STAG: Smart Tools and Applications in Graphics (2022), pp. 1–10
D. Cabiddu, T. Schneider, and G. Cherchi (Editors)

CAD 3D Model classification by Graph Neural Networks: A new approach based on STEP format

paper id #6

Abstract

In this paper, we introduce a new approach for retrieval and classification of 3D models that directly performs in the CAD format without any format conversion to other representations like point clouds of meshes, thus avoiding any loss of information. Among the various CAD formats, we consider the widely used STEP extension, which represents a standard for product manufacturing information. This particular format represents a 3D model as a set of primitive elements such as surfaces and vertices linked together. In our approach, we exploit the linked structure of STEP files to create a graph in which the nodes are the primitive elements and the arcs are the connections between them. We then use Graph Neural Networks (GNNs) to solve the problem of model classification. Finally, we created two datasets of 3D models in native CAD format, respectively, by collecting data from the Traceparts model library and from the Configurators software modeling company. We used these datasets to test and compare our approach with respect to state-of-the-art methods that consider other 3D formats.

CCS Concepts

• ***Computing methodologies*** → ***Shape modeling;*** • ***Information systems*** → ***Information retrieval;***

1. Introduction

3D model designers spend a big part of their time searching for the right information during the product design process and most of their design could be created by modifying an already existing CAD model. For this reason, the retrieval and reuse of CAD models play a very important role in the domain of CAD model management. However, huge repositories of CAD models need a priori categorization on engineering data or require significant organization making design reuse a hard task. As a matter of fact, traditionally, 3D CAD model classification and retrieval are achieved by a time-consuming and troublesome human manual process of labeling that is prone to errors. These problems are even more frequent if the models are generated by product development activities and they are so specified by different labeling and tags that would require a difficult process of data harmonization. Furthermore, due to the intrinsic complexity of 3D CAD model definition, no rigid and general rules can be applied in model classification. In fact, depending on product origin, the features and parameters of the models may vary significantly. An automatic classification and retrieval approach is then needed to overcome these difficulties.

In most of the previous works, the 3D model classification problem was approached using machine learning techniques. However, in those methods, the 3D data is usually taken from sensors that can scan real objects. 3D data from sensors is typically represented in various formats such as depth images, point clouds, meshes, or grid voxels, whereas 3D CAD data created with modeling software is typically stored in one of the CAD formats. The main state-of-the-art methods that work on 3D models are based on models taken from sensors without considering the possibilities of directly using data in CAD formats. In general, the information contained in a CAD model is greater than that of the same model expressed in other 3D formats, as it allows access to all the information stored during model creation. For example, a model with a cylindrical shape can be represented in the CAD format by the equations describing the surface of the cylinder, while in other 3D formats it is represented by a collection of many primitive elements such as point clouds, voxel grids or triangular meshes. Therefore, the conversion from one CAD format to other 3D formats is achievable simply by sampling respective elements in the model, while the opposite is not always possible due to the loss of information.

In this paper, we focus on models in a native CAD format, and we propose a deep learning approach to automatically classify 3D CAD models. In particular, among the various CAD formats we consider the widely used STEP extension, which represents a standard for product manufacturing information. This format represents a 3D model as a set of primitive elements such as surfaces and vertices linked together. In our approach, we exploit the linked structure of STEP files to create a graph in which the nodes are the primitive elements and the arcs are the connections between them. We then use graph neural networks to solve the problem of model classification and retrieval. To the best of our knowledge, no past research does exist that tries to exploit the STEP format for this

purpose. Furthermore, as there are no large datasets of natively 3D CAD models, we built two STEP datasets by finding models from the Traceparts online library [Gro] and by collecting models from the Configurators [Srl] software modeling company. Finally, we tested our approach with the collected dataset, and compared it with the classic state-of-the-art methods based on 3D data.

The contributions of this paper are summarized as follows:

- We proposed a new approach that consider data directly in a CAD format by transforming STEP files in graphs;
- We constructed two datasets of native CAD models, gathering elements from the Traceparts open CAD library and from the Configurators [Srl] software company, and organized them into classes;
- We tested our approach and compared it with state-of-the-art methods that work on general 3D models.

