www.cad-journal.net
Computer-Aided Design AND APPLICATIONS

Exploring the Effect of Preprocessing on Real-world STL Model Classification Performance

Lovro Sever[1], Stanko Škec[1], Robert Mašović[1] and Tomislav Martinec[1]

[1]University of Zagreb Faculty of Mechanical Engineering and Naval Architecture

Corresponding author: Lovro Sever, lovro.sever@fsb.unizg.hr

Abstract. This paper presents the impact of preprocessing methods on 3D model classification performance, applied to a real-world dataset of 1,618 STL models of dental abutments. Unlike commonly used 3D model benchmark datasets, this dataset is characterized by a lower number of models (both in total and within classes), high similarity between classes, and inconsistent mesh quality. The study examines the impact of several preprocessing methods, including realignment, mesh repair, remeshing, and unification of facet count, coupled with five neural network-based classification algorithms: PointNet, Volumetric CNN, Multi-view CNN, FusionNet, and PVNet. The results show that preprocessing significantly improves classification accuracy, with remeshing having the most pronounced effect. The PointNet algorithm, when applied to remeshed STL models, achieved the best compromise between accuracy and processing time, based on a direct comparison of both metrics. The combination of PVNet and remeshing also produced competitive results. The study emphasizes the importance of preprocessing for classification of real-world STL datasets and identifies key challenges related to data quality, representation, and limited computational resources.

Keywords: STL model, Preprocessing, Classification, Neural network, Real-world dataset
DOI: https://doi.org/10.14733/cadaps.2026.626-645

1 INTRODUCTION

Standard Tessellation Language (STL) is a widely used data format for describing complex 3D shapes. It is commonly employed during the transition from design to manufacturing, such as in additive manufacturing [1] or CNC machining. The main weakness of the STL format compared to native CAD formats is that it contains only basic geometric information and the fact the model geometry is approximated by numerous connected triangular surfaces, each defined by three vertices and normal vectors [1]. Despite these limitations, this file format often remains the only available option for various 3D model postprocessing tasks, including dataset classification, model segmentation, or simple geometric modifications.

The challenges of classifying 3D models stored in STL format involve ensuring the efficiency and accuracy of methods for specific classification tasks. While many classification techniques are available for general use, STL model classification mainly depends on convolutional neural network (CNN)-based methods, which use different representations of STL models such as point clouds, voxels, or rendered 2D images from various viewing angles (e.g., [2], [3]). Adapted STL models can be classified with high accuracy; however, when dealing with real-world industrial datasets, new challenges emerge. Notably, training samples from industrial datasets are usually much smaller than those used in academic research, and not all files are created in controlled environments (e.g., by the same algorithm). Additionally, some meshes may have defects like duplicated facets or points, or models may be generated by combining two or more STL models.

Most relevant studies on CAD model classification focus on developing, testing, and applying neural networks for 3D geometry classification. Common case studies use benchmark models such as ModelNet10, ModelNet40, and ShapeNet (e.g., [4], [5], [6]). These benchmark datasets contain large numbers of carefully prepared models for each class. However, such datasets do not replicate real-world industrial scenarios, especially those involving highly customized or individualized products. Real-world datasets of individual products often feature multiple variations of the same item that share close geometric similarities but lack the distinctive features and classes found in benchmark collections, such as airplanes, cars, benches, or bottles. Instead, these industrial models are typically modified, configured, customized, or personalized based on specific customer needs. While most variants are similar in shape, some may show notable differences in particular geometric features compared to other versions. In addition to limited variability and scarcity of industrial models, the ways of creating and exporting geometric models to STL format can differ substantially depending on the settings used within the exporting algorithm or by employing entirely different algorithms. Consequently, industrial STL datasets often require preprocessing of models to harmonize mesh properties and model orientation before performing classification tasks.

The goal of this paper is to investigate the importance of real-world 3D model dataset preprocessing and to compare its impact across different classification methods. The novelty of this approach stems from applying and combining different preprocessing and classification methods on a real-world, industrial STL model dataset, which enables the identification of the main challenges that arise when dealing with limited, imbalanced, and geometrically similar data. The specific product selected for this study is a dental abutment – a connector piece that attaches a dental prosthetic, such as a crown, to an implant anchored in the jawbone. Dental abutments are usually personalized to fit the specific dental structure of each unique customer, while the general shape depends on the tooth position.

2 BACKGROUND

Understanding the context and methods of 3D model representation and classification is important for positioning the study within the context of processing real-world datasets. This section reviews some examples of how 3D shapes can be represented and classified, with a specific focus on STL files as input models. The STL file format is a representation of 3D geometry that describes an object's boundary as a continuous mesh of triangular surfaces, referred to as facets. Facets are defined by vertex positions and a normal vector for each facet, indicating which side of the triangular surface is inside the model [14]. Triangulation algorithms, used for generating the STL file from the native 3D model format, can introduce mesh defects such as holes, flipped normals, or overlapping surfaces [14]. These issues must be considered when analyzing models, remeshing, or transforming them into other representations.

Understanding these transformations and potential data imperfections is essential before selecting appropriate classification methods, as the choice of representation heavily influences the performance and applicability of classification algorithms.

2.1 Overview of Representations Based on STL Files

Before reviewing the classification algorithms, different types of 3D model representations, as well as their advantages and weaknesses, should be considered. This includes how STL models can be converted into different representation types and what information loss may occur during the conversion. The ability of the STL file format to include different types of data, and a comparison to other formats, was analyzed by Yuchu Qin et al. [1] in the context of additive manufacturing. When an STL file is used as the original input, important information such as product metadata and modeled features is lost. Additionally, one major drawback is the geometric inaccuracy compared to the original B-rep geometry, as a finite number of facets cannot accurately describe complex 3D surfaces.

These native models, as well as their STL conversions, can be translated into different representations such as point clouds, multi-view projections, and voxel models. Each of these representations and their combinations has been used in 3D model classification tasks reported in the literature. For example, point clouds (unlike STL models) do not contain data on the connections between points and are used as input for the PointNet algorithm [7]. One of the major challenges in this process is choosing a suitable representation-algorithm combination for accurate classification. Another challenge is training neural networks on real computers with limited RAM capacity. This issue is especially significant with voxel model representations. For instance, a 400×400×400 voxel resolution requires about 64 MB of memory. While this is not a problem for a single model, it becomes a serious constraint when processing more than 1,000 models using complex algorithms. Moreover, this resolution may not be sufficient for real-world applications, which demand higher fidelity - and therefore much more computational power [15], often unavailable in industrial settings.

This paper considers three different 3D shape representations as inputs for the classification. The first is the point cloud, which represents the model as an unordered set of point coordinates [16]. Typically, point clouds are generated directly from sensor data capturing the surface of real-world objects [17]. STL models, which describe facets using vertex coordinates [14], can be converted into point clouds by extracting these coordinates. Point clouds are memory-efficient but lack topological information, such as vertex connectivity, making direct mesh reconstruction impossible. To build a surface model from a point cloud, algorithms for surface generation must be applied [18].

