Final-year internship report

ACCOU Martin

TODO

Version 0.1 of
September 18, 2024

Contents

List of Figures

Chapter 1

Introduction

1.1 D3S, a leader in 3D CAD Analytics

D3S, which stands for Data Science Softwares & Services, specializes in delivering customized software solutions utilizing AI technologies. The company comprises a team of Data Scientists and Full Stack Developers with deep expertise in 3D CAD (Computer-Aided Design) Analytics and Natural Language Processing (NLP).

Its state-of-the-art technologies are built on open-source libraries and supported by internal R&D, enabling efficient data extraction through computer vision, Optical Character Recognition (OCR), and NLP. D3S also excels in deep learning applications such as 3D morpho analysis, metrics comparison, BoM (Bill of Materials) analytics, and time series processing.

The company's solutions are designed to provide scalable, adaptable, and secure business value for industries like aerospace and automotive.

I have been working with the 3D CAD Analytics team, where my focus was on developing a 3D similarity model designed to compare CAD models solely based on their geometric properties.

Figure 1.1: D3S, Data Science Softwares & Services

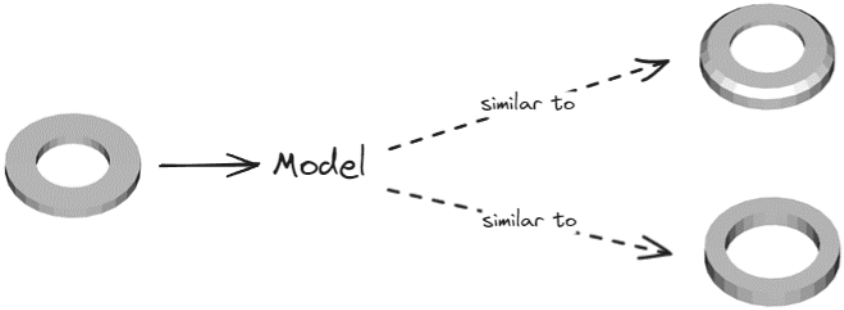


Figure 1.2: Purpose of our similarity model

1.2 3D similarity model

3D model designers spend a significant amount of time searching for relevant information during the product design process, even though much of their work could be done by modifying existing Computer-Aided Design (CAD) models. As a result, the retrieval and reuse of CAD models are crucial in CAD model management. However, large CAD model repositories often require extensive categorization or organization of engineering data, making design reuse challenging. Traditionally, the classification and retrieval of 3D CAD models involved a manual process of labeling, which is time-consuming, prone to errors, and inefficient. This issue becomes even more pronounced when models are generated in product development, as inconsistent labeling and tagging across different systems lead to the complex task of data harmonization. Additionally, the inherent complexity of 3D CAD model definitions makes it difficult to apply rigid, general classification rules, as model features and parameters vary depending on their origin. Therefore, an automated approach to classification and retrieval is needed to address these challenges.

The goal is to automatically associate a given piece to similar other pieces, as depicted in Figure 1.2. This will make it possible to leverage the D3S dataset of industrial 3D models, in order to infer missing information, such as the name of a piece, its function, or its material.

In recent years, point cloud representations has become one of the research hotspots in the field of computer vision [19]. In our case, we can not directly train a powerful classifier, because of a lack of clean labeled data and of the variety of the possible 3D models. The most comprehensive dataset available consists of just 2,000 CAD models, with rather imprecise labeling. Examples of labels include coupling strap, shackle, and long beam. It is evident that there is a significant need for a more curated and accurate dataset in this area, which is really hard to obtain.

Figure 1.3: Pipeline for building the model

Given the recent success of self-supervised learning methods, more specifically contrastive learning [10, 18, 7], an innovative and promising approach has been proposed to tackle this problem.

The goal is to learn a representation of data such that similar instances are close together in the representation space, while dissimilar instances are far apart. To do so, a triplet loss, popularized by the FaceNet model [11], will be used. Since we lack labeled data, we can't generate triplets directly as in [11]. Instead, a 'Tinder-like' application has been developped and used by the whole company to build our labeled triplets database. To clarify, a labeled triplet consists of three 3D models: an anchor, a positive, and a negative model. The detailed framework of our triplet loss is outlined in section 3.2. In this setup, the anchor and positive models are similar to each other, whereas the negative model is distinct from both.

To summarize, the pipeline comprises the two main following steps:

1. **Triplets collection**: Offline unlabeled triplets are generated. Triplets are then labeled by the users of the app and stored in the database.

2. **Model training**: An encoding model is trained on the labeled triplets. The model is then used to compute the similarity between two 3D models.