This rest of the paper is organized as follows: in Section 2, we summarize the existing methods the address he problem of retrieval and classification of 3D data. In Section 3, we present our proposed approach; in particular, in Section 3.1, we discuss the characteristics of the STEP format and how to convert 3D models to graphs, and in Section 3.2, we describe the proposed Graph Neural Network (GNN) for comparing and classifying the obtained graphs. In Section 4, we describe the collected datasets and the experiments carried on them to validate our approach. Finally, in Section 5, conclusions and future research directions are reported.

2. Related work

The number of works that perform 3D model retrieval or classification by directly operating on CAD models is very limited and, to the best of our knowledge, we are the first who try to exploit the structure of the STEP format. In the following, we discuss the methods in the literature that worked on 3D data: we divided them into model that are based on the extraction of low level features, or that directly use point clouds, voxel grids, model views, or user sketches.

2.1. Feature-based approaches

A general approach to handling 3D models is to extract some engineered features to summarize the content of the models in vectors (feature-based techniques) or consider the models as polygon meshes and obtain different transformation invariants to measure the similarity among 3D models (shape-based techniques). In both the approaches, the descriptors obtained are then compared or used as input for supervised machine learning techniques to solve the problem of classification and retrieval.

In [Ip05], several shape descriptors (*e.g.*, Zernike descriptor [NK04]) Multi-resolution Reeb graph descriptor [HSKK01] and spectral properties) were extracted to represent a model as n numerical attributes and then a Nearest neighbor classifier or a Support Vector Machine is applied. In [IR05], for each model a shape distribution vector [RTBD02] is computed by sampling points and creating a histogram from theirs properties. The histograms are then compared with curve matching techniques (*e.g.*, Minkowski L_N distance). In [CDT*03] and [QFL*14], the LFD (Light Field Descriptor) [CDT*03] view-based features were used, assuming that two 3D models belonging to the same class look similar from all viewing angles. Using LFD, features were extracted further (using the Zernike moments and the Fourier descriptor) after 2D images were generated from 3D models through light field projection. The vectors obtained were finally passed to a deep neural network to solve the classification problem. In [LWM*20], the CAD model files were converted to a frequency domain representation and then the corresponding spectrograms were compared.

2.2. Voxel grid approaches

In [MS15], trying to overcome the feature-based methods, authors developed a new approach by considering the models as 3D grids and processing them through a supervised network that uses 3D convolutions (3D CNN). This machine learning end-to-end approach allows the neural network to learn which are the most significant features instead of defining them manually. More recently, in [GCD*21] authors sampled from the CAD models a grid 32^3 and extracted in the same way a descriptor. In [ZLLG18], authors proposed LightNet, a volumetric CNN architecture with a small number of training parameters, to address the real-time 3D object recognition problem by leveraging multitask learning.

2.3. Point cloud approaches

Point clouds are becoming popular approaches for representing 3D objects. It allows feeding point clouds to deep networks directly since it does not need any transformation to the input data, such as voxelization or projection and so any loss of information is avoided. The difficulty lays on the fact that point cloud data is spatially irregular and permutation invariant, which is essentially different from rasterized data as grid voxels or pixels.

In [QRS*16] and [QYSG17], Qi *et al.* directly used point cloud data, represented by three coordinates, to feed a networks able to perform feature transformations and aggregate data points by max pooling. In [NWSL20], authors utilized the PointNet++ model to segment each 3D model into a set of shapes and extracted their features. The K-means method was then utilized to construct from each different shape a node of a 3D shape knowledge graph. Then, a graph embedding was performed based on the 3D shape knowledge graph and an entities' retrieval method was used to handle the 3D model retrieval problem. In [KRLV17], authors used a *kd*-tree to represent point clouds. The network performed the operations in CNNs adapted to the use of data structures as an input.