The second 3D shape representation considered is the voxel model. Voxels are cubic units of fixed size that approximate a 3D model as a grid-based 3D matrix. Each voxel is either filled ("1") or empty ("0"), based on whether most of it falls within the volume of the modeled body. Complex geometries with strict accuracy requirements demand higher voxel resolutions, which increase memory usage [15].

The third 3D shape representation considered is the multi-view representation, where the three-dimensional geometry is depicted using a series of 2D images. These images are typically rendered from predefined camera angles, such as eight orthogonal views. Image processing algorithms and 2D CNNs are well-established and effective for machine learning, which is a major advantage of these approaches. Multi-view CNNs analyze the rendered 2D views separately and then combine the outputs to understand 3D geometry [9]. Although additional representations exist and can be used, such as facet center points and their normal vectors, they are beyond the scope of this paper.

2.2 Overview of Classification Algorithms for 3D Representations

Each of the aforementioned 3D representations can be used as input to specific neural networks designed to process that specific type of representation. All considered algorithms are based on convolutional neural networks (CNNs), adapted to handle different input representations. When classification accuracy is crucial, a combination of multiple input representations can be used to improve performance.

This overview covers the application of different neural network-based classification algorithms that are directly or indirectly (i.e., conversion is needed) applicable to STL files, including PointNet

[6, 7], Volumetric CNN [3, 4, 8] and Multi-View [4, 9, 10], which require only one type of input data. Algorithms that use more than one input type can be divided into two general types. In the first type, all inputs are processed jointly. In the second type, each input is processed separately, and the results are fused before generating the final output [12].

Two algorithms that combine multiple input types are examined in this paper: FusionNet [11] and PVNet [12]. Previous research on implementing, testing, and improving these classification algorithms has primarily been focused on processing benchmark 3D model datasets, such as the ModelNet and ShapeNet benchmark datasets [13]. The first considered classification algorithm is PointNet, which, as previously mentioned, uses point clouds as input. Its architecture includes max-pooling layers, structures for combining local and global features, and two joint alignment networks [7]. A similar method was used by Lipeng Gu et al. [6], whose study focused on data from Lidar sensors. Their algorithms were evaluated on benchmark datasets such as ModelNet40, ScanObjectNN, and ShapeNetPart. Their PointNet-based architecture achieved 86.4% accuracy on the real-world ScanObjectNN dataset [6].

Instead of converting models into point clouds, STL models can also be converted into voxel models. A. A. Muzahid et al. [19] explored CNN-based voxel classification methods (volumetric CNN). Similar architectures were used by C. Qi et al., achieving 87.2% accuracy on benchmark datasets [4]. C. Ma et al. developed a variant called Binary Volumetric CNN, which reached 90.69% accuracy on the ModelNet10 dataset [3]. C. Wang et al. proposed NormalNet, a voxel-based CNN that incorporates normal vectors, and achieved 91.5% accuracy.

Multi-view representations, which rely on taking 2D snapshots of the model from different angles, offer some additional advantages, given that Multi-View CNNs can analyze each 2D view independently and combine results [9]. This approach can reach an accuracy of up to 95%. A related method, MVCNN++, utilizes 2D renderings from native CAD models and achieves validation accuracy up to 95.45% [20]. Some algorithms combine multiple input types. FusionNet, for example, combines voxel and multi-view inputs. V. Hedge et al. [11] tested this method on the ModelNet10 dataset, reporting 93.11% accuracy. PVNet, another approach, fuses point cloud and multi-view inputs, and achieves 93.2% accuracy on the ModelNet10 dataset [12].

The overview of existing literature reveals one major research gap: most studies rely on benchmark datasets and do not address the problem of dataset preparation and preprocessing, which is particularly important in industrial environments and when handling real-world datasets. For example, STL model preprocessing for neural network input is relatively underexplored. In previous studies, data preprocessing is often described only as the conversion of raw sensor data into usable 3D models (3D models prepared for classification analysis). However, detailed steps focused on preparing and optimizing these models to improve classification results, such as cleaning, repairing, or normalizing, are generally not addressed. For example, raw sensor data often contains noise and other imperfections. To improve model quality, the data must be preprocessed using filtering, segmentation, and other cleaning methods [21]. Therefore, this paper investigates various STL preprocessing techniques and analyzes their impact on classification performance, specifically addressing the real-world industrial constraints.

3 METHODOLOGY

To compare different methods for classifying real-world STL files and to assess the impact of preprocessing before applying classification methods, a dataset of 1,618 individual dental abutment models was employed. This dataset is particularly relevant because it represents real industrial cases, with the abutments categorized into seven classes, each corresponding to a specific tooth position. Within each class, every abutment is unique, and its geometry can vary significantly depending on individual customer requirements. This leads to large geometric differences within the same class and, in some cases, complete dissimilarity among models in the same class, making classification a challenging problem that closely reflects real-world industrial scenarios. An overview of different preprocessing and classification steps implemented in the study is provided in Figure 1.

The first step was sorting the dataset into training and testing subsets. During the sorting process, each class had to be divided into both a training and a testing dataset. This division of models into training and testing datasets can be done automatically (by applying a designated function) or manually. Manual sorting has the advantage of ensuring the same training and testing datasets are used for all classification methods, ensuring that the results are more comparable. On the other hand, the automatic sorting function randomizes the models included in the training and test datasets, which means the same dataset cannot be ensured each time, and that the results may vary between training runs. In this study, manual sorting was used, to ensure better control over the dataset.

After collecting and organizing the dataset, there was an option of applying different preprocessing methods to adjust some of the model properties (e.g., mesh quality and model alignment) before the training. In general, the nature of preprocessing can vary depending on the type of models (3D model format) and how the models' format within the dataset was generated. For example, when all models are generated using the same CAD tool and settings, the resulting meshes will be harmonized in terms of origin position, mesh density, facet shape, etc. However, when different tools or settings are used to export STL models, these properties can vary significantly. In addition, the resulting meshes might contain errors such as missing facets or invalid facet orientation. Lack of harmonization and mesh-related errors frequently occur across industrial datasets; hence, different degrees of preprocessing can be tested to improve classification performance.

Once the models were prepared for training (with or without preprocessing), five different neural network-based classification algorithms were implemented on this dataset. The resulting trained neural network classifiers were either saved for future use or directly applied to the test dataset, as was done in the presented case.

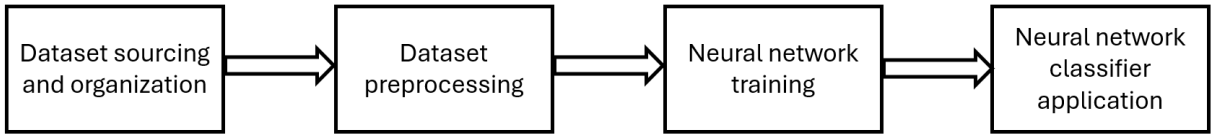

Figure 1: Study methodology.