Chapter 2

Related work

2.1 3D Understanding

3D data can be presented in various formats, and the selection of the format is critical and depends on the specific needs of the application. A CAD model is a 3D representation of a physical object. At a higher level, the Standard for the Exchange of Product model data (STEP format) is a widely adopted ISO standard for data exchange that can represent 3D objects in CAD and related information. In this format, a CAD model is defined by its topological components such as faces, edges, or vertices and the connections between them. At a lower level, the STL format is a file format that represents 3D objects as a collection of triangles (mesh). This format is commonly used in 3D printing and computer graphics. Additionally, the point cloud format consists of a set of points in a 3D space, with each point representing a single point on the surface of the object. This format can be easily derived from an STL file by sampling points on the mesh surface. Other formats worth mentioning include voxel and multi-view image formats [19].

The decision was made to explore both STL and point cloud formats. The choice of the STL format is driven by its widespread use in the industry and the ease of generating a point cloud from it. Notably, models based on the STEP format [8], while promising, were not considered because they limit the scope too much.

Following recent trends in the field, two main classes of models were investigated, graph-based models and transformer-based models.

Graph-based models

Graph neural networks (GNN) have been used recently in numerous applications [17, 6] The flexible nature of a graph permits its usage from data sets concerning large social networks to smaller networks that describe the chemical bonds of a molecule. Graph neural

networks use the correspondences between elements instead of focusing on individual elements. These correspondences help create neighborhoods and local regions, which greatly enhance the predictive accuracy of the resulting features.

Given an input mesh, a natural candidate is a graph which nodes are the vertices of the mesh and where each vertex is connected to the vertices that share a face with it. This information is not available for the different GNNs that have been developed for 3D point cloud data [14, 15, 2]. These models are designed to work with point clouds. The main challenge is to define a graph structure that captures the local and global features of the point cloud. The most common approach is to define a graph where each point is a node and the edges are defined by the k-nearest neighbors of each point. The graph is then fed to a GNN to extract features from the point cloud.

Transformer-based models

Transformers [13] and self-attention models have revolutionized machine translation and natural language processing. They can equal or even surpass convolutional networks when applied to sequences and 2D images [3].

Self-attention is especially relevant in our context, as it naturally functions as a set operator, where positional information is treated as an attribute of elements within a set. Given that 3D point clouds consist of sets of points with positional attributes, the self-attention mechanism appears particularly well-suited to handling this type of data. Many recent works have explored the application of transformers to 3D data [20, 18, 7].

2.2 Contrastive learning

The goal of contrastive representation learning is to learn an embedding space in which similar sample pairs stay close to each other while dissimilar ones are far apart. Contrastive learning can be applied to both supervised and unsupervised settings. When working with unsupervised data, contrastive learning is one of the most powerful approaches in self-supervised learning [16].

Triplet loss is a widely used loss function in contrastive learning, made famous by the FaceNet model [11]. Numerous strategies have been suggested to enhance or refine the standard triplet loss [5, 4, 12]. Our approach incorporates some of these optimizations while leveraging our in-house labeling tool (TODO).

Chapter 3

Method

We propose a method for constructing a similarity model from an unlabeled dataset that contains a small proportion of approximately labeled instances. This method is applied on our use case of 3D CAD models, but it is actually generalizable to other domains.

Unlabeled triplets are generated from the entire dataset section 3.3. These triplets are then labeled using a dedicated application section 3.1. Finally, the labeled triplets are used to train a model based on triplet loss section 3.2.

3.1 Labeling application

Before delving into the details of the pipeline, it is essential to introduce the user interface that has been developed in order to build the labeled triplets database.

There are actually two main use cases for the application: labeling triplets (Figure 3.2) and comparing two iterations of a model (Figure 3.3).

- **Labeling triplets:** Users are presented with a triplet of 3D models, with the reference model (anchor) in the center. Their task is to swipe left or right to indicate whether the left or right model is more similar to the anchor model. A skip option is also available for ambiguous cases.

  Users can also change to a canonical view of the models to facilitate the comparison. This changes the perspective of a camera to a view aligned with the model's principal axes. This may or may not be useful depending on the model's geometry, as shown in Figure 3.1.

  Additional metadata is also provided to the users, such as the piece length. This information can help users make more informed decisions when the geometry does not suffice.

