International Journal of User-System Interaction 17 (1) 2024, 25-36

Enhancing object classification accuracy by incorporating tangent data in the model architecture

Dragos-Bogdan Tudor, Elena Pelican

Ovidius University of Constanta

Mamaia Blv 124, Constanta 900527, Romania
E-mail: epelican@365.univ-ovidius.ro

Abstract. This paper presents a method that aims to improve the recognition rate of PointMLP (Xu et al., 2022) for the classification task with minimal impact on training and inference speed by extracting and incorporating more information from the 3D objects used during the training process. Firstly, when extracting the 3D point cloud from the 3D mesh, besides sampling 3D points across the surface of the 3D objects and applying other common preprocessings, our proposed algorithm also saves the normal of the triangle out of which the 3D point was sampled and calculates 2 orthogonal vectors that are tangent to the surface of the 3D model in that point. Afterwards, the resulting vectors are incorporated into the architecture of PointMLP through the addition of new layers to the calculation of the embedding vector, so that the information about the tangents and normals is incorporated into the model architecture. This effectively provides the model with more context about the surface of the original 3D object, slightly improving the accuracy of the model with close to 0 impact on training/inference speed.

Keywords: 3D objects classification, PointMLP framework, tangents/bi-tangents/normals.

DOI: 10.37789/ijusi.2024.17.1.2

Introduction

Classification of 3D objects is crucial in many areas and across many industries and domains, like automotives (for self-driving vehicles and various systems for road safety), aeronautics (for many types of sensors and flight guidance equipment), military (smart radars, guided missiles), video games production (enabling some accessibility options), etc. One of the most important metrics for the classifiers is accuracy, defined as the number of correct predictions reported to the number of total predictions, as incorrect predictions could confuse the systems that use the data from the classifier and

lead to incorrect behaviors.

In computer graphics, 3D models are usually represented as unordered sets of R3 floating point vectors, grouped by their indices into polygons (commonly triangles or quads) that constitute the surface of an object. Whilst this structure is useful for interpolating fragments for rendering the model, it is not ideal for object classification. Classical approaches used on 2D images cannot be used directly, as the data in this structure is unordered and irregular. For this reason, the 3D models are usually transformed into structures that are easier to process and learn from during the training process (Section 2). One such data structure is the point cloud, which only keeps sets of R3 vectors that represent 3D points located on the surface of a 3D object, eliminating any connection between the vertices. Whilst this representation is popular for capturing real-world objects and environments (as in Zeid at al. (2023)) and is the representation provided by most capture devices (Wang et al. (2019)), this structure discards a lot of the information provided by an artificial 3D model (modeled by an artist, e.g.). This paper aims to propose an intuitive solution to extract more information about the surface of an artificial 3D object when converting it to a point cloud, with respect to the properties that a point cloud has (unordered, e.g.) by integrating additional input vectors into the point cloud: the normal vector, a vector perpendicular on the surface of the 3D object on the parent polygon of the extracted vertex, and some tangent vectors (vectors perpendicular to the normal vector). The necessity to extract more information from the 3D object is given by the lack of annotated training data available for 3D objects (Xue et al. (2022), Zeid at al. (2019), Qi et al. (2023)) as learning more features from the 3D objects could decrease the amount of necessary training data.

In mathematics, a tangent is defined as a vector that touches a curve or surface at a specific point in space. For a 3D object, the number of tangents to the surface is infinite, and we could theoretically choose any arbitrary vector that is tangent to the surface of the object in the chosen point. However, it is intuitive that if the tangent vectors are correlated with the tangents of their neighbors (from a Euclidean distance perspective), results might have higher accuracy. We have followed through with this intuition and extracted the vectors in such a way that they are aligned with their UV projections, similarly to how this process is done for game engines, where two orthogonal tangent vectors are extracted from the model in such a way that that they point in the direction of increasing texture coordinate U, respectively V. In 3D

graphics, this information is used for lighting calculations for improving the aspect of 3D objects during the rendering process by faking the lighting of bumps and dents.

As datasets of 3D objects available as of now do not include UV data, the projection vectors must be generated. In order to create the UV projection for a certain model, the model is placed in a 3D box with an edge of 2 units (after the model is centered and the vertices are normalized), and it is projected on the closest of the 8 sides of the cube. The projections are 2D vectors that can be used as placeholders for the UV vectors, a common method used in 3D graphics known as cube projection. There are other methods that can be used to generate the UV vectors, but this one was chosen because it generates vectors quickly and efficiently, which is important because when applying a more complex method to an entire data set, the time to compute the vectors quickly degenerates: the time required to calculate a cube projection for an object with about 100.000 vertices takes less than a second on most modern hardware, whilst more complex methods take a few minutes for the same object, as per our tests (ex: Blender's "Smart UV Project").