Each of the methodology steps is described in more detail in the following subsections. As mentioned previously, different combinations of preprocessing methods were applied to the complete dataset. To analyze the impact of preprocessing, the performance of each algorithm in combination with different preprocessing methods was evaluated using four metrics, which have been divided into two main groups. The first group relates to the training process, where the time required for training (including the time for converting to the appropriate input format and preprocessing) and training accuracy were analyzed. Training accuracy is a metric that describes how well a classification algorithm can classify the data used for training. The second group of metrics is similar to the first but is related to applying the trained model to the testing dataset. Metrics in this group were used to track the time needed for classifying the entire testing dataset and to measure the testing accuracy.

3.1 Dataset Sourcing and Organization

The dataset of 1,618 models of dental abutments was provided by a company that designs and manufactures dental prosthetics and other related products. The models in the dataset belong to seven classes, each corresponding to a different position of the teeth. Figure 2 shows the layout of tooth positions within the human jaw. Teeth are labelled with two numbers. The first number marks the quadrant of the human jaw, and the second number marks the position of the tooth. Teeth in

different positions are geometrically different; for example, the tooth in position 11 is much thinner than that in position 17. This geometric difference was used as a criterion for generating seven classes, where each class corresponds to one specific tooth position.

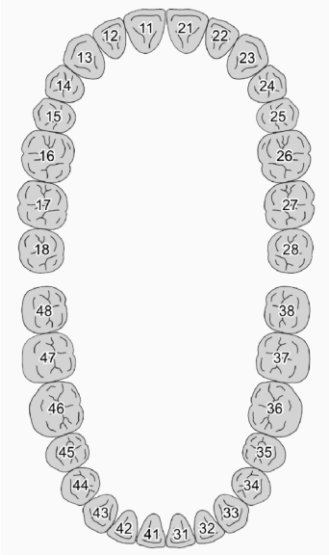

Figure 2: Teeth positions on jaw (source: [https://wiki.exocad.com/wiki/index.php/Modulo_DentalDB])

Dental abutments, which serve as a link between dental implants and prosthetic restorations (crowns or bridges), must match the shape of the specific tooth placement. Essentially, abutments can be divided into two basic groups – stock and individual. While the stock dental abutments' shape depends mostly on the tooth position, the individual ones depend on characteristics of the patient's jaw. This study is focused on individual abutments.

Classifying the models from the sourced dataset represents a challenge because, despite belonging to the same class (designed for the same tooth position), the abutments can exhibit significantly different geometric characteristics. The geometric differences within a class are primarily caused by patient-specific anatomical factors, including the unique morphology of the jaw, bone condition, and the characteristics of the remaining teeth. However, there are some cases where the difference between two classes is not obvious when performing a visual inspection of the geometry. Additionally, the models in the dataset were created by combining two STL models generated by different dental and general-purpose CAD tools, and the meshes on these models varied significantly. These parts were generated using several different CAD tools, and the origin positions and model orientations may differ.

Examples of abutment models corresponding to each class are shown in Figure 3. The models at the top represent characteristic geometries that are distinctive for a specific tooth position. The models in the middle show the "average" geometries that are most common within a class (tooth position). Finally, the bottom row shows examples of indistinct models, with geometries that cannot be visually classified in a straightforward manner. These models directly reflect the specifics of individual abutments and the related differences.

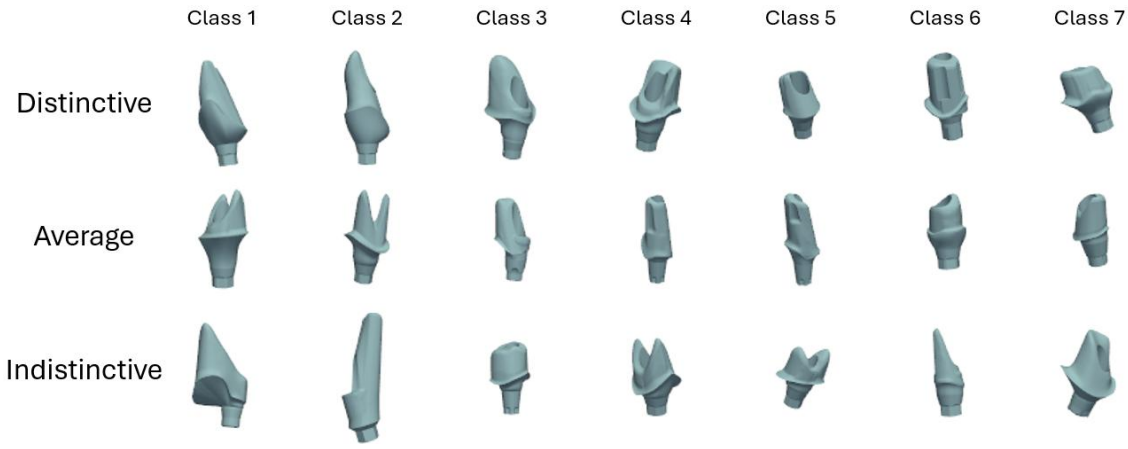

Figure 3: Examples of models from the dataset.

The previously mentioned division of models into training and testing datasets was manually implemented. This means that, for each class, the number and geometrical characteristics of models within the class were considered and the models were manually assigned for training and testing. From the collection of models that belong to the same class, 10% to 27% were selected to form the test dataset, whereas the remaining models were allocated to the training dataset. Typically, about 10% of the dataset is used for testing the classification model. However, in this specific case, for some categories that contain several times more models than the others, a larger number of models were separated to balance the number between the testing and training datasets. Table 1 shows the distribution of training and testing models for each class.

Class	Total number of models	Number of models for training	Number of models for testing	Relative size of the testing sample
1	73	65	8	11.0 %
2	76	68	8	10.5 %
3	69	62	7	10.1 %
4	227	197	30	13.2 %
5	282	250	32	11.3 %
6	745	545	200	26.8 %
7	146	126	20	13.7 %

Table 1: Model distribution on the training and testing datasets.

The largest number of models belongs to class 6, with 545 models used for training and 200 for testing the neural network classification algorithms. On the other hand, only 69 models belong to class 3, of which 62 were used for training and 7 for testing. The largest difference is between these two classes, as class 6 contains approximately eleven times more models than class 3.

3.2 Dataset Preprocessing

Four options of applying dataset preprocessing methods were considered in this study: (1) without preprocessing, (2) realignment only, (3) realignment + remeshing (fine mesh), (4) realignment +

remeshing (fine mesh) + remeshing (unified facet number). The first option, that is, without processing, implied that no modification at all would be done to the training and testing datasets (it must be noted that this option was not possible in the case of using the multi-view-based classification algorithms, since they rely on snapshots of models around the main axes). The second option included preprocessing in the form of realigning the models so that they were all placed into a similar position and orientation relative to the origin and main axes. The third and fourth preprocessing options combine realignment and two distinctive types of remeshing – into a fine mesh with a large number of facets vs. a coarse mesh with a smaller, but unified number of facets. All preprocessing methods included in the study are described in more detail in the following subsections.