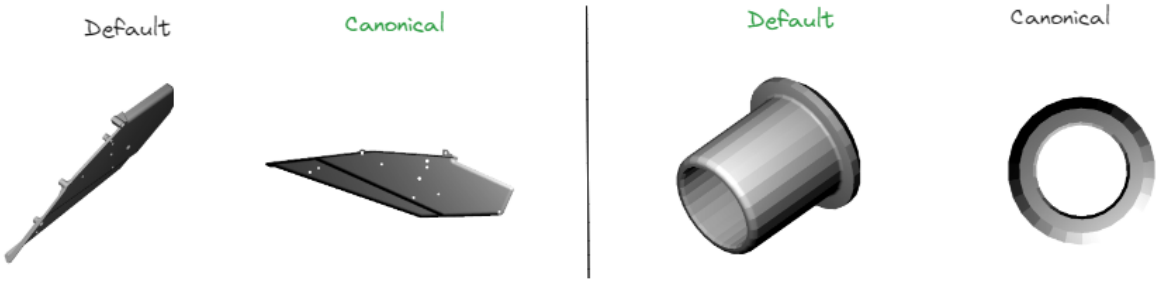


Figure 3.1: Two pieces where the appropriate view is different.

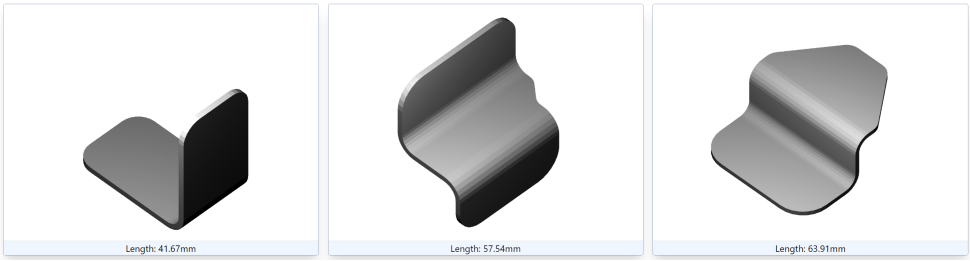


Figure 3.2: User interface for the labeling use case. Here, the piece on the right is more similar to the anchor piece in the center (curved edges).

- **Validation:** A similar interface is used, but now the propositions on the left and right are two nearest neighbors propositions of two different models. Users judge which model gives the best nearest neighbor proposition. This gives an easy way to compare two iterations of a model. This is especially useful to bulletproof the model's performance, ensuring robustness not only in quantitative metrics but also in qualitative assessments.

This application is designed to be intuitive and user-friendly, allowing users to quickly and efficiently label triplets.

Figure 3.3: User interface for the validation use case. Here, the proposition of the model on the left is more similar to the anchor piece in the center (two holes at the top).

Additional gamification elements were considered at one point, but they were deemed unnecessary as we were already achieving strong results with a small number of triplets.

3.2 Triplet loss training

The goal is to learn a representation of data such that similar instances are close together in the representation space, while dissimilar instances are far apart. We are going to use a triplet loss approach to train our encoder model. Once we will have our embeddings, the **cosine similarity** will be used to define a distance among the data points.

$$d(x, y) = 1 - \frac{x \cdot y}{\|x\|\|y\|}$$

The loss will be defined over triplets of embeddings:

- An anchor.
- A positive, of the same class as the anchor.
- A negative, of a different class than the anchor.

Given a batch size N, a distance function d, a margin $margin$ and a, p and n tensors representing anchor, positive and negative examples, respectively.

$\ell(a, p, n) = L = \{l_1, \ldots, l_N\}^\top, \quad l_i = \max\{d(a_i, p_i) - d(a_i, n_i) + \text{ margin }, 0\}$

Input tensors a, p and n are of shape (N, D), where D is the embedding dimension. The loss l_i is computed for each triplet in the batch, and the final loss is the mean of all individual losses.

Intuitively, you aim to group the data into separate clusters, ensuring that each class forms its own cluster. However, the precise distance between clusters isn't a concern, as long as they remain clearly distinct. Without limiting the loss, the model could improve by pushing an easy negative example very far away, while neglecting more challenging negative examples. By capping the reward beyond a certain distance, the model is encouraged to focus on the harder examples to continue improving.

Based on the definition of the loss, there are three categories of triplets:

- **Easy triplets:** Triplets which have a loss of 0, because $d(a, p) + margin < d(a, n)$.
- **Hard triplets:** Triplets where the negative is closer to the anchor than the positive, i.e. $d(a, n) < d(a, p)$.

