Machine Learning for 3D Geometry
Multi-view Vision Transformer for 3D Object Recognition

Boris Burkalo, 03763993
ge89rah@mytum.de

Ludwig Gräf, 03700273
ge56ceh@mytum.de

Alexandre Lutt, 03764286
ge89baf@mytum.de

Mohamed Said Derbel, 03699062
derbel@in.tum.de

Abstract

The introduction of Transformers in Natural Language Processing (NLP) is regarded as a breakthrough similar to the debut of Convolutional Neural Networks (CNNs) in the field of Computer Vision (CV). Recently, a trend is to leverage the advantages of Transformers in CV [3].

Classifying objects by visual features is a core task that is not only restricted to 2D but can also be done with 3D objects. In this paper the work of Chen et al. *(2021) [1] is replicated to check on their results and give others the opportunity to leverage on a detailed description of hyperparameters and training procedures. Even if there is still a gap in performance, the results are close to those of the paper.*

1. Introduction

Transformers were originally introduced for sequence-to-sequence processing in NLP. In this domain they prove that the self-attention mechanism is capable of benefiting already well performing models. In general, it is assumed that this mechanism allows networks to process longer sequences of data [8] better than classical Recurrent Neural Networks (RNNs).
In their paper, Chen *et al*. (2021) demonstrate the applicability of Transformers in CV. On the ModelNet10 [9] dataset, the introduced model was capable of predicting the 10 classes with an accuracy of around 95%. In contrast to previous approaches that relied on Convolutional Neural Networks (CNNs), such as the Multi-View-CNN, they increased the performance by 5% [6].
It can be assumed that this performance increase is achieved by the self-attention mechanism of the Transformers as the overall architecture as well as the dataset are unchanged.

This project attempts to implement a Transformer network similar to Chen *et al*. With no code base available, the provided implementation would provide a foundation for others to draw upon. Furthermore, the provision of hyperparameters as well as optional training steps makes the implementation less complex for third parties.

2. Related Work

Using several images of a 3D object and process them by CNNs is proven to be a reasonable approach [6]. Thereby, a size-invariant operation is reducing the respective latent spaces to a fixed dimension. This allows further processing by a Multi Layer Perceptron (MLP) model for classification. The drawback is that the CNNs processes each image without contextual information. Following the introduction of Transformers [7] in the field of NLP, the potential of these is leveraged in various applications like CV. And similarly to adopting CNN for 3D object understanding after achieving State-of-the-Art performance with 2D images, Transformer-based models were lately introduced to solve various tasks including 3D objects. In [4], the authors choose an approach to combine the powerful and effective local feature learning capabilities of CNN with the global context modeling capabilities of Transformers, the authors introduce a unique hierarchical architecture called the 3D Convolution-Transformer Network (3DCTN) and show that it delivers outstanding classification performances on the ModelNet40 dataset.
For the task of shape completion, ShapeFormer [10] was introduced, a Transformer-based network that generates a set of completions conditioned on incomplete and noisy point clouds, which are then combined to construct a dense final object.

3. Implementation

3.1. Preprocessing

As the goal is to analyze 3D objects by 2D views, it is the first step to project synthetic 2D views of the 3D models. As stated in the paper, 12 is a reasonable number of views. To be more specific, there are 12 snapshots, each of them

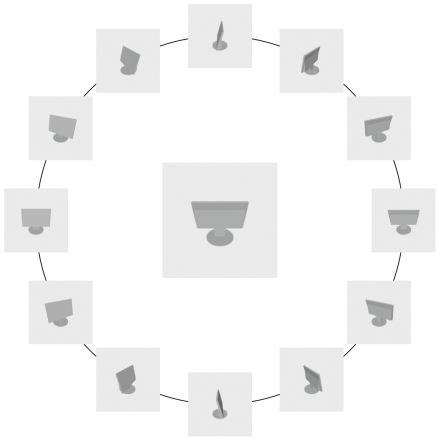

Figure 1. An example of snapshots taken around a 3D model

being a ($W \times H \times 3$) image, rotating around the model on the 30th parallel north, resulting in one snapshot taken every $\frac{2\pi}{12}$ radians (see Fig. 1).