3.2.1 Model realignment

Generating models with different CAD tools causes differences in model positions and orientations (relative to the origin and main axes) across the dataset. Aligning models can be done by choosing some specific geometric features on the models (e.g., a hole through the abutment), selecting the centroid of the model, or selecting the center of the bounding box and aligning it with the origin of the coordinate system. The first type of alignment (by choosing a specific geometric feature on the model) requires additional preprocessing steps, like extracting the specific feature on all models and locating its boundaries or centers. While this can result in better classification performance, the drawback is that the selected feature and the corresponding extraction method must be adjusted for the specific dataset, and the method cannot be generalized.

The other two types of alignment are not limited to a specific dataset, given that they rely on a bounding box or the centroid of the model, which can be extracted, for example, by Mirtich's method, and calculated in the same way for all types of products. These two alignment methods gave similar results when applied to the abutment dataset, with the maximum difference between the center of the bounding box and the centroid of the model being 1.24 mm. Therefore, in this study, center of the bounding box was used because it requires fewer computational resources to translate the model to the origin coordinates (X=0, Y=0, Z=0).

The translation of the models solved the problem of different origins, but better alignment required that each model is also rotated by an angle that maximizes their overlap. The models' overlap can be measured using several methods, one of which is the Hausdorff distance [22]. To solve the problem of rotation, one random model can be used as the reference model, and all other models can then be rotated around all three axes by angles that minimize the Hausdorff distance in relation to the reference model. For this purpose, each model was rotated 360 degrees in 60 steps (6 degrees per step), and for each step, the Hausdorff distance was calculated. After the Hausdorff distances were calculated for all steps, the rotation angle corresponding to the minimal distance was used to align the model. Depending on the specific case, rotation can be implemented only about one, two, or three axes. For the abutment dataset, the best result was achieved by rotating the models around one axis (Z-axis). The following figure shows an example of a reference and a dislocated model before and after realignment.

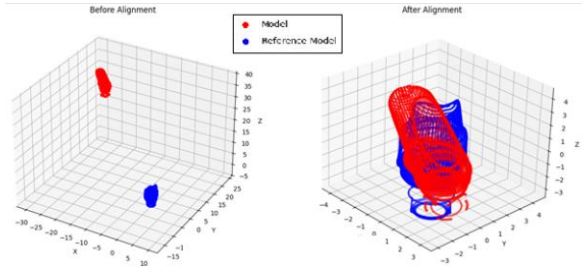

Figure 4: Example of applying the realignment method.

3.2.2 Mesh repair

Algorithms that generate STL meshes can produce errors such as flipped normal vectors, duplicated facets, and holes in the meshes (i.e., missing facets), where one of which is shown on the left side of Figure 5. Each of these errors can cause issues in subsequent preprocessing steps or significantly decrease classification accuracy. Different tools (e.g., *Autodesk Meshmixer* as well as functions from *pymeshlab* and *trimesh* Python libraries) are available for repairing the mesh.

Table 2 shows the distribution of types of errors that can be found in the original dataset. Most of these errors are not problematic for converting into representation types such as the point cloud or multi-view, but the same errors can prevent certain preprocessing methods or algorithms for conversion to other 3D model representations (e.g. conversion to a voxel model requires watertight mesh). In the used dataset, where 473 models had at least one type of error, where missing facets and non-manifold edges were present on most models.

Type of errors	Number of models
Flipped normal vectors	53
Duplicated facets	51
Missing facet	292
Non-manifold edges	186

Table 2: Overview of models with different types of errors.

In this study, the 'pymeshlab' Python library was used to repair models. The left side of Figure 5 shows an example of a missing facet. By applying the 'meshing_close_holes' function from the library, all missing faces are detected and filled. The result is shown on the right side of Figure 5.

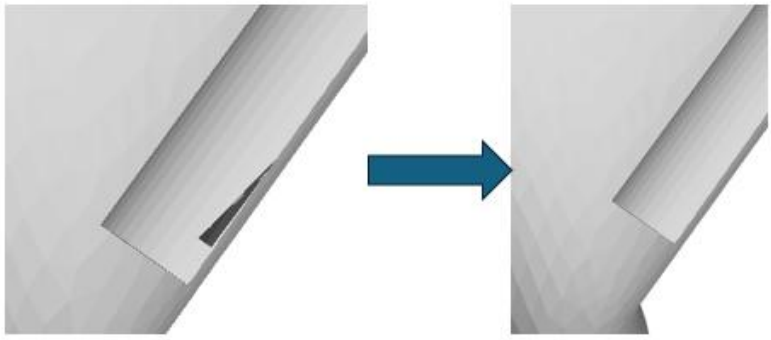

Figure 5: Example of applying the mesh repair method.

3.2.3 Remeshing to fine mesh

After fixing the meshes, each mesh can be remeshed to harmonize the mesh properties across the dataset (e.g., the size of facets and minimal angle between two edges belonging to the same facet). In this study, the remeshing was done using the isotropic explicit remeshing function in the MeshLab tool, which has a corresponding Python function within the 'pymeshlab' library.

The mesh on the left side of Figure 6 shows the original mesh with elongated, non-uniform facet sizes. On the right side of the figure, the same mesh is shown after applying the remeshing function. The resulting model has a higher mesh density (fine mesh) combined with a unified structure, that is, consistent element sizes on both flat and curved surfaces.

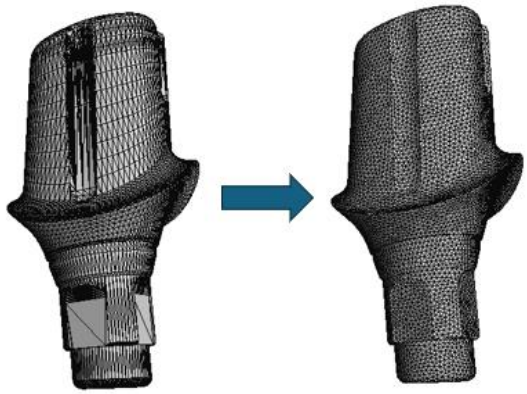

Figure 6: Example of unifying the mesh density throughout the model.

3.2.4 Remeshing to a unifying number of facets

The number of facets in the remeshed and original models vary significantly. To mitigate the impact that the different numbers of facets can have on the training process, one of the preprocessing options involved remeshing to a consistent number of facets. After applying this method, the resulting meshes did not need to have a uniform facet size, but all meshes must have contained a target number of facets, with a tolerance of 5%. This was achieved by using an appropriate function from the 'pymeshlab' Python library, which can only reduce the number of facets by merging existing facets into one. As a result, models remeshed in this way contain a similar number of points when represented as a point cloud. An example of a model before and after remeshing is shown in Figure 7. It must be noted that this method was only applied in combination with remeshing to a fine mesh (after the fine mesh was created), since this ensured consistant results across the dataset.