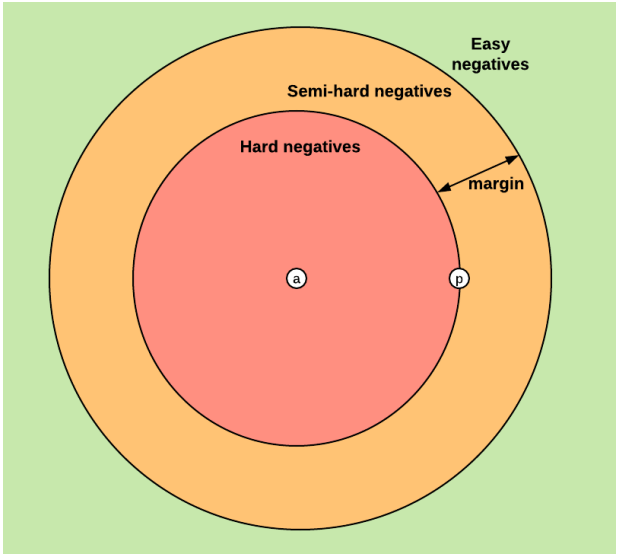


Figure 3.4: The three types of negatives, given an anchor and a positive.

- **Semi-hard triplets:** Triplets where the negative is not closer to the anchor than the positive, but which still have positive loss: $d(a,p) < d(a,n) < d(a,p) + margin$.

The balance of these triplet types is crucial for the model's performance, as we will see in TODO.

Before delving deeper into working on triplet loss, we had to first ensure that training a triplet loss model with 'fake triplets' generated from our labeled dataset yielded similar results to a classification model trained on the same dataset. To do so, a GNN based on the EdgeConv architecure [15] was used.

The performance of the classification model was evaluated against the triplet loss model using a nearest neighbor approach. This method assessed how often a piece's label matched that of its closest neighbor in the learned representation space. Both models achieved a 95% accuracy rate, suggesting that the triplet loss model successfully learned to create meaningful representations of the data. It is important to emphasize that at this point only triplets generated from the labels were used, and no human labeling was done. Triplets are generated following the same approach as in section .1, but ensuring that the anchor and the positive are indeed of the same class.

With just 1000 triplets, the model achieved a nearest neighbor accuracy similar to the classification model. However, optimal performance was observed when using approximately 10000 triplets, which resulted in a 95% accuracy rate.

3.3 Triplets generation

There is still the question of how to generate the triplets that are going to be labeled by the user of the app. Let's draw the properties of the triplets we want to generate, ordered by ease of implementation:

1. The triplets should be diverse enough to cover the entire dataset.
2. Almost every triplet should be 'labelable'. A triplet is 'labelable' if users do not have to skip it. This is a mainly a user experience concern.
3. A great proportion of triplets should be 'hard triplets' for the model that generated them, ie triplets where the negative is hard. This will force the next iteration of the model to learn from its misconceptions.

A simple yet effective way has been proposed to generate triplets. For the initial generation, a model trained on the small labeled dataset we had was used to compute embeddings for the entire dataset. This model was used to generate triplets based on heuristics which are detailed in section .1. Subsequently, the model was retrained on the labeled triplets, and the process was repeated iteratively.

With such a method, property 1 is straightforward to achieve, as a random selection of triplets will cover the entire dataset. Property 2 and 3 are in conflict, as a triplet that is easy to label is likely to be easy for the model as well. Considering the limits, selecting candidates that are equidistant from the anchor is intriguing, but tends to result in many 'skipped' triplets. Conversely, choosing candidates where the first is closer than the second will lead to fewer skipped triplets, but also a lower proportion of semi-hard and hard triplets.

It's why a proper balance has to be found, as explained in section .1.

Chapter 4

Experiments

Our final dataset for training the triplet loss model consisted of 20000 triplets, gradually accumulated through our labeling application. We employed consistent training configurations (section 4.1). The experiments were conducted using two different model architectures : GNN (section 4.2) and Transformer-based models (section 4.3).

4.1 Training setups

4.1.1 Data

The dataset used for training the models consists of 20000 triplets, each containing three 3D pieces: an anchor, a positive, and a negative model. Input data are meshes stored in stl files.

For the GNN model, vertices of the mesh are represented as nodes in a graph, while edges are defined based on the connectivity of the vertices. When two vertices share a face, an edge is created between them. For the transformer model, that take as input a point cloud, we convert the mesh into a point cloud by sampling points uniformly from the surface of the mesh. The number of sampled points is fixed to 10000, following [7].

Initially, the use of Principal Component Analysis (PCA) for training or inference, as done in [9], was considered but ultimately discarded. The idea of PCA is appealing because it allows working within a canonical reference frame. However, to ensure the model is as general as possible, data augmentation techniques were favored over PCA. One of the issues with PCA is demonstrated in Figure 4.1. Additionally, we prefer to let the model architecture handle these aspects directly.