After this step, each of the 12 images is cropped into $w \times h$ non-overlapping patches. Each of the patches is a ($p \times p \times 3$) image (with $w = \frac{W}{p}$, $h = \frac{H}{p}$). Then, every patches is unfolded into a $\mathbb{R}^{3p^2}$ vector. The i-th patch of the j-th view ($(i, j) \in [\![1, w \times h]\!] \times [\![1, 12]\!]$) is denoted by $\boldsymbol{p_i^j}$. In short, for each of our 3D objects, 12 2D views are computed, defined by $\frac{W}{p} \times \frac{H}{p}$ patches of dimension $3p^2$. Then, each $\boldsymbol{p_i^j}$ vector is multiplied by a $\mathbb{R}^{D \times 3p^2}$ weight matrix W_0 to get:

$$\boldsymbol{x_i^j} := \boldsymbol{W_0 p_i^j} \in \mathbb{R}^D \tag{1}$$

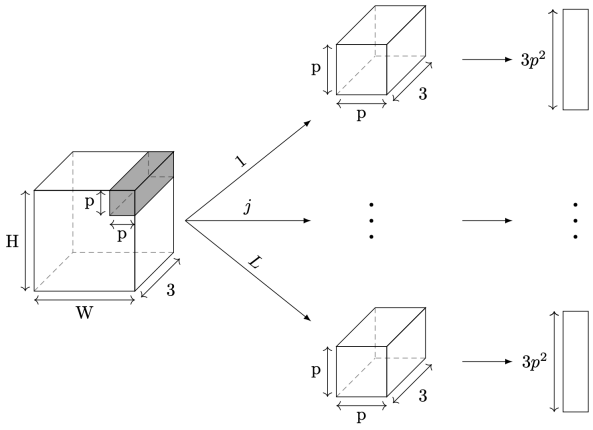

Figure 2. Preprocessing of the 2D views. L is equal to the number of views (here: 12)

3.2. Local and Global Transformers

3.2.1 Learned Position Embedding

The order of the images might be a relevant feature for the Transformer network to learn. The CNN-based approach for 3D object classification [6] relies on an order-invariant pooling operator to standardize the dimensionality of the different views. Thereby, the information on where certain features originate from is vanished. Transformers, that are used for NLP or CV, rely on learned positional embeddings that allow the network to estimate the position of inputs [2].

Thereby, it needs to be emphasized that all input sequences are processed simultaneously in contrast to the sequential processing in a classical RNN [5]. There exist several implementations that rely on mathematical properties to encode position information [2]. But, in general, the performance differences are negligible [7].

Once the position embedding are initiated, $\boldsymbol{p_i}$ (which only depends on the view i), they are added to the visual feature $\boldsymbol{p_i^j}$ to get $\boldsymbol{z_i^j} = \boldsymbol{p_i^j} + \boldsymbol{p_i}$. There is also a special token defined $\boldsymbol{z_0^j}$, which will help recover the view representation after the Transformer encoder. The concatenated matrix $\boldsymbol{Z} = [\boldsymbol{Z^1}, \boldsymbol{Z^2}, \dots \boldsymbol{Z^j}] \in \mathbb{R}^{D \times 12(wh+1)}$, where, for $j \in [\![1, 12]\!]$, $\boldsymbol{Z^j} = [\boldsymbol{z_0^j}, \boldsymbol{z_1^j}, \dots, \boldsymbol{z_{wh}^j}] \in \mathbb{R}^{D \times (wh+1)}$, will then be used as the input of the multi-view Transformer.