2.4. Model view approaches

In MVCNN [SHM*15], exploiting the success of the Convolutional Neural Netwoks (CNN), authors utilized images taken from 12 or 80 multiple 2D views around the 3D model, which were aggregated by a view-pooling layer, then passed to a CNN pre-trained on ImageNet to generate a compact shape descriptor. The problem of 3D model classification is so defined as a view recognition problem. In [SKT*17] and more recently in [SPT18], a special view of the 3D model, the PANORAMA representation, was obtained and then

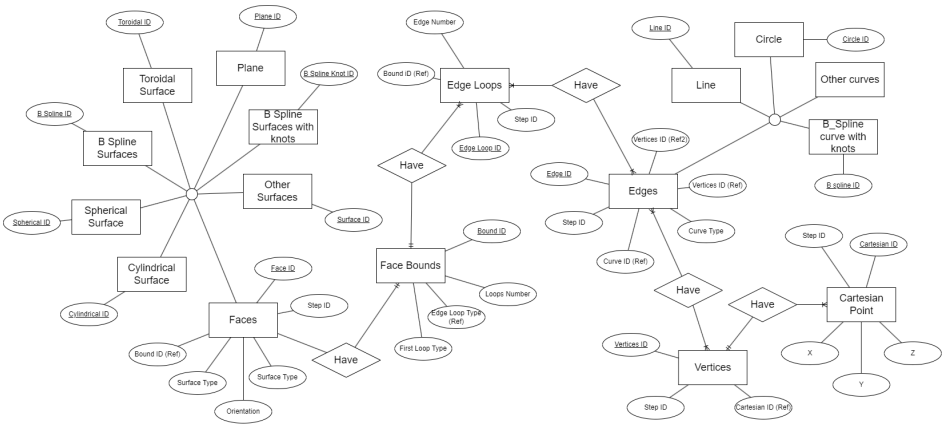

Figure 1: Entity relationship diagram that shows some of the entity defined in the STEP standard.

passed to a Convolutional Neural Network to classify the models. In RotationNet [KAM*16], multiple 2D views of an object were used as input and the network jointly estimates their pose and the object category.

2.5. Sketch-based approaches

Many works considered the problem of retrieval of CAD models with a sketch-based approach: a 2D sketch is given as query input and from it a feature vector is extracted by using image descriptors. The vector is then used to measure the similarity with the feature vectors extracted from the 3D models stored in the database. This kind of sketch-based methods need therefore to project the 3D model into a 2D-view with the best angle and select an image descriptor capable of matching the view and the hand-drawn sketch.

Hou *et al.* [SK07], [SK06] proposed a sketch-based retrieval method using three views of the 3D models and by combining image descriptors such as spherical harmonics, Fourier transform and Zernike moment. In [QQGB22], authors proposed an approach based on sketches and unsupervised learning that extracted geometric information and structural semantic information from the views and the sketch.

3. The proposed approach

Our approach consists of two steps: first, we transformed the STEP files into graphs as described in Section 3.1, then we apply the GNNs for the classification and retrieval tasks in Section 3.2.

3.1. From STEP to Graph

Here, we first illustrate the main characteristics of the STEP format, then we propose a solution to pass from the STEP format to a graph representation.

3.1.1. STEP format

The STEP format (Standard for the Exchange of Product model data) is a widely used ISO standard for data exchange that can represent 3D objects in Computer-aided design (CAD) and related information.