This article describes the methods used for generating the specified vectors for each 3D point in the sampled cloud and effectively incorporate this additional data in the architecture of the PointMLP classifier. This classifier was chosen specifically because it is the most recent classifier that uses the point cloud approach. However, this additional data could theoretically be incorporated in most other point-cloud based classifier architectures.

Related Works

Recent research on 3D object classification usually follows one of these approaches:

- conversion of the 3D object into voxel grids, followed by volumetric or convolutional approaches (as depicted in Maturana & Scherer (2015), Liu et al. (2021), Li et al. (2021), Zhang et al. (2021), Wu et al. (2015)). Being the first types of 3D object classifiers, some of those are simply generalizations over 2D image classifiers that extend well-known image classifier architectures into the 3rd dimension. Those methods are less efficient, as their memory requirements and computational power complexity are cubic, which limits the resolution of the grid even on

modern hardware. Because of the low resolution, the accuracy of those solutions is usually lower than other approaches.

- rasterization of the 3D objects, followed by conventional methods used in 2D image classification (as in Tatarchenko et al. (2018), Su et al. (2015), Gupta et al. (2014), Li et al. (2016), McCormac et al. (2016), Qi et al. (2017)). Articles that use this method start by drawing/rendering or otherwise calculating some form of projection of the object onto a plane, either from one or from multiple angles with various rendering techniques. Those images are stacked and are processed through various methods. Although those methods usually yield better results than the previous method, they are still limited because of the computationally expensive process of rendering images and the high volume of data that is costly to process.
- unified learning from multiple data representations (multi-modal networks). Those methods merge various types of data (like 2D images and 3D objects), which enables the model to learn from additional data. 3D object datasets are few and relatively small in size, which limits learning of most models. This approach, however, aims to solve this problem, and achieves better results than most other previous models. Depending on the specific model, the additional data can be extracted directly from the objects (e.g. rendering the 3D objects to generate images) (Yan et al. (2022)) or from external datasets with other types of data (Xue et al. (2022)).
- converting the 3D object to a point cloud and using the point cloud as input (as in Xu et al. (2022), Qi et al. (2017), Xu et al. (2021), Liu et al. (2019), Thomas et al. (2019), Zhao et al. (2020)). There are various methods to extract/sample a point cloud from the surface of a 3D object and even more methods of processing it for the task of object classification, many of those however being limited by the irregularity and sparseness of the data, as the classifiers does not have a lot of context about the surface of the object. This paper expands on a classic way of extracting this data, in an attempt to reduce this issue; but before describing our own methods, we will expand for more context on two well-known point cloud classifiers: PointNet, and PointMLP.

PointNet is the first multi-layer perceptron for 3D object classification, part segmentation and semantic segmentation ever created that trains directly on clouds of 3D points. This method was an important scientific breakthrough at its time, as it respected the permutation invariance of point clouds, moving away from classical approaches like convolutions that require highly regular input formats. This method has 3 key modules: max pooling as a symmetric operation to aggregate information from all the points, a local and global geometric combination structure, and two joint alignment networks that align both input points and point features (see Qi et al. (2017)).

PointMLP is a residual neural network (ResNet) used for classification and part segmentation, built on top of the PointNet idea. It is a neural network built with training and inference speed in mind that features a lightweight affine geometric module, which significantly improves recognition rate and with minimal impact on the performance.

The core of the architecture can be resumed using the following formula:

$$g_i = \theta_{pos}\left(A\left(\theta_{pre}\left(f_{i,j}\right), |j = 1, \dots, K\right)\right)$$

where θ_{pos} and θ_{pre} are residual multi-layer perceptron blocks, and ”A” is an aggregation function (namely, the max pooling operation is being used). The θ_{pre} block is a shared block meant to learn shared local weights, and the θ_{pos} block is designed to extract deep aggregated features. This equation is repeated for a number of stages, in order to form a profound (deep) neural network. The number of stages used is 4, and each of the θ_{pre} and θ_{pos} stages have 2 residual blocks.

The architecture was augmented with a geometric module that transforms the points via normalization while maintaining the original geometric properties, as it was found that deeper architectures are harmful for the accuracy and stability of the model (Xu et al., 2022).