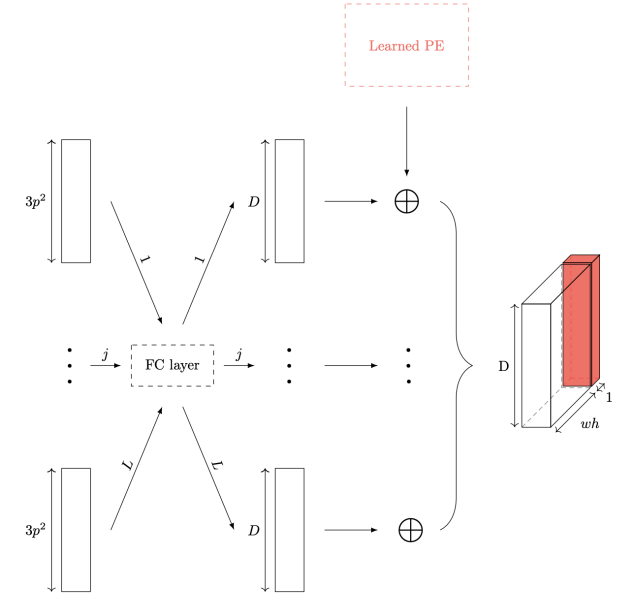

Figure 3. Using learned position embeddings

3.2.2 Local Transformer

The architecture starts by processing the different patch features from each view separately, which means that one can

consider that the input of each local Transformer block is a specific $\boldsymbol{Z^j}$, where $j \in [\![1, 12]\!]$. As there are S Transformer blocks with layers $\{\text{MSA}_s^l, \text{MLP}_s^l, \text{LN}_{s,1}^l, \text{LN}_{s,2}^l\}$ ($s \in [\![1, S]\!]$), this part of the architecture can be seen as a basic algorithmic procedure:

Algorithm 1: Local Transformer

Input: $\boldsymbol{Z} \in \mathbb{R}^{D \times 12(wh+1)}$
Output: $\boldsymbol{M} \in \mathbb{R}^{D \times 12*wh}$

for $j \leftarrow 1$ **to** 12 **do**
  for $s \leftarrow 1$ **to** S **do**
    $\boldsymbol{Z_s^j} \leftarrow \boldsymbol{Z_{s-1}^j} + \text{MSA}_s^l(\text{LN}_{s,1}^l(\boldsymbol{Z_{s-1}^j}))$
    $\boldsymbol{Z_s^j} \leftarrow \boldsymbol{Z_s^j} + \text{MLP}_s^l(\text{LN}_{s,2}^l(\boldsymbol{Z_s^j}))$
  end
end
$\boldsymbol{M} = [\boldsymbol{Z_S^1}, \boldsymbol{Z_S^2}, \ldots, \boldsymbol{Z_S^1 2}]$

3.2.3 Global Transformer

For the global Transformer, the previously concatenated matrix is used, which means that the different patch features from each view are processed together. As there are T Transformer blocks with layers $\{\text{MSA}_t^g, \text{MLP}_t^g, \text{LN}_{t,1}^g, \text{LN}_{t,2}^g\}$ ($t \in [\![1, T]\!]$), this part of the architecture can also be seen as a basic algorithmic procedure:

Algorithm 2: Global Transformer

Input: $\boldsymbol{M} \in \mathbb{R}^{D \times L(wh+1)}$
Output: $\boldsymbol{M_T} \in \mathbb{R}^{D \times L(wh+1)}$

for $t \leftarrow 1$ **to** T **do**
  $\boldsymbol{M_t} \leftarrow \boldsymbol{M_{t-1}} + \text{MSA}_t^g(\text{LN}_{t,1}^g(\boldsymbol{M_{t-1}}))$
  $\boldsymbol{M_t} \leftarrow \boldsymbol{M_t} + \text{MLP}_t^g(\text{LN}_{t,2}^g(\boldsymbol{M_t}))$
end

3.2.4 Final Classification

Subsequent to the global Transformer blocks, the features $\boldsymbol{M} = [\boldsymbol{m_0^1}, \ldots, \boldsymbol{m_{wh}^1}, \ldots, \boldsymbol{m_0^{12}}, \ldots, \boldsymbol{m_{wh}^{12}}] \in \mathbb{R}^{D \times 12(wh+1)}$ are used to predict the final representation $\boldsymbol{m}$ of the 3D object using the special tokens $(\boldsymbol{m_0^j})_{j \in [\![1,12]\!]}$.

$$\boldsymbol{m} = \frac{1}{12} \sum_{j=1}^{12} \boldsymbol{m}_0^j \tag{2}$$

Finally, $\boldsymbol{m}$ is processed by a fully-connected layer with a softmax activation function to get the probabilities $\boldsymbol{\mathcal{P}}$ that the 3D object belongs to one of the classes.