A generic STEP-file is composed of a list of instances of standard entities that are stored in the file as lines as it is shown in Code 1. Every instance has its own unique *id* and it is characterized by a series of attributes passed as arguments that can be numeric values, string values or references to other instances. The values of the attributes differentiate between instances of the same entity: two instances of the entity *Point* vary for the numeric values of the coordinates x, y, and z passed as arguments.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(
/* description */ ('A minimal AP214 example with a single part'),
/* implementation_level */ '2;1');
FILE_NAME(
/* name */ 'demo',
/* time_stamp */ '2003-12-27T11:57:53',
/* author */ ('Lothar Klein'),
/* organization */ ('LKSoft'),
/* preprocessor_version */ ' ',
/* originating_system */ 'IDA-STEP',
/* authorization */ ' ');
FILE_SCHEMA (('AUTOMOTIVE_DESIGN { 1 0 10303 214 2 1 1}'));
ENDSEC;
DATA;
#10=ORGANIZATION('00001','LKSoft','company');
#11=PRODUCT_DEFINITION_CONTEXT('part definition',#12,'
     manufacturing');
#12=APPLICATION_CONTEXT('mechanical design');
#13=APPLICATION_PROTOCOL_DEFINITION('','automotive_design'
     ,2003,#12);
#14=PRODUCT_DEFINITION('0',$,#15,#11);
#15=PRODUCT_DEFINITION_FORMATION('1',$,#16);
#16=PRODUCT('A0001','Test Part 1','',(#18));
#17=PRODUCT_RELATED_PRODUCT_CATEGORY('part',$,(#16));
#18=PRODUCT_CONTEXT('',#12,'');
#19=APPLIED_ORGANIZATION_ASSIGNMENT(#10,#20,(#16));
#20=ORGANIZATION_ROLE('id owner');
ENDSEC;
END-ISO-10303-21;
```

Code 1: Example of a STEP file.

The entities that the format considers are geometric primitives such as *points*, complex geometric surfaces such as *splines* and semantic properties of the model such as *name*, *security level* and domain.

An entity relationship diagram that shows some of the entities defined in the STEP standard is shown in Figure 1.

Using the STEP format over other formats for the representation of 3D objects has the following advantages:

- It contains more information than just the geometry of the objects, such as the domain to which they belong and semantic information;
- It allows defining a complex surface through a few nodes, reducing the redundancy of information. For example, a cylindrical surface is defined just by its radius, its height and a spatial point for the center of the base. On the contrary, to represent such a surface in a point cloud or grid format it would be necessary to sample the space in a large number of elements;
- Compared to formats that represent an object with multiple 2D views, the STEP format allows considering information on internal elements or, in general, parts of the object not visible from the outside;
- In real business contexts, CAD models are often created by assembling smaller components in order to facilitate reusability. From the structure of the STEP file, it is possible to derive its hierarchical structure and use it as additional information.

The presence of geometric implicit surfaces, the semantic information and the hierarchical structure of the CAD files therefore offer greater information than classic formats such as mesh, point cloud or voxels.

3.1.2. Graph conversion

From each STEP file, we aim to obtain a graph. This is achieved by parsing the file line by line and each entity instance present in each line is coded as a node. Each instance is then characterized by a series of attributes passed as an argument, which can be numeric, strings or references to other instances. The references between instances are encoded as arcs in the graph, numerical values and strings as the attributes of the nodes. An example of the conversion between a STEP file and a graph is shown in Figure 2.

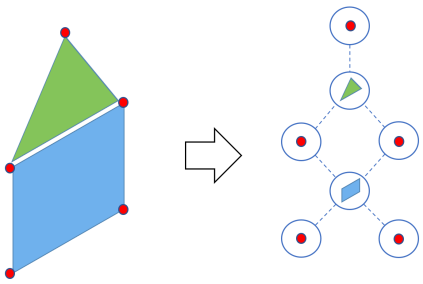

Figure 2: Example of conversion from the geometric information of a STEP file to a graph structure.

