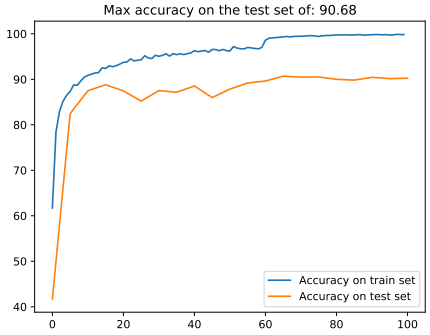

Figure 8. Accuracy on training and testing dataset for 100 epochs using Adam optimizer and a transformer dimension $d = 4$

4.5. Impact of the Optimizer

As I mentioned earlier, the optimizer with same parameters as in the paper did not give satisfactory results. Therefore I decided to try out different optimizers with fixed learning rate $lr = 0.001$ and with a transformer dimension of 32 to see if there was any difference. The results are presented bellow in figure 9 and in table 1

Optimizer	Adam	SGD	MadGrad
Maximum test accuracy	90.43	90.19	91.08

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

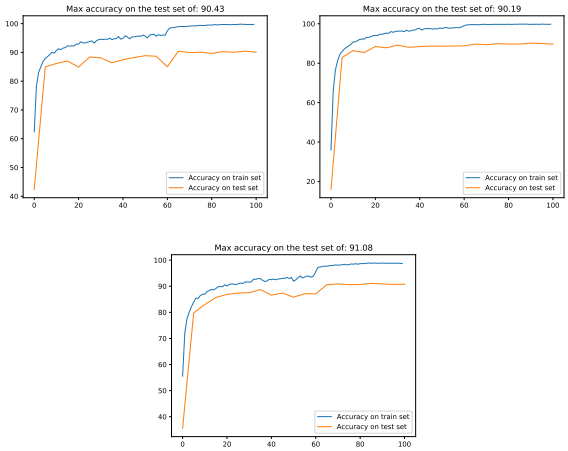

Figure 9. Accuracy on training and testing dataset for 100 epochs using different optimizers and a transformer dimension $d = 32$ Top left: Adam, Top right: SGD, Bottom: MadGrad

We can see that there is a significant difference between the behavior of the optimizers. Adam achieves better overall score than SGD but is less stable. MadGrad [4] seems to combine the better of the 2 worlds. It was the first time I tested it and it seems to reach its promises. Nevertheless, I would argue that I should have reproduced the results multiple time and averaged them for the results to be more statistically significant (I have not done it for time reasons).

4.6. Impact of the Transformer Dimension

The final study that I present is the impact of the Transformer Dimension d that has been mentioned multiple times during the report. The accuracies throughout the training are presented in figure 10. 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). It is at the cost of computational power, since it significantly increases the time needed to train the network. I tried to test a higher value of 128 but failed to finish the training at each of my trials (more than 6 hours to train).

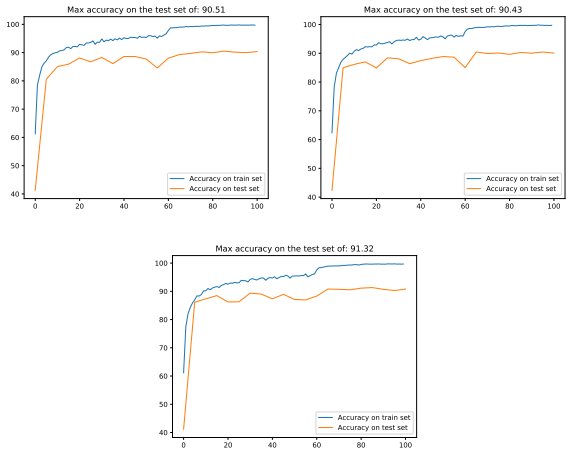

Figure 10. Accuracy on training and testing dataset for 100 epochs using Adam optimizer and varying transformer dimensions $d = 1, 32, 64$

4.7. Miscelaneous

I first wanted to mention the fact that I used Google Colab for the whole project in order to be able to use a GPU. It presents some limitations. First, it is difficult to upload large datasets: I have been limited by the capacity of Drive and by the download speed when using WGET. The ModelNet40 was relatively easy to use since a pre-processed .zip file already exists, but it would have been trickier for different datasets like ShapeNet (I tried to download it from the official website but it weighted at least 25 Go so I would not have been to use it in Colab). Besides, it is very unpractical when trying to train large architectures. I had to make sure that the window would stay active for more than 4 hours for some trainings (it can be done using a trick found in https://huggingface.co/blog/fine-tune-wav2vec2-english#training). Bigger training time can not even be achieved due to the daily GPU limitations. Consequently, it is hard to optimize well the hyperparameters, and very difficult to reproduce the training procedures used in this type of papers. I believe that one part of the gap between my results and the one of [24] can be explained from that.

I also wanted to share that after having tried to reproduce the results of the authors for a long time, I understand how important it is to provide code with a paper, as a proof of reproducibility. I also found that some details could have been added in the paper or in its appendix. For instance, the paper argues that vector attention outperforms scalar attention (see Section 4.4 of [24]), but they never give the dimension of the attention weights, although the study in 4.6 showed this parameter has crucial importance.

5. Conclusion and Further Work

The paper presented by Zhao et al. presents an adaptation of Transformers and self-attention to 3D point cloud processing. Overall, it is very well conducted and it seems possible that this method achieves SOTA on the tasks that are presented in it, i.e semantic scene segmentation, object part segmentation, and object classification.

In this project, I focused on trying to implement and reproduce the results on the ModelNet40 classification task. I obtained 91.32% accuracy on the test dataset at most, which are far from 93.7% of the authors' implementation. However, 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 and theirs.

Furthermore, I carried out several studies that did not appear in the original paper to show the importance of some parameters such as the optimizer and the vector-attention dimension. I also improved my first implementation by using Conv2d instead of linear transformations for the Transition Down layer.

As a conclusion, further work would consist in comparing this paper to [7] and [6]. These papers also introduce Transformers for Point Cloud processing but do it differently, and do not seem to achieve as good results as [24]. It could be very interesting to implement the other methods and to understand the difference between the 3 of them.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015. 2

[2] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 1, 2

[3] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019. 2

[4] Aaron Defazio and Samy Jelassi. Adaptivity without compromise: A momentumized, adaptive, dual averaged gradient method for stochastic optimization, 2021. 6

[5] A. Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, M. Dehghani, Matthias Minderer, Georg Heigold, S. Gelly, Jakob Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, 2020. 1, 2

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

[7] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R. Martin, and Shi-Min Hu. Pct: Point cloud transformer, 2021. 7

[8] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 1

[11] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on χ-transformed points. In *NIPS*, 2018. 2

[12] Niki J. Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *ICML*, 2018. 1, 2

[13] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 1, 2, 3, 4

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

[15] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015. 1, 2

[16] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 2

[17] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention, 2021. 2

[18] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers, 2021. 1, 2

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 2017. 1, 2

[20] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *TOG*, 2019. 1

[21] 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

[22] Xu Yan. Pointnet/pointnet++ pytorch. *https://github.com/yanx27/Pointnet_Pointnet2_pytorch*, 2019. 4, 5

[23] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. *CVPR*, 2020. 1, 2, 3

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

[25] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018. 1, 2