Extracting Input Data for the Classifier

Our method directly consumes triangulated 3D objects, optionally UV-mapped (in the absence of an UV map, one will be automatically generated). Therefore, those objects are defined as data collections that carry the following information:

- sets of R^3 floating-point vectors that represent the vertices of an object in 3D space, normalized and centered
- sets of R^3 integer vectors that represent valid vertex indexes connected in triangles that define the surface of a 3D object
- sets of R^2 floating-point vectors that represent the UV coordinates of the vertices from the first set.

Starting from this structure, we describe how to extract additional information besides the point cloud and how to feed it to the PointMLP classifier.

In order to extract the point clouds from the 3D objects, we apply the most common algorithm for this task: randomly select a number of triangles (with a bias towards triangles with a greater surface) (Qi et al. (2017)), and then sample a random 3D location that sits on the surface of the triangle using Devroye's algorithm (Devroye (1986)). We are sampling 8192 points for each model using this method, followed by a reduction to 1024 of the most relevant vertices using the Furthest Point Sampling algorithm (see Eldar et al. (1997)).

The approach used to calculate the tangent vectors is similar to the methods used in 3D graphics for calculating the tangent and bitangent vectors for normal mapping, but with a few differences.

The classical process consists of calculating two of the triangle edge vectors (L_1, L_2) and rewriting them as combinations of the tangent vectors (B_1, B_2) with the components of the UV vectors, obtaining a system that can be solved using conventional methods (see Lengyel (2012)):

$$\begin{cases} L_1 = (u_1 - u_0)B_1 + (v_1 - v_0)B_2 \\ L_2 = (u_2 - u_0)B_1 + (v_2 - v_0)B_2 \end{cases}$$

Because we are trying to extract the tangent of a point that was randomly sampled on the surface of a 3D object, there are multiple approaches to extracting the tangent vectors that yield slightly different results. The simplest method to extract the tangent vectors is to calculate the tangents of the nearest vertex in the parent triangle, a trivial process using modern software. However, the vectors obtained using this method are slightly inaccurate, which might affect accuracy.

This is because in a smooth surface the tangents of the vertices are

interpolated for each triangle fragment (Lengyel (2019)). To obtain more accurate tangent vectors, we could calculate them by interpolating the tangents of the parent triangle; this method might yield better results precision-wise, but comes at a cost: we have to calculate 3 times more tangents for each triangle, which might affect the computation speed.

The method that was found to work best is to calculate the UV projection of the newly-sampled point using the same weights that were used to generate the randomly sampled 3D point. This provides the full 3D information needed to form a new triangle by combining the extracted point with 2 points from the original parent triangle. Using this triangle, we can now directly calculate the tangent vectors.

Incorporating the Extracted Data in the PointMLP Model

The input data received in the forward propagation call is processed into an embedding vector, which is then processed by the model. Naturally, the best method to incorporate the additional extracted data into the model is to modify the embedding vector. PointMLP uses for the calculation of the embedding vector a simple 1D convolution on the point cloud that summarizes an object's point cloud to just a few numbers (a R^{64} vector, or R^{32} for PointMLPElite), followed by a batch normalization and an activation (commonly ReLU). This embedding is mainly used as a dimension reduction method, as simply executing the forward pass on the entire point cloud data would be expensive to compute even on modern hardware.

Initially, the input size of this operation was intuitively modified to allow processing the additionally calculated data. However, this intuitive approach proved to decrease the accuracy of the model, either because the size of the embedding vector is insufficient to accurately summarize the locations of the 3D points alongside with the other 3 vectors, or because the 3D position of the point cloud vertices should weigh more than the additional vectors. The intuition that the latter is more likely the culprit of the decreased accuracy was followed, and the location vectors of the points were separated from the others and processed independently.

This results in two independent embedding vectors, that can be arbitrarily scaled to prioritize the embedding obtained from the location of the location vectors. Although fixed values could have been used to scale the vectors, a

better result can be obtained by using two independent learnable parameters instead. The two now scaled vectors could theoretically be kept and processed separately throughout the model and combined later, but this approach would significantly increase the execution time for training and inference, defeating the purpose of this classifier, as PointMLP (and especially its elite version) was created with a lot of emphasis on execution speed. For this reason, the two vectors are combined immediately after their creation, through a 1D convolution operation.

RESULTS

The modified PointMLP was only tested for the classification task. Incorporating the tangents and the normals using the described methods resulted in a slight improvement in the accuracy of the model for the PointMLP classifier, using the recommended 80% training, 20% testing split. The training/inference speed is negatively affected by the additional operation, but not significantly. The results are depicted in Tables 1, 2, and 3, respectively, while Figure 1 is the confusion matrix for our classification problem.