From the STEP file, it is also possible to derive whether the 3D model is made starting from a sub-component structure. Generally, during a CAD modeling phase, the overall model is created by assembling smaller components previously made. For example, a bicycle model can consist of two wheel components and a frame component, which can be composed by even more primitive components. In this way, a single component can be easily reused several times in the same model or in different models without having to create it every time from scratch. The hierarchical structure of the STEP files allows us to decompose a model encoded as a graph into several smaller sub-graphs, as it is shown in Figure 3.

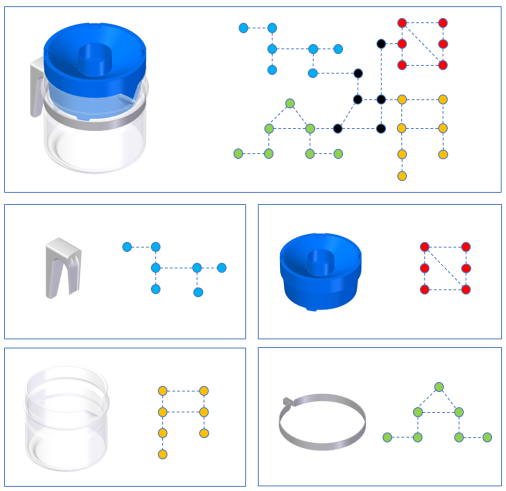

Figure 3: Example of the decomposition into components of a CAD file and its related sub-graphs.

Each sub-graph represents a component of the overall model. The breakdown into components provides two advantages:

- We can use the additional information of the hierarchical structure of the models;
- Comparing the various sub-graphs with each other is much faster than comparing two overall graphs.

3.2. Graph classification and retrieval

After creating graphs from 3D CAD models in STEP format, we want to classify them using an end-to-end graph neural network. The classification network can then be used for the retrieval task by extracting feature vectors from intermediate layers of the network and comparing them using various metrics.

For this task, we hypothesized that similar models can be defined by a finite set of possible combinations of primitives and therefore of different graphs. For example, in the design phase, a sphere can be realized by a single sphere or by two half-spheres depending on the modeling process. However, it will hardly be realized by infinite representations of unique combinations of primitive elements. Hence, the number of possible graphs describing a geometry will be limited.

For simplicity, in this study that fist explores the potentiality of considering STEP files as graph input data, we did not consider the attributes of the nodes but only their typology. Each node is then represented by a vector of dimension $|P|$ that indicates the one-hot coding of the node type, where P is the set of possible entities defined in the STEP standard. To restrict even more the size of the vector, we considered $P' \subseteq P$ representing all the entities contained in the STEP models of the considered dataset.

Graph Neural Network: GNNs are a class of deep learning

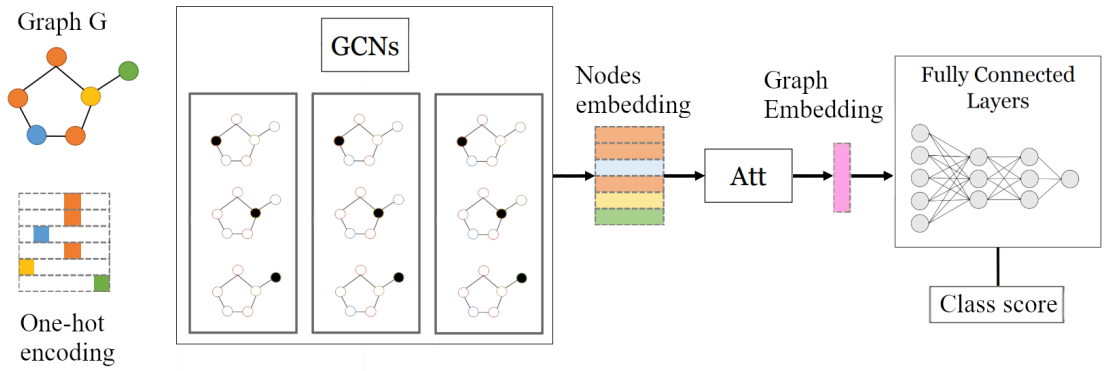

Figure 4: Structure of the proposed GCN network. The GCN layers generate the nodes embedding that are then passed together through an attention mechanism which generates an embedding of the entire graph. Two fully connected (FC) layers and the softmax function lead to the classification score.

methods designed to perform inference on data described by graphs in order to provide an easy way to do node-level, edge-level, and graph-level prediction tasks.

Graph Convolutional Networks: GCNs are a type of GNNs based on an efficient variant of convolutional neural network that operates directly on graphs. Let $G(\mathcal{V},\mathcal{E})$ be an undirected graph with $\mathcal{V}$ the set of nodes and $\mathcal{E}$) the set of the edges. Let A the adjacency matrix of G, I the identity matrix and $\tilde{A} = A + I$ the adjacency matrix with self-connections. Formally, the graph convolution can be modeled as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}), \tag{1}$$

where $\tilde{D}$ and $W^{(l)}$ are trainable weight matrices and σ denotes an activation function of the l-th layer. $H^{(l)} \in \mathcal{R}^{N\times D}$ is the matrix of activations in the layer, where $N = |\mathcal{V}|$.