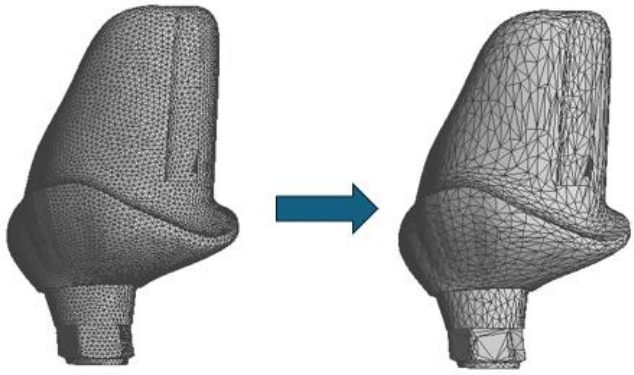

Figure 7: Example of the result of unifying the number of facets across models.

3.3 Neural Network Training

As mentioned previously, this study utilizes classification approaches based on three types of model representations: 1) Point cloud, 2) Voxel model, and 3) Multi-view. A point cloud is a type of representation that describes a model as a set of unordered points. Each point is described by its X,

Y, and Z coordinates. Point cloud models require fewer computational resources in comparison to other representations when processed by neural networks. Compared to STL models, which also include the information about coordinates of facet vertices (where 2 connected vertices form an edge) and the corresponding normal vector, point clouds do not include the information about point connectivity. This advantage is particularly expressed in cases where complex models need to be described with a large number of points, or a large number of models need to be processed during training. An example of the point cloud model representation of a dental abutment is shown on the left side of Figure 8.

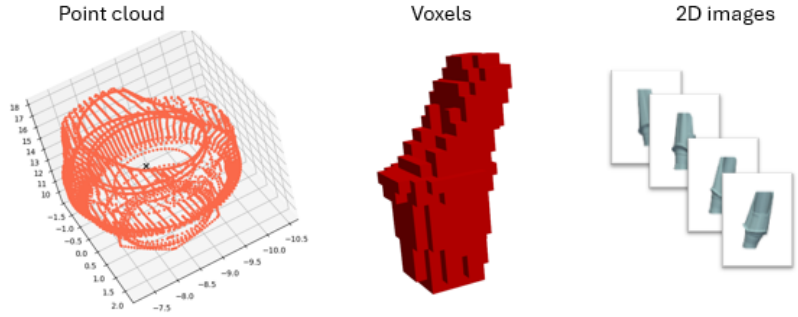

Figure 8: Types of model representation.

The second type of representation used is the voxel model. This representation describes the complete volume of the model, meaning that data loss is less significant. As mentioned earlier, the major disadvantage of using this representation is the large amount of memory needed to describe complex geometry. In this study, a resolution of 110 x 110 x 110 voxels was used, since it represents the upper limit of all available computational resources. An example of a voxel model of a dental abutment is shown in the middle of Figure 8.

The last representation used is the multi-view, an example of which is shown on the right side of Figure 8. In this case, each 2D rendered image displays the model from a different angle. The main disadvantage of this approach is the loss of 3D geometry information, such as the hollow features within the model, which cannot be captured by the images rendered from the outside. Additionally, these 2D images do not retain a direct connection to the original model, meaning that the STL mesh model cannot be restored, unlike in the case of the point cloud.

The approach to training neural networks for the purpose of model classification based on these three representation types is described in the following subsections. Each of the neural network-based algorithms used in this work can be characterized as a different type of convolutional neural network (CNN). In general, these algorithms consist of convolutional and pooling layers that are adapted to process 3D data in various formats. The number of layers and their configuration can be adjusted to better match the complexity of the data and the specific requirements of the task. In addition, several input parameters must be defined before training, such as the number of epochs, learning rate, and batch size. These parameters significantly impact the training process in terms of speed, accuracy, and overall efficiency, and can be fine-tuned based on the dataset and the desired performance of the neural network classifier.

3.4.1 Selection of classification algorithms

TensorFlow and PyTorch libraries were used for the purpose of training and applying the classification models. Each of these libraries offers certain advantages for specific cases. In general, TensorFlow is considered a more robust solution for building and deploying machine learning systems, especially in production environments. However, when flexibility is needed or rapid development and experimentation are needed, PyTorch is often the better choice. Additionally, for loading and preparing input data, the os and NumPy libraries were used.

Based on these tools, five different classification algorithms were implemented and tested. Each algorithm is designed to handle a specific type of 3D model representation, as described below:

- **PointNet:** This method is designed to directly process unordered point clouds, which are sets of 3D coordinates typically captured from sensors like Lidar. It learns both local and global geometric features from raw point sets without requiring mesh connectivity. PointNet is particularly effective for real-world sensor data.
- **MVCNN:** In this method, several 2D images are rendered from a 3D model from various viewpoints (typically 8 to 12 orthographic views). These images are then processed using conventional 2D CNNs, which are well-developed and computationally efficient. Features extracted from each view are aggregated, often via max-pooling or attention mechanisms, to form a holistic understanding of the 3D shape.
- **Volumetric CNN:** This method converts 3D mesh models into a volumetric representation by discretizing the space into 3D grids of voxels. Each voxel is marked as filled or empty. A 3D convolutional neural network is then applied over the voxel grid to learn spatial patterns. Volumetric CNNs are effective for structured data and support high accuracy on datasets like ModelNet40.
- **FusionNet:** This method integrates voxel-based and multi-view representations to benefit from both volumetric detail and visual perspectives. The architecture typically consists of two parallel branches: one for voxel input processed by a 3D CNN and another for 2D images processed by a standard CNN. The feature vectors from both branches are fused to improve classification robustness.
- **PVNet:** This method fuses point cloud data with 2D multi-view images. The point cloud is processed using a structure similar to PointNet, while the 2D views are handled by a separate CNN stream similar to MVCNN. After independent feature extraction, the network merges with these features to make the final prediction.

Finally, it must be noted that FusionNet, MVSNN, and PVNet approaches were not considered without applying one of the preprocessing methods, since they require realignment to ensure correct sampling of 2D model images. Specifically, the camera that captures the images rotates around the origin (0, 0, 0), while the center of some STL models is placed in a random position. This means that only a few images would capture the model geometry properly, and large portions of the object might be missed during rendering.

3.4.2 Training of neural network classifier

Training neural network-based algorithms involved feeding the neural network classifier with input data and adjusting its internal parameters, such as batch size, learning rate, and number of epochs, to minimize the error between predicted and actual outputs. During training, smaller portions of the dataset (defined by the batch size) were passed through the network one at time. A complete pass through the entire dataset is referred to as an epoch, and this process was repeated multiple times to improve the classifier's performance. The learning rate is a key parameter that controls how much the model's weights are updated in response to the calculated error. A properly chosen learning rate helps the neural network classifier converges efficiently, while values that are too high or too low can lead to poor training results or instability.

During training, the accuracy of the training was tracked for each epoch. The measured training accuracy shows how well the model could predict the class to which each model in the training dataset belongs. In addition to accuracy, the time needed for converting the datasets into the appropriate format and for training the classification models was also tracked.