$$\boldsymbol{\mathcal{P}} = \boldsymbol{\sigma}(\boldsymbol{W}\boldsymbol{m} + \boldsymbol{b}) \tag{3}$$

Model	Test accuracy
Our model (w/o pre-train)	89.5%
Our model (w/ pre-train)	92.4%
Chen *et al*. (w/o pre-train)	92.5%
Chen *et al*. (w/ pre-train)	95.0%

Table 1. Significant performance gap between Chen *et al*. and our models.

4. Results

This section discusses the model results and gives insights into different training approaches to further improve the performance. As in [1], the model is trained and tested on the ModelNet10 [9] dataset that consists of 4,900 objects belonging to ten different classes.

In general, it is possible to achieve reasonable results that are close to the ones stated by Chen *et al*. Nevertheless, as seen in table 1 there is still a significant difference of more than two percent in accuracy on the test set. It is important to note that the authors did not specify in greater detail, how they trained their models. Especially interesting is that the pre-training seems to improve the model performance. This comes at the cost of an increased model size with a significantly increased compute time.

In general, the introduction of the Transformer is improving the overall model performance. It is important to note that this paper likely applied several modifications that should reduce overfitting and improve model performance, that were not explicitly stated in the official paper.

The introduction of dropout significantly reduced the gap between train- and validation-set. Thereby the overall performance is improved. Furthermore, the weights are regularized by a L2-regularizer. This further improved the performance slightly.

On a subsequent step, the dataset was augmented with the help of different transformations. It is important to note that these transforms were only applied to the train set and not on the validation-/test set. This step improved the performance significantly.

The difference between of the model performance on the train and validation set can be observed in figure 4, 5, 6. As seen in the figures, all the models achieve a reasonable performance close to and above 90%. In general, the best performing models use L2-regularization. The improvement in performance can be seen in figure 4 and 5. Interestingly, the models with the regularization at place performed better on the validation set than on the train set. This can be explained by the use of Dropout, as there is more capacity available in test time than on train time.