Data augmentation was performed to enhance the model's robustness, especially considering rotations. Other augmentation techniques, such as symmetry and noise addition,

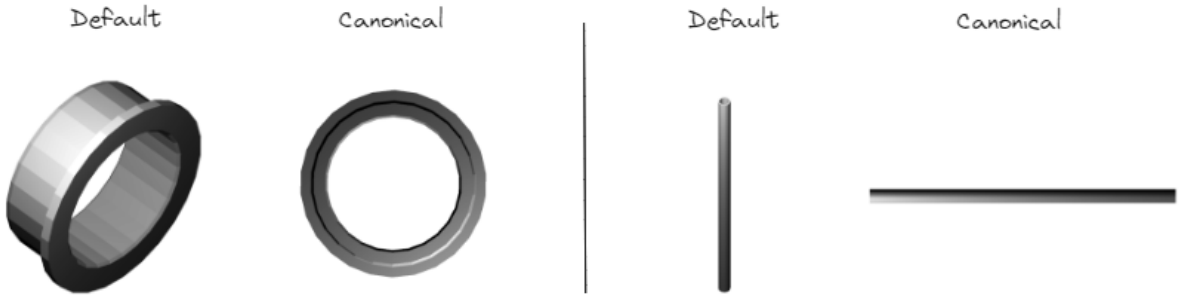


Figure 4.1: Illustration of the issue using PCA as a preprocessing step. Both pieces are tubular structures, shown in their default orientation and in the canonical reference frame. While the piece on the left aligns its first principal axis with the tube's axis, the piece on the right does not. This inconsistency can result in misalignment during training, particularly since the behavior varies depending on the specific piece.

were also explored but ultimately discarded. The following two techniques were retained:

1. **Normalization to unit sphere**: We standardize all the pieces to fit within a unit sphere. This enhances the model's ability to capture geometric features rather than scale. The length of the piece has been integrated into the architecture of the GNN model.

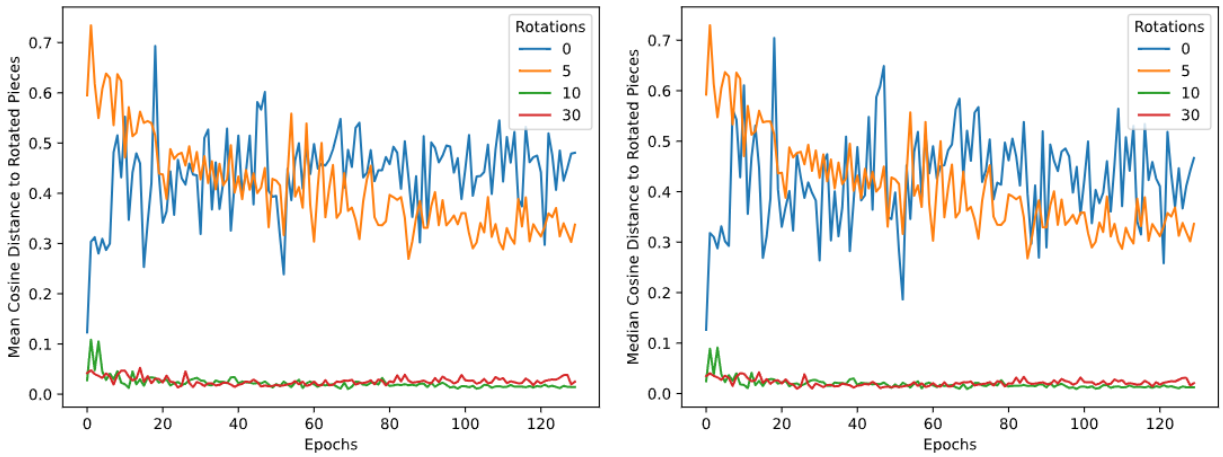
2. **Random rotations**: Each piece is randomly rotated around the three axes. This is crucial for ensuring that the model learns to recognize pieces regardless of their orientation. A special focus was placed on rotation invariance, as explained hereafter.

Rotation invariance is a critical aspect of our model, as it allows the model to recognize pieces regardless of their orientation. This is particularly important in our context since 3D model designers often work with similar pieces saved in different orientations. To achieve this, 3 metrics were defined to evaluate the model's performance in terms of rotation invariance:

- **Mean distance to rotated distribution**: This metric calculates the average distance between the model's prediction of a piece and the predictions of its n rotated versions.

- **Median distance to rotated distribution**: Similar to the mean distance, this metric computes the median distance between the model's prediction and the predictions of the n rotated versions.

- **Rotation matching accuracy**: The two previous metrics are absolute. To provide a more relative measure, we also compute a rotation matching accuracy. If we



(a) Mean distance to rotated distribution. (b) Median distance to rotated distribution.

Figure 4.2: Influence of the number of rotations on the rotations metrics.

fix a value of n, we can compute the proportion of nearest neighbors in the whole dataset that are also among the n rotated versions of the piece. n is set to 10 in our experiments, as for the two previous metrics.