3.4 Neural Network Classifier Application

After the neural network classifiers were trained, they were tested using the testing dataset. The testing dataset comprised a collection of models that were not used during training. The results of the testing were tracked in the form of testing accuracy, which, like training accuracy, indicates how well the classifier can correctly assign each model to its corresponding class. The second tracked

parameter was the time required to classify the entire testing dataset. This also included the time needed for preprocessing and for conversion between data representations.

4 RESULTS

The sourced dataset was classified using five different classification algorithms, both with and without the application of individual preprocessing combinations. In general, the results were compared using two main metrics: (1) accuracy and (2) time. This section is divided into two parts. The first subsection presents the results related to the training process metrics, while the second subsection focuses on the results of the testing process. The accuracy metric indicates the percentage of correctly classified models from the dataset. The time metric represents the total duration of the data preparation and the neural network training or testing steps.

All applied algorithms were based on neural networks and required both the training and the testing datasets. The training process involved tuning a large number of parameters, including batch size, the learning rate, the training/testing split ratio, and others. Depending on the algorithm used, these batch size ranged from 8 to 72, the number of epoch was set to 100, and the learning rate was set to 0.05, combined with the Adam optimizer. Additionally, an early stopping algorithm was applied, meaning that the classifier stopped training when the training accuracy converged. The experiments were conducted using the following hardware configuration: Intel Core i7-12700K, 32 GB RAM, and NVIDIA RTX A2000 GPU. It is important to note that the accuracy of each algorithm may vary slightly with each training iteration and each application of the trained classifier. This variability is a result of point sampling or small differences in generated voxel models or multi-view representation.

4.1 Training Time and Accuracy

In Table 3, the time required for data preparation, preprocessing, and training of the neural network classifier is shown for each combination of algorithm and preprocessing. The approach that required the least amount of training time was the combination of PointNet without any preprocessing, where only 25 minutes were needed from loading the dataset to producing a trained classifier. Additionally, the time required for completing each preprocessing step is also shown in Table 3. Realignment took 30 minutes, remeshing of models to fine mesh took an additional 13 minutes, and remeshing (including mesh repair) to unify the number of facets required 23 minutes.

Preprocessing method	Time (min)
Realignment	30
Remeshing (fine mesh)	13
Remeshing (unified facet number)	23

Table 3. Time needed for preprocessing steps.

Hence, when combining preprocessing combinations with the PointNet algorithm, the total training time increased significantly, reaching up to 1 hour and 18 minutes. Similar trends were observed for other algorithms, with the longest training time associated with preprocessing steps for unifying the number of facets, followed by remeshing to fine mesh. Training the Volumetric CNN and MVCNN algorithms took approximately 1 hour and 15 minutes, but converting STL models into the appropriate input format for these methods added 45 minutes to the overall process. The training time, both with and without preprocessing, for each algorithm is shown in Tables 4 and 5.

Training algorithm	Preprocessing step

	Without preprocessing	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
PointNet	25 min	55 min	1 h 8 min	1 h 18 min
VolumetricCNN	2 h	2 h 30 min	2 h 25 min	2 h 31 min

Table 4: Training time results for different combinations of training algorithms and preprocessing steps for PointNet and VolumetricCNN.

Training algorithm	Preprocessing step		
	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
Multi-View	1 h 45 min	2 h 25 min	2 h 32 min
FusionNet	3 h 32 min	3 h 42 min	3 h 51 min
PVNet	1 h 32 min	1 h 50 min	2 h 10 min

Table 5: Training time results for different combinations of training algorithms and preprocessing steps for Multi-view-based algorithms.

Table 4 includes the results for algorithms that can be performed with or without preprocessing (PointNet and VolumetricCNN), since the representations used in these algorithms was directly generated from the original STL model. The remaining algorithms, which utilize multi-view model representations (Table 5), required at least preprocessing in the form of realignment, that is, translation to the origin of the coordinate system. Therefore, the results in Table 5 include also the "without preprocessing" option.

Similar to the training time, the results related to training accuracy, shown in Tables 6 and 7, were recorded for each combination of algorithm and preprocessing steps combination. The best result was achieved using PVNet with fine-mesh models, reaching 72.2% accuracy, followed by PointNet in combination with the same remeshing option, where accuracy was lower by only 0.2 percentage points. The lowest accuracy overall was observed for PointNet without any preprocessing, with the training accuracy of 52.2 %.

Training algorithm	Preprocessing step			
	Without preprocessing	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
PointNet	52.2 %	56.1 %	72.0 %	59.3 %
VolumetricCNN	55.4 %	56.8 %	57.9 %	56.9 %

Table 6: Training accuracy results for different combinations of training algorithms and preprocessing steps for PointNet and VolumetricCNN.

Training algorithm	Preprocessing step		
	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
Multi-View	60.8 %	61.5 %	61.6 %

FusionNet	61.2 %	61.1 %	61.0 %
PVNet	61.1 %	72.2 %	61.8 %

Table 7: Training accuracy results for different combinations of training algorithms and preprocessing steps for Multi-view-based algorithms.

When considering both the time and accuracy metrics, it was concluded that the combination of PointNet with remeshing to the fine mesh provided the best training performance. Although the PVNet algorithm (combined with the same preprocessing option) achieved similar training accuracy, it was achieved with a cost of 42 minutes longer training time. Other combinations of the classification algorithms and preprocessing steps resulted in lower training accuracy, while also requiring more time. For example, the FusionNet algorithm, with the same preprocessing steps, achieved an accuracy of 61.1% and required 3 hours and 42 minutes.

4.2 Classification Time and Accuracy

After analyzing the training process, the application of the trained classifiers was evaluated in similar way, and the results are shown in Tables 8, 9, 10, and 11. PointNet required the least amount of time to classify the complete testing dataset, taking only two minutes. When additional preprocessing methods were applied, more time was needed (up to 22 minutes). The longest time was recorded for the FusionNet algorithm in combination with unifying the number of facets, where preprocessing and classification took 41 minutes. After testing all algorithms, it was observed that the most time-consuming step was the conversion from one format to another. For example, Volumetric CNN classified the testing dataset in about three minutes, but converting STL files into voxel models took around 12 minutes, which is several times longer.

Training algorithm	Preprocessing step			
	Without preprocessing	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
PointNet	2 min	7 min	18 min	22 min
VolumetricCNN	15 min	21 min	29 min	37 min

Table 8: Test classification time needed for different combinations of training algorithms and preprocessing steps for PointNet and VolumetricCNN.

Training algorithm	Preprocessing step		
	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
Multi-View	18 min	27 min	35 min
FusionNet	30 min	37 min	41 min
PVNet	15 min	19 min	25 min

Table 9: Test classification time needed for different combinations of training algorithms and preprocessing steps for Multi-view-based algorithms.