Table 1: PointMLPElite statistical results (average after training 3 times with different seeds)

	w. modifications	w/o. modifications
Training time (1 epoch)	32.916 s	32.751
Inference speed	0.398	0.369
Accuracy	92.52%	91.98%

Table 2: PointMLP statistical results: w. modifications w/o. modifications

Training time (1 epoch)	411.125 s	395.979 s
Inference speed	4.4692	4.660 s
Accuracy	93.15%	91.97%

Table 3. Classification report for the modified PointMLP (non-elite) classifier

Object	Precision	Recall	F1-score	Support
airplane	1.00	1.00	1.00	100
bathtub	0.98	0.90	0.94	50
bed	1.00	0.99	0.99	100
bench	0.85	0.85	0.85	20
bookshelf	0.95	0.98	0.97	100
bottle	0.98	0.97	0.97	100
bowl	0.74	1.00	0.85	20
car	1.00	1.00	1.00	100
chair	0.98	1.00	0.99	100
cone	0.95	0.90	0.92	20
cup	0.72	0.65	0.68	20
curtain	1.00	0.85	0.92	20
desk	0.93	0.80	0.86	86
door	0.95	1.00	0.98	20
dresser	0.84	0.92	0.88	86
flower pot	0.29	0.40	0.33	20
glass box	0.97	0.96	0.96	100
guitar	1.00	1.00	1.00	100
keyboard	0.95	1.00	0.98	20
lamp	0.85	0.85	0.85	20
laptop	1.00	1.00	1.00	20
mantel	1.00	0.98	0.99	100
monitor	0.98	1.00	0.99	100
night stand	0.89	0.77	0.82	86
person	1.00	0.95	0.97	20
piano	0.98	0.95	0.96	100
plant	0.90	0.87	0.88	100
radio	0.77	0.85	0.81	20
range hood	1.00	0.96	0.98	100
sink	0.88	0.75	0.81	20
sofa	0.98	1.00	0.99	100
stairs	0.95	0.95	0.95	20
stool	1.00	0.85	0.92	20
table	0.82	0.95	0.88	100
tent	0.76	0.95	0.84	20
toilet	0.98	1.00	0.99	100
tv stand	0.87	0.91	0.89	100
vase	0.85	0.82	0.83	100
wardrobe	0.88	0.75	0.81	20
xbox	0.94	0.85	0.89	20
accuracy	**0.93**	**0.90**	**0.90**	**2468**
macro avg	**0.91**	**0.90**	**0.93**	**2468**
weighted avg	**0.94**	**0.93**	**0.93**	**2468**

These results were obtained after training for 300 epochs, as recommended in the original article.

The GPU used for training is Nvidia Geforce RTX 4070Ti.

Both the training and inference scripts were modified so that the model can train using mixed precision (so that FP16 tensor cores found on the GPU can be used). The model was evaluated on the ModelNet40 shape classification benchmark data set (as in Wu et al. (2014)).

The accuracy results shown in this table reflect the results obtained during own testing, not the statistics from the original paper, where the accuracy is a bit higher. The likely culprit for this slight inconsistency is the version of the used dataset or the parameters and procedures used for extracting the point clouds from the said dataset. There are infinitely many valid point clouds representing the same 3D shape (Yan et al., 2022), and some might be better for training a certain architecture than others.

CONCLUSIONS

This paper showed the importance of extracting as much context as possible from the objects that we wish to classify, both during inference and training. This is important because training data for 3D object recognition is scarce; there are very few labeled datasets for 3D objects, and those that exist are small in size when compared to datasets used for 2D image recognition or other tasks.