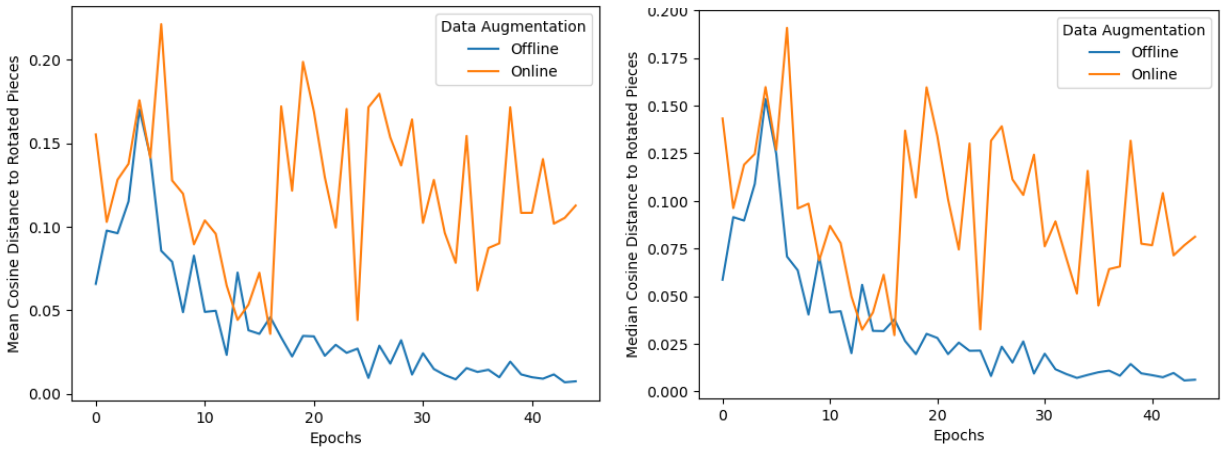
Initially, data augmentation was performed offline. This means that for each triplet, each piece was augmented by applying random rotations before the training process. We choose 10 rotations per piece, as it yields the best results as shown in Figure 4.2. Results are shown with our starting GNN EdgeConv model.

We also experienced online data augmentation, where random rotations are applied on-the-fly during training. Indeed, there is no clear recipe for the best approach [1] and we wanted to explore both options. The results of both approaches are compared in Figure 4.3. The offline augmentation approach yielded better results.

4.1.2 Triplets metrics

As is described in TODO, training a triplet loss model is notoriously difficult. To better monitor the training process, several metrics were defined:

- **Train easy triplets proportion**: This metric computes the proportion of easy triplets in the training set. An easy triplet results in a loss of 0.
- **Test easy triplets proportion**: This metric computes the proportion of easy triplets in the test set.
- **Training triplets order**: This metrics computes the proportion of triplets that are 'correctly' ordered. A triplet is correctly ordered if the distance between the anchor



(a) Mean distance to rotated distribution.

(b) Median distance to rotated distribution.

Figure 4.3: Influence of the type of data augmentation on the rotations metrics.

and positive is less than the distance between the anchor and negative. This is the same as an easy triplet proportion for a triplet loss with margin 0.

- **Test triplets order**: This metrics computes the proportion of triplets that are 'correctly' ordered in the test set.

Proportions of the training data were incorporated to analyze the model's behavior when converging to a loss of 0. The order of triplets was included to both assess the impact of the margin on triplet loss and to align with the fact that this order ultimately matters, as it's the one used by app users for labeling.

In addition to the rotation metrics, we also computed the accuracy of the model on the small labeled dataset by evaluating the proportion of pieces which nearest neighbor shares the same label. This metric is referred to as the **test accuracy**.

4.2 Graph-based models

4.3 Transformer-based models

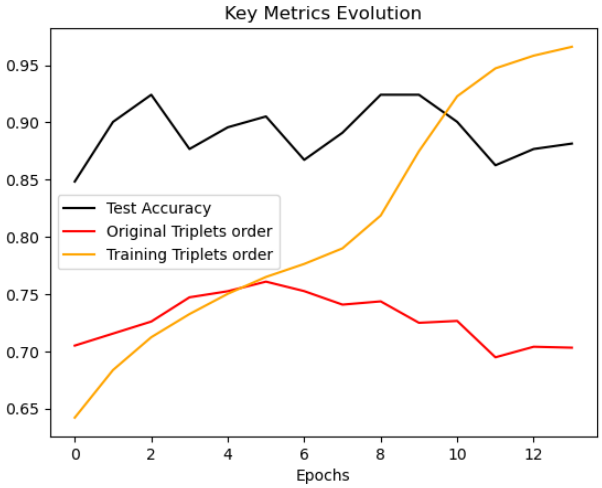


Figure 4.4: GNN model key metrics accuracies.