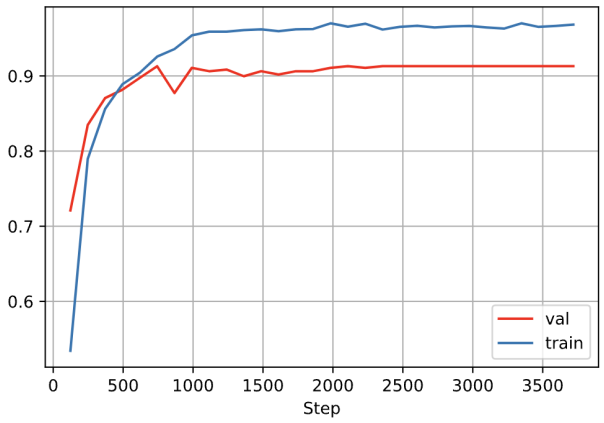

Figure 4. Validation and training accuracies, without regularization

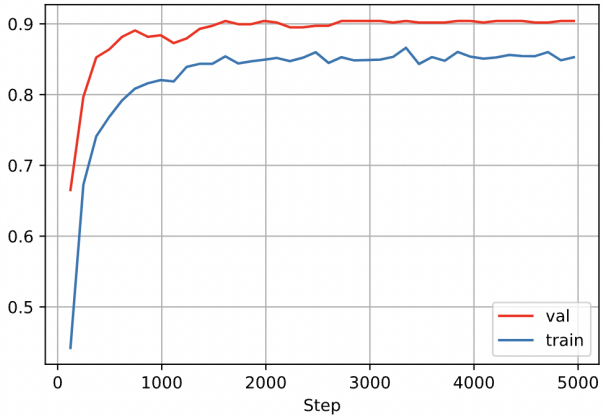

Figure 5. Validation and training accuracy, with pre-trained backbone

4.1. Hyperparameters

As stated in the section 3.1 section, the decision was made to employ the 12 views system. This is due to the findings of Chen *et al*., where they demonstrated that for non-restrictive computing environments, 12 projected views of the 3D models yielded the best results. In this paper, the goal is to provide a code basis that recreates the cited papers' proposed model and results, for which they proposed different versions. The implementation in this paper, follows the 8-4 tiny model. Specifically, a model that comprises 8 local blocks and 4 global blocks where each block has 3 attention heads. For the embedding space, the authors propose a 192 dimensional space. Initially, a non-regulated model setting was assumed. Thereby, an optimization for the expansion ratio is employed by the Transformers' MLP heads together with the rest of training-specific parameters like learning rate and batch size. Figure 4, shows the training curves using the optimized hyperparameters, one can notice that the model is in an overfitting regime. To reduce the memorization of the train set, a minor weight decay in the optimizer is introduced. Furthermore, dropout in the classification module and in the transformer heads is applied. Subsequently, this paper introduces input augmentation where all views of given 3D instance have a probability p of being augmented with a randomly picked transformation out of 6 differently parameterized CenterCrops and RandomRotations. The regularized models showed better generalization (see table 2). The results that can be seen in table 2 are all achieved by using the hyperparameters in table 3.

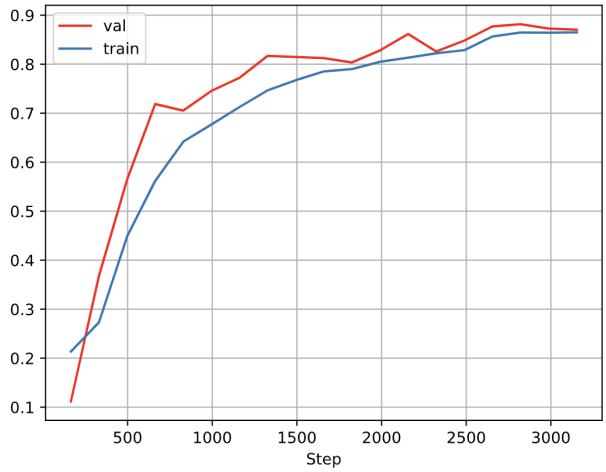

Figure 6. Validation and training accuracy, without pre-trained backbone

	w/o regularization	**w/ regularization**
w/o backbone	81.0%	89.0%
w/ backbone	89.0%	92.0%

Table 2. Accuracy for both models (with and without regularization)

5. Discussion

In general, the implementation covers all the proposed model's architecture introduced in [1] in a parameterized fashion that covers all its variants. Nevertheless, there is still a performance gap between the models trained in this paper and the ones stated by Chen *et al*. As stated in the previous section, there can be multiple reasons for not reaching the same performance levels due to specific training hyperparameter settings and to regularization levels. Other possibilities would be the use of more advanced training techniques specific to Transformers.

Appendix

parameter	value
num views	12
num patches	16
patch wh	4096
local blocks	8
global blocks	4
attention heads	3
expansion ratio	2
weight decay	7×10^{-6}
classifier dropout	0.2
transformer dropout	0.3
augmentation rate	0.3
learning rate	8.18×10^{-5}
batch size	32

Table 3. Final value for each hyperparameter

References

[1] Shuo Chen, Tan Yu, and Ping Li. Mvt: Multi-view vision transformer for 3d object recognition. *arXiv preprint arXiv:2110.13083*, 2021. 1, 3, 4

[2] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017. 2

[3] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)*, 2021. 1

[4] Dening Lu, Qian Xie, Linlin Xu, and Jonathan Li. 3dctn: 3d convolution-transformer network for point cloud classification. *arXiv preprint arXiv:2203.00828*, 2022. 1

[5] Abdelrahman Mohamed, Dmytro Okhonko, and Luke Zettlemoyer. Transformers with convolutional context for asr. *arXiv preprint arXiv:1904.11660*, 2019. 2

[6] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015. 1, 2

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 2

[8] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020. 1

[9] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 1, 3

[10] Xingguang Yan, Liqiang Lin, Niloy J Mitra, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Shapeformer: Transformer-based shape completion via sparse representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6239–6249, 2022. 1