This method scales linearly in the number of graph edges, and it is therefore suitable in our case where the graphs can consist of thousands of nodes and arcs. This operation allows us to learn hidden layer representations that encode both local graph structures and the features of nodes. The GCN produces a reduced and meaningful representation of the nodes of a graph.

In order to obtain an overall graph descriptor, the information of the nodes must be summarized into a single vector that represents the whole graph. Instead of performing an unweighted average of node embeddings or a weighted sum, where the weight associated with a node is determined by its degree, we implemented an attention mechanism to let the model learn weights following the ideas of [BDB*]. In this way, the nodes that are more important will receive more weights and the model will learn by itself the best weights through the backpropagation steps.

Let's $U \in \mathcal{R}^{N\times D}$ be the nodes representation, where the n-th row represents the embedding u_n of node n. For a node n the node attentive mechanism h is calculated as follow:

$$\begin{aligned} h &= \sum_{n=1}^{N} \sigma(u_n^T c)u_n \\ c &= tanh(\frac{1}{N}W\sum_{n=1}^{N} u_n), \end{aligned} \tag{2}$$

where c represents a global context of the graph and it is obtained by a simple average of node embeddings followed by a nonlinear transformation $C \in \mathcal{R}^D$. The function σ is the sigmoid function and $W \in \mathcal{R}^{D\times D}$ is a learnable weight matrix. The attention weight for each node is calculated by an inner product between c and its node embedding. In this way, nodes that are more similar to the global context will receive higher attention weights. The resulted vector, that represents the information of the whole graph, is then passed to two fully connected layers, reducing the vector size to the number of classes specified by the classification task. A summary scheme of the architecture used is shown in Figure 4.

4. Experiments

In order to evaluate our proposed approach, we first collected two datasets of CAD models in STEP format as discussed in Section 4.1. Then, we performed classification and retrieval experiments on those datasets, respectively, in Section 4.3 and Section 4.2. Classification results are also compared with state-of-the-art solutions that work on different data format. Evaluations while varying different parameters and settings for our approach are also reported,

4.1. Datasets

Large datasets of native CAD models do not exist. While converting CAD format to other formats such as point clouds, views, voxels or meshes is possible, the reverse process is not since CAD models contain more information than that available in the

other formats. Not being possible to use the major 3D model datasets such as ShapeNet [CFG*] or ModelNet [WSK*15] for our purposes, we have created and used for our tests two CAD datasets (the TraceParts amd the Configurators ones), and made the first one available at the link https://mega.nz/folder/YfESgYDA#FNFLcCsg8mOkb6O2y0o-tA.

TraceParts CAD dataset; The first dataset consists of 6 classes with 100 CAD models each. The models were obtained from the free online TraceParts CAD library [Gro], which stores many STEP models of various categories produced by different companies and with different modeling processes. Unfortunately, the components of the models in the library are not defined, but the models are treated as a single piece. Therefore, it was not possible to exploit the hierarchical composition of the CAD models as discussed in Section 3.1.2.

The classes considered belong to the domain of mechanical components and are given by, respectively, *screws*, *nuts*, *hinges*, *fans*, *iron bars*, and *wheels*. Examples of the six classes of the dataset are shown in Figure 5.

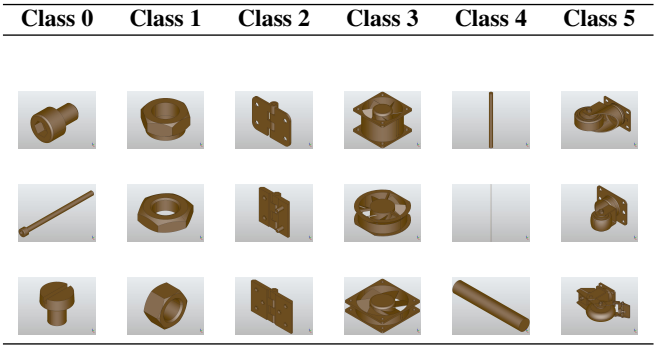

Figure 5: Examples of the models contained in the collected Traceparts CAD dataset.