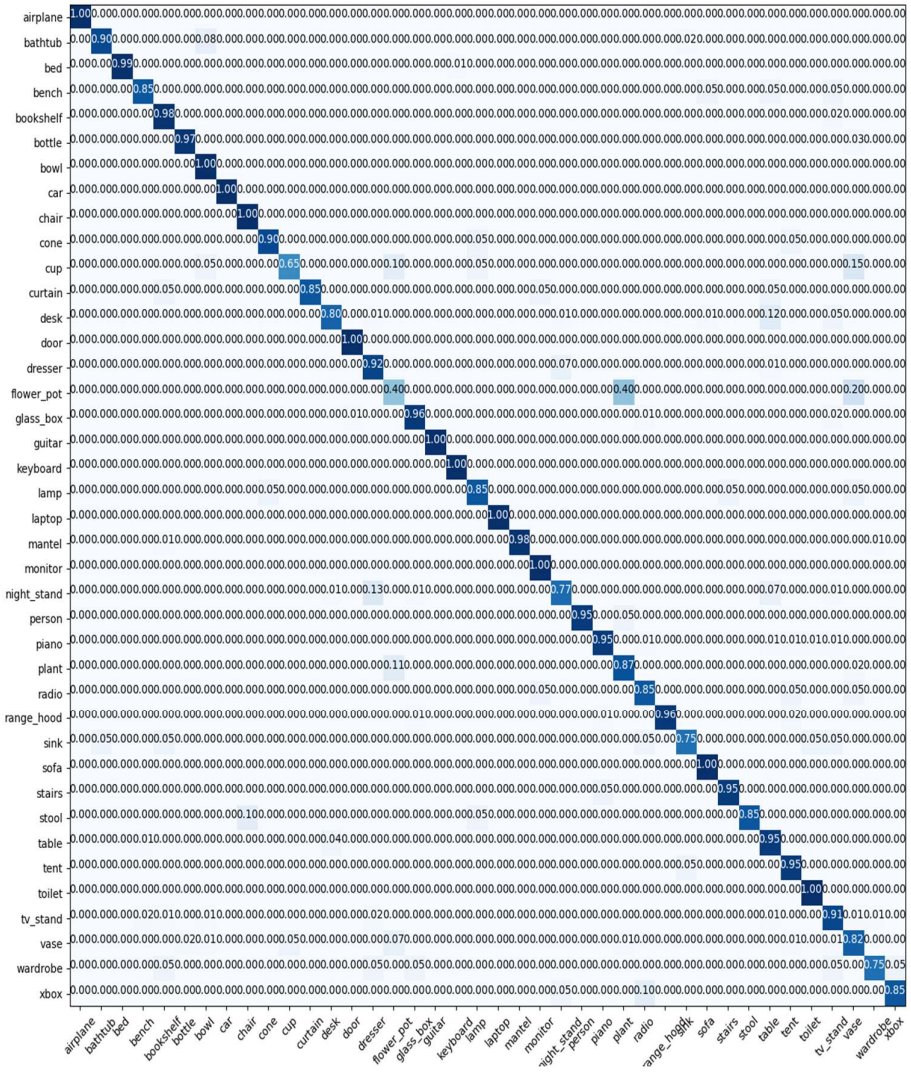

Figure 1: Confusion matrix for the PointMLP (non-elite)

References

Devroye, L. *Non-Uniform Random Variate Generation*, Springer, 1986, 569–570.

Eldar, Y., Lindenbaum, M., Porat, M., and Zeevi, Y.Y. (1997). The farthest point strategy

for progressive image sampling, *IEEE Transactions on Image Processing*, 6(9):1305–1315.

Gupta, S., Arbelaez, P., Girshick, R., and Malik, J. (2015) Indoor scene understanding with rgb-d images: Bottom-up segmentation, object detection and semantic segmentation. *International Journal of Computer Vision*, *112(2),* 133–149.

Lengyel, E. (2012) *Mathematics for 3D Game Programming and Computer Graphics*, Cengage Learning, 180–183.

Lengyel, E. (2019). *Foundations of Game Engine Development: rendering*, vol. 2. Terathon Software LLC, page 38.

Li, Z., Gan, Y., Liang, X., Yu, Y., Cheng, H., and Lin, L. (2016) RGB-D scene labeling with long short-term memorized fusion model. *arXiv preprint arXiv:1604.05000v3*

Li, Z., Wang, F., and Wang, N. (2021) Lidar R-CNN: an efficient and universal 3D object detector. *arXiv preprint arXiv:2103.15297v1*

Liu, Y., Fan, B., Xiang, S., and Pan, C. (2019). Relation- shape convolutional neural network for point cloud analysis, *Proceedings of CVPR Conference*, 8895-8904.

Liu, Z., Zhang, Z., Cao, Y., Hu,H., and Tong, X. (2021) Group-free 3D object detection via transformers. *arXiv preprint arXiv:2104.00678v2*

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

McCormac, J., Handa, A., Davison, A.J., and Leutenegger, S. (2016) Semanticfusion: Dense 3D semantic mapping with convolutional neural networks. *arXiv preprint arXiv:1609.05130v2*

Qi, C.R., Su, H., Mo, K., and Guibas, L.J. (2017). Pointnet: Deep learning on point sets for 3D classification and segmentation, *Proceedings of CVPR Conference*, 652-660.

.