In terms of accuracy, the best results were achieved by PointNet combined with fine-mesh preprocessing, reaching 63 % accuracy. This approach was followed closely by PVNet, trained on

fine-meshed models, which achieved 62.8 % accuracy, and FusionNet, trained on realigned and fine-meshed models, with 61.2 % and 61.1 % accuracy, respectively. The lowest accuracy performance was measured for PointNet without any preprocessing, with an accuracy of 44 %. Realignment slightly improved the results for all classification algorithms, but the largest improvement was observed when the models were remeshed into fine-meshed models.

Training algorithm	Preprocessing step			
	Without preprocessing	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
PointNet	44.0 %	51.1 %	63.0 %	56.3 %
VolumetricCNN	51.2 %	54.8 %	54.9 %	54.7 %

Table 10: Test accuracy results for different combinations of training algorithms and preprocessing steps for PointNet and VolumetricCNN.

Training algorithm	Preprocessing step		
	Realignment	Remeshing (fine mesh)	Remeshing (unified facet number)
Multi-View	55.8 %	55.5 %	55.6 %
FusionNet	61.2 %	61.1 %	61.0 %
PVNet	53.8 %	62.8 %	52.2 %

Table 11: Test accuracy results for different combinations of training algorithms and preprocessing steps for Multi-view-based algorithms.

To evaluate the classification performance per class, a confusion matrix was generated for all combinations of algorithms and preprocessing options, showing the distribution of predicted labels for each actual class, based on classification accuracy per class. This allows the identification of classes with higher accuracy and those with greater confusion with other classes. Figure 9 shows the confusion matrix for the PVNet algorithm combined with fine-mesh preprocessing, which achieved the highest testing accuracy among all evaluated classifiers.

Figure 9: Confusion matrix for PVNet algorithm.

The highest classification accuracy was observed for class 6, which, as shown in Table 1, has the largest number of models included in training dataset. Other classes exhibited significantly lower classification performance. Notably, class 1 and class 2 show mutual misclassification, where model from class 1 are often predicted as class 2, and vice versa. However, due to the relatively small testing set for these classes (only 8 models each), these results should be interpreted with caution. Models from class 1 were also predicted as class 4 and class 6. class 3 had the lowest classification accuracy, with 0 % of its models correctly classified. Most of these models were misclassified as class 4, indicating a high degree of overlap or confusion between these two classes. A similar situation is observed with class 7, for which the achieved accuracy was only 6.7 %. Models from this class were predominantly predicted as class 4 and class 6.

5 DISCUSSION

By combining training and testing performance of all combinations of preprocessing options and classification algorithms, the best overall results were archived using PointNet and PVNet algorithms when applied to fine-mesh STL models. At the same time, the PointNet algorithm also produced the lowest classification accuracy when applied without any preprocessing, which highlights the significant impact that dataset preparation can have on overall performance. Although simple preprocessing methods, such as realignment, require relatively little additional time, other approaches, such as remeshing or converting to different input representations, can be time-consuming and computationally demanding. In particular, preparing the dataset for voxel-based or multi-view algorithms requires substantial time and resources. In some cases, the time required for generating these input formats can exceed the duration of the neural network training process. For example, generating voxel models or rendering multi-view images for a large dataset introduces a significant overhead, which must be taken into account when selecting preprocessing and classification strategies. Considering training performance alone, the highest accuracy was obtained using PVNet after fine mesh models. However, its testing accuracy dropped to 62.8%, which was lower than that of PointNet. This algorithm also requires at least some type of preprocessing prior to performing the training; however, fine remeshing again provided the best result compared to only realignment or unifying the number of facets.

As mentioned in the background section, the reviewed classification studies have relied primarily on benchmark datasets to measure algorithm performance. These datasets are relevant when a new classification algorithm is developed, but their results are not transferable to industrial applications. Direct comparison of used algorithms indeed confirms that they perform much better when applied to benchmark datasets (e.g. [3], [8], [9]) compared to the selected industrial case. In addition, the reviewed studies provide little or no information about the preprocessing methods used, given the nature of benchmark datasets, which have already been harmonized for research purposes.

Taking into account the classification performance and the analysis of the predicted class for each model, it can be noticed that most often the wrong predictions occur for classes 1 and 7, that is, the corresponding tooth positions one and seven. Class 1 is geometrically similar to class 2, whereas class 7 is similar to class 6. Similarity between models from different classes cannot be easily resolved. Combining additional input parameters can result in improved performance (and therefore higher training accuracy), but the neural network can become overfitted, and testing accuracy may decrease. Additionally, class 6 contains a significantly higher number of models than class 7. Coupled with model similarity, these results indicate that most models belonging to class 7 were predicted as class 6. To improve the performance of the overall neural network classifier, a larger training dataset, especially for classes with a relatively small number of models, must be collected. In summary, the best results are achieved for class 6, where almost all models from this class are correctly predicted.

In addition to the identified challenges of classifying real-world, industrial 3D model datasets, this study has several limitations. Firstly, the methods considered were tested on a single real-world dataset, meaning that the results could vary for other cases. Moreover, due to the scope of the presented research, the study was limited to five common classification methods. Further testing of additional and new methods could result in significantly better performance and must be considered

in future work. Also, as mentioned earlier, the resolution of the voxel and image model representations was relatively low, due to limitations imposed by computational resources. Future studies should investigate the resolution's effect on classification performance. The last limitation is closely related to the challenge of dealing with a real-world dataset, which in this case consisted of a relatively small number of models that were not uniformly distributed across classes. For example, one of the classes only had 70 models for network training and testing, while others had up to 500 models. Hence, the effect of increasing the number of models and harmonizing their distribution across classes should also be considered in further studies.

6 CONCLUSIONS

This study compares various methods for STL model classification on a dataset of 1,618 dental abutment STL models. Unlike commonly used datasets in CAD model classification research, this dataset is characterized by a relatively small number of models and low diversity. As such, it reflects challenges often encountered in industrial settings, namely, a limited number of models, high inter-class similarity, and restricted computational resources. The primary goal of this study was to identify preprocessing methods that can enhance classification performance.

Comparative analysis revealed that existing classification methods, applied to raw STL models, provide limited accuracy, emphasizing the need for adequate preprocessing to achieve reliable classification results. Improved results may be attainable through the application of preprocessing steps, primarily creating fine-mesh models with uniform facet size and shape. Notably, combining point cloud data implemented in the PointNet algorithm together with fine-mesh preprocessing leads to improved classification outcomes, with relatively low training times and minimal computational cost. Still, the performance of the PointNet algorithm is highly sensitive to the method used for point cloud sampling. As such, this sensitivity introduces challenges related to data volume, requiring computational resources that are often unavailable in industrial environments. If sufficient computational resources can be assured, one potential avenue for further enhancement is the use of higher-resolution voxel representations and image inputs.

Future research should focus on integrating additional input parameters and identifying dataset-specific features that could improve classification accuracy. The findings presented in this study are limited to a single dataset consisting of dental abutments, and future work should explore the effects of preprocessing on a broader range of industrial datasets, such as standard mechanical components or different types of personalized products. An additional challenge is that the original STL models were generated using different CAD tools by various users, which introduced inconsistencies in resolution and mesh generation. These variations can significantly affect classification methods that rely on point cloud representations. Therefore, the influence of such input-related factors must be examined in greater detail.