The Traceparts library collects templates from different companies. In this way, two visually very similar models that belong to the same class but that are produced by different companies can be generated in different ways and therefore have different graph structures. For example, a sphere can be defined by a company *A* starting from a single sphere, while a company *B* can define it as the union of two half spheres.

On the other hand, two models belonging to the same class but visually different, if generated by the same company, can be represented by graphs with similar topology. In this case, the visual differences between the models are given by the different attributes of the nodes in the graphs. For example, two different spheres, one very large and one very small, will be characterized by the same graph structure but different radius values in the node that defines the sphere. In Figure 6, we show some examples of visual similarity compared to graph similarity in the case of models made by the same company and different companies.

Table 1 shows the characteristics of the graph nodes in each class. The average number of nodes varies greatly depending on the class: screws, nuts, and iron bars have hundreds of nodes, whereas

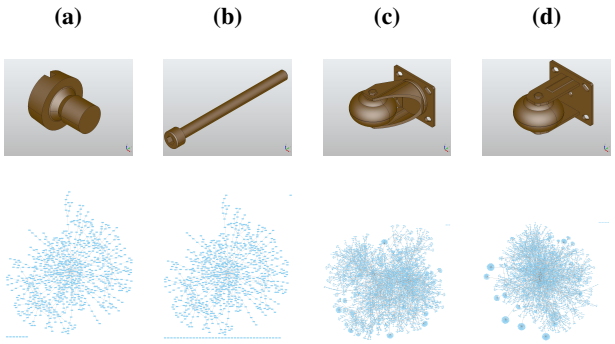

Figure 6: Examples of CAD models in the dataset and their respective generated graphs. 3D models belonging to the same class, **(a)**-**(b)** and **(c)**-**(d)**, can be represented by graphs with similar structure even if they are visually different such as **(a)**-**(b)** or by graphs with different structure even if they are visually similar such as **(c)**-**(d)**.

hinges, fans, and wheels have thousands. Some classes are therefore represented by a greater number of nodes, corresponding to more complex graphs. While a screw or iron bar can be represented with a relatively small graph, a wheel needs 10 times the number of nodes to be represented with a graph derived from the original STEP file.

The variance in the number of nodes also varies considerably across classes. In particular, the classes of screws and iron bars are characterized by zero variance, despite the fact that models within them are visually different. This is due to the fact that all models in those classes are made by a single company and are therefore characterized by a single style of model creation, which corresponds to a very similar graph structure.

Class	Mean # nodes	Variance # nodes
screws	773.0	0.0
nuts	981.4	403428.7
hinges	3282.6	2987419.7
fans	3782.2	18587.3
iron bars	130.0	0.0
wheels	9307.2	8803938.1

Table 1: Properties of the models in the classes of the proposed Traceparts CAD dataset.

Configurators CAD dataset: The second dataset was created from the models made available thanks to the software modeling company Configurators [Srl]. It is made up of 8 classes of 50 elements each. Examples of models for each class are shown in Figure 7. In Table 2, we show the characteristics of the graphs for each class.

This dataset contains models taken from a real-world working application in which each class represents a different component of a single construct. On average, these 3D CAD models are more complex than those included in the TraceParts CAD dataset and the classes are more similar to each other. For example, classes 0 and 1 vary only by the position, vertical or lateral, of a hole on

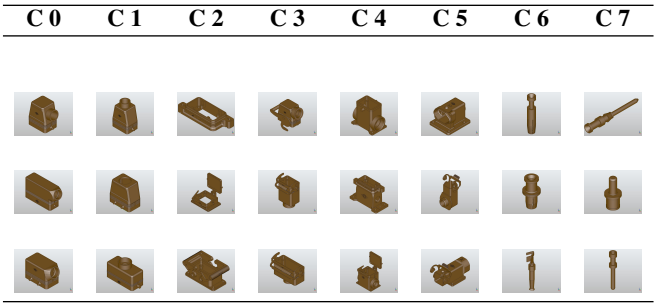

Figure 7: Examples of the models contained in the collected Configurators CAD dataset.

Class	Mean # nodes	Variance # nodes
0	11244.0	3112635.5
1	10300.4	1799380.8
2	13027.9	15810147.7
3	18241.6	12651546.6
4	14707.9	14708304.2
5	15748.2	23589269.2
6	6784.2	10737704.7
7	958.9	34371.9

Table 2: Properties of models for the classes of the Configurators CAD dataset.