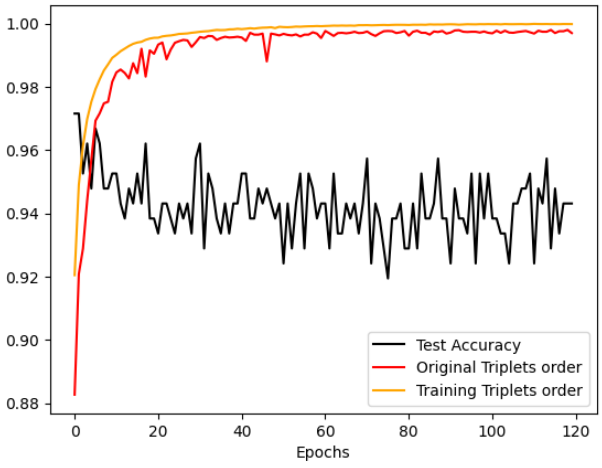
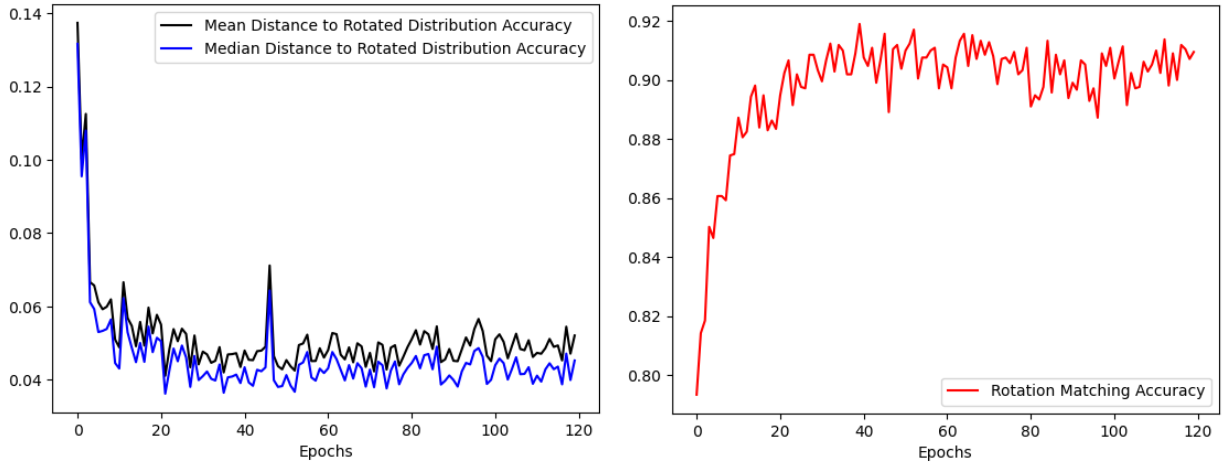


Figure 4.5: Fine-tuned model key metrics accuracies.



(a) Distances to rotated pieces accuracies.

(b) Rotation matching accuracy.

Figure 4.6: Fine-tuned model rotation accuracies.

Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis. Curabitur aliquet pellentesque diam. Integer quis metus vitae elit lobortis egestas. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi vel erat non mauris convallis vehicula. Nulla et sapien. Integer tortor tellus, aliquam faucibus, convallis id, congue eu, quam. Mauris ullamcorper felis vitae erat. Proin feugiat, augue non elementum posuere, metus purus iaculis lectus, et tristique ligula justo vitae magna. Aliquam convallis sollicitudin purus. Praesent aliquam, enim at fermentum mollis, ligula massa adipiscing nisl, ac euismod nibh nisl eu lectus. Fusce vulputate sem at sapien. Vivamus leo. Aliquam euismod libero eu enim. Nulla nec felis sed leo placerat imperdiet. Aenean suscipit nulla in justo. Suspendisse cursus rutrum augue. Nulla tincidunt tincidunt mi. Curabitur iaculis, lorem vel rhoncus faucibus, felis magna fermentum augue, et ultricies lacus lorem varius purus. Curabitur eu amet.