Altogether, these results confirm that industrial datasets require careful preprocessing, including consistent model alignment and orientation, mesh repair, control of mesh density, and other geometry-related corrections, in order to enable robust and accurate classification. Finally, an important aspect that warrants further investigation is the classification ability of human experts. Future research should evaluate how reliably experts who created the models can classify them within a given dataset, as well as determine the expected level of classification accuracy achievable by human evaluators.

ACKNOWLEDGEMENT

This research was funded by the project NPOO.C3.2.R3-I1.04.0121: Generative Design for Mass Personalization of Dental Implantoprosthetic Abutments (GENKON).

Lovro Sever, https://orcid.org/0009-0000-0493-3439
Stanko Škec, https://orcid.org/0000-0001-7549-8972

Robert Mašović, https://orcid.org/0000-0002-0091-0480
Tomislav Martinec, https://orcid.org/0000-0002-6487-4749

REFERENCES

[1] Qin, Y.; Qi, Q.; Scott, P.J.; Jiang, X.: Status, comparison, and future of the representations of additive manufacturing data. Computer-Aided Design 2019, 111:44 64. https://doi.org/10.1016/J.CAD.2019.02.004.

[2] Feng, Y.; Zhang, Z.; Zhao, X.; Ji, R.; Gao, Y.: GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2018, 264–72. https://doi.org/10.1109/CVPR.2018.00035.

[3] Ma, C.; Guo, Y.; Lei, Y.; An, W.: Binary Volumetric Convolutional Neural Networks for 3-D Object Recognition. IEEE Trans Instrum Meas 2019, 68, 38–48. https://doi.org/10.1109/TIM.2018.2840598.

[4] Qi, C.R.; Su, H.; Niebner, M.; Dai, A.; Yan, M.; Guibas, L.J.: Volumetric and Multi-View CNNs for Object Classification on 3D Data. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016, 5648–56. https://doi.org/10.1109/CVPR.2016.609.

[5] Su, H.; Maji, S.; Kalogerakis, E.; Learned-Miller, E: Multi-view Convolutional Neural Networks for 3D Shape Recognition 2015.

[6] Gu, L.; Yan, X.; Nan, L.; Zhu, D.; Chen, H.; Wang, W. et al.: PointeNet: A Lightweight Framework for Effective and Efficient Point Cloud Analysis. Comput Aided Geom Des 2023;110. https://doi.org/10.1016/j.cagd.2024.102311.

[7] Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 2016;2017-January:77–85. https://doi.org/10.1109/CVPR.2017.16.

[8] Muzahid, A.A.M.; Muzahid, A.A.M.; Wan, W.; Wan, W.; Hou, L.: A New Volumetric CNN for 3D Object Classification Based on Joint Multiscale Feature and Subvolume Supervised Learning Approaches. Comput Intell Neurosci, 2020, 5851465. https://doi.org/10.1155/2020/5851465.

[9] Su, H.; Maji, S.; Kalogerakis, E.; Learned-Miller, E.: Multi-view Convolutional Neural Networks for 3D Shape Recognition. 2015 IEEE International Conference on Computer Vision (ICCV), IEEE; 2015, p. 945–53. https://doi.org/10.1109/ICCV.2015.114.

[10] Ghadekar, P.; Deshmukh, M.; Deshmukh, S.; Jangid, D.; Dewalkar, D.; Dighole, R.: 3D Image Classification Based on Multi View CNN Using 2D Images. Proceedings - 2023 3rd International Conference on Innovative Sustainable Computational Technologies, CISCT 2023 2023. https://doi.org/10.1109/CISCT57197.2023.10351341.

[11] Hegde Stanford, V.; Zadeh Stanford, R.: FusionNet: 3D Object Classification Using Multiple Data Representations 2016.

[12] You, H.; Ji, R.; Feng, Y.; Gao, Y.: PVNet: A joint convolutional network of point cloud and multi-view for 3D shape recognition. MM 2018 - Proceedings of the 2018 ACM Multimedia Conference 2018, 1310–8. https://doi.org/10.1145/3240508.3240702.

[13] Mirbauer, M.; Krabec, M.; Krivanek, J.; Sikudova, E.: Survey and Evaluation of Neural 3D Shape Classification Approaches. IEEE Trans Pattern Anal Mach Intell, 2022, 44, 8635–56. https://doi.org/10.1109/TPAMI.2021.3102676.

[14] Szilvási-Nagy, M.; Mátyási, G.: Analysis of STL files. Math Comput Model, 2003, 38, 945–60. https://doi.org/10.1016/S0895-7177(03)90079-3.

[15] Patil, S.; Ravi, B.: Voxel-based representation, display and thickness analysis of intricate shapes. Proceedings - Ninth International Conference on Computer Aided Design and Computer Graphics, CAD/CG 2005, 2005, 415–20. https://doi.org/10.1109/CAD-CG.2005.86.

[16] Han, X.F.; Jin, J.S.; Wang, M.J.; Jiang, W.; Gao, L.; Xiao, L.: A review of algorithms for filtering the 3D point cloud. Signal Process Image Commun 2017, 57, 103–12. https://doi.org/10.1016/J.IMAGE.2017.05.009.
[17] Woo, H.; Kang, E.; Wang, S.; Lee, K.H.: A new segmentation method for point cloud data. Int J Mach Tools Manuf, 2002, 42, 167–78. https://doi.org/10.1016/S0890-6955(01)00120-1.
[18] Yang, Y.; Lin, H.; Zhang, Y.: Content-based 3-D model retrieval: A survey. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews 2007, 37, 1081–98. https://doi.org/10.1109/TSMCC.2007.905756.
[19] Muzahid, A.A.M.; Wan, W.; Sohel, F.; Khan, N.U.; Villagomez, O.D.C.; Ullah, H.: 3D Object Classification Using a Volumetric Deep Neural Network: An Efficient Octree Guided Auxiliary Learning Approach. IEEE Access 2020, 8, 23802–16. https://doi.org/10.1109/ACCESS.2020.2968506.
[20] Angrish, A.; Bharadwaj, A.; Starly, B.: MVCNN++: Computer-Aided Design Model Shape Classification and Retrieval Using Multi-View Convolutional Neural Networks. J Comput Inf Sci Eng 2021;21. https://doi.org/10.1115/1.4047486/1084494.
[21] Su. H.; Maji, S.; Kalogerakis, E.; Learned-Miller, E.: Multi-view Convolutional Neural Networks for 3D Shape Recognition. 2015 IEEE International Conference on Computer Vision (ICCV), IEEE; 2015, 945–53. https://doi.org/10.1109/ICCV.2015.114.
[22] Marošević, T.: 2018. The Hausdorff distance between some sets of points. Mathematical Communications, 23(2), 2018, 247–257. Available at: https://hrcak.srce.hr/198692