Chapter 5

Applications and perspectives

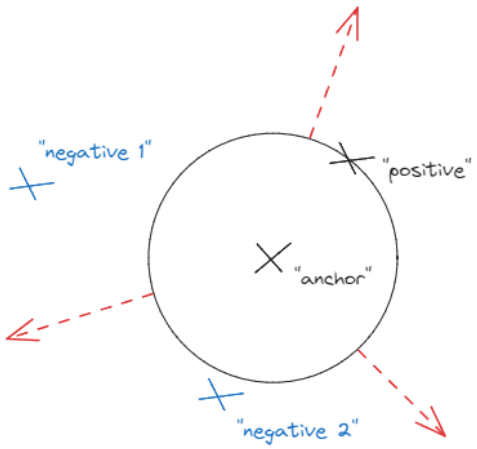


Figure 1: Triplet selection process. An "anchor" point is selected, and a "positive" point is found at a fixed target distance. A "negative" point is then found at a distance defined by the target distance plus a delta percentage. How far is the negative point matters a lot. If negative 1 is chosen, the triplet will likely be easy to label, but will not bring a lot of value to the model. If negative 2 is chosen, the triplet will likely be hard to label (and thus skipped), but will bring a lot of value to the model.

.1 Triplets generation heuristics

.1.1 Triplets selection

For each anchor data point, we fix a target distance, and a delta percentage. We can in this way find a first data point which distance to the anchor data point is close to the fixed target distance.

Then, we can find a second data point which distance to the anchor data point is close to the fixed target distance **plus** a percentage of this target distance, defined by delta. We have to be careful not to propose the same data point here (especially working with small values).

The proposed triplet is the concatenation of these three data points. In the following, for simplicity, we will call these 3 data points respectively the anchor, the positive and the negative, even if the proposed negative can actually be the real positive (hard triplet).

This selection process is shown in Figure 1.

We initially used target distances uniformly distributed between 0.001 and 0.05, and delta uniformly distributed between 0.1 and 0.5.

These hyperparameters must be tuned according to the dataset. Iterative visual inspections were conducted to ensure that the triplets generated were relevant according to

the properties defined in section 3.3.

.1.2 Triplets filtering

Once our triplets are selected, we have to filter some triplets that do not bring enough value, or that will be too hard for the user to label.

1. Concatenation of the triplets found for each pair of parameters can lead to duplicate triplets -> Duplicated triplets are discarded.
2. It turns out that triplets with the same anchor and positive points share really similar negative point -> Duplicated (anchor, positive) triplets are discarded.
3. Working with small target distances and deltas can lead to undesirable behaviours -> Triplets where the distance from the anchor to the positive 0 are discarded. Triplets where the distance from the anchor to the positive is greater than the distance from the anchor to the negative are discarded.
4. Sometimes, the positive and the negative data points turn out to be pretty similar -> Triplets where the relative distance from the positive to the negative is greater than a percentage of the distance from the anchor to the positive are discarded.

All these filters aim at reducing as much as possible the amount of triplets that won't be labeled by the user.

Bibliography

[1] Offline Data Augmentation. https://docs.nvidia.com/tao/tao-toolkit/text/data_services/augment.html.

[2] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPFNet: Global Context Aware Local Features for Robust 3D Point Matching, March 2018.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021.

[4] Weifeng Ge, Weilin Huang, Dengke Dong, and Matthew R. Scott. Deep Metric Learning with Hierarchical Triplet Loss. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, volume 11210, pages 272–288. Springer International Publishing, Cham, 2018.

[5] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In Defense of the Triplet Loss for Person Re-Identification, November 2017.

[6] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021.

[7] Minghua Liu, Ruoxi Shi, Kaiming Kuang, Yinhao Zhu, Xuanlin Li, Shizhong Han, Hong Cai, Fatih Porikli, and Hao Su. OpenShape: Scaling Up 3D Shape Representation Towards Open-World Understanding, June 2023.

[8] L. Mandelli and Stefano Berretti. *CAD 3D Model Classification by Graph Neural Networks: A New Approach Based on STEP Format*. October 2022.

[9] Alexandru Pop, Victor Domșa, and Levente Tamas. Rotation Invariant Graph Neural Network for 3D Point Clouds. *Remote Sensing*, 15(5):1437, January 2023.

[10] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision, February 2021.

[11] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering, June 2015.

[12] Divyanshu Sundriyal, Soumyadeep Ghosh, Mayank Vatsa, and Richa Singh. Semi-Supervised Learning via Triplet Network Based Active Learning (Student Abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(18):15903–15904, May 2021.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023.

[14] Nitika Verma, Edmond Boyer, and Jakob Verbeek. FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis, March 2018.

[15] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds, June 2019.

[16] Lilian Weng. Contrastive Representation Learning. https://lilianweng.github.io/posts/2021-05-31-contrastive/, May 2021.

[17] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021.

[18] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling, June 2022.

[19] Huang Zhang, Changshuo Wang, Shengwei Tian, Baoli Lu, Liping Zhang, Xin Ning, and Xiao Bai. Deep Learning-based 3D Point Cloud Classification: A Systematic Survey and Outlook. *Displays*, 79:102456, September 2023.

[20] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point Transformer, September 2021.