the structure. This allows us to test our proposed method in a more challenging and real world application scenario that can also stress more the differences with other existing solutions in the literature.

4.2. GCN-based CAD model classification

After transforming the STEP models into graph data, the information contained in the nodes is summarized into vectors. The total number of different types of STEP entities contained in the datasets models is 80, therefore each node is represented by a vector of dimension 80 obtained through the one-hot encoding of its entity type.

We then tested the GCN network described in Section 3.2 on the two graph datasets. The network has been trained for 50 epochs for the first dataset and 100 epochs for the second one with an initial learning rate (lr) equal to 0.0005 and a scheduler mechanism that decays the lr by a gamma value equal to 0.1, when the current loss value is greater than the average of the last 6 epochs.

Each dataset was divided into a train set and a test set with a 90/10 split; furthermore, 10% of the train set models were taken to build a validation set. We therefore get, for the first dataset, 486 models for the train set, 60 models for the test set and 54 models for the validation set. For the second dataset we obtained 324 models for the train set, 40 models for the test set, and 36 models for the validation set. We set the GCN with 3 convolutional layers with dimensions of 64, 32 and 32, followed by the attention mechanism and two fully connected layers with sizes 32 and 6 or 8, equal to the number of classes for the first and second datasets, respectively.

This configuration was decided after a set of tests performed on the first dataset as shown in Table 3; in these test, the network performance were compared while varying the size of the first FC layer (*i.e.*, the bottleneck layer). By increasing the size of the bottleneck, we were able to increase the expressive power of the network to 100% accuracy.

GCN bottleneck size	Accuracy	Loss	Precision
8	91.66%	0.84	0.93
16	100.0%	0.55	1.00
32	**100.0%**	**0.51**	**1.00**

Table 3: Study on the effect of varying the size of the GCN bottleneck layer when testing on the first dataset.

Regarding the attention mechanism, we studied how it helps to obtain a general graph representation from its nodes embedding. We compared our network with an attention mechanism with two other networks that merge nodes information through an unweighted average of node values and a weighted sum, where the weight associated with a node is determined by its degree. Results reported in Table 4 indicate that the attention mechanism allows us to significantly increase the accuracy of the model compared to other non-learning-based methods.

Network module	Accuracy	Loss	Precision	Recall
Unweighted average	93.33%	0.68	0.942	0.933
Degree weighted sum	93.33%	0.64	0.952	0.933
Attention	**100.0%**	**0.51**	**1.000**	**1.000**

Table 4: Study on the attention mechanism on the first dataset to obtain an overall representation of the graph starting from the embedding of its nodes compared to: the use of a unweighted average; a weighted sum based on the degree of the nodes.

We compared our methods with other existing approaches that operate in the domain of 3D but using different formats for the data. We considered the following methods:

- **PointNet++**: we sampled from each of our CAD models a point cloud in order to train the PointNet++ network [QYSG17] that represents a standard in 3D model classification. As the number of sampling points, we considered 1024, 2500 and 1000 points, but we did not find any significant variation. The network has been trained for 50 epochs with a learning rate equal to 0.001.
- **MVCNN**: 12 2D views for each model are generated through a mobile camera that rotates around the model. Each image is resized to 224×224, passed through the convolutional layers and then aggregated at a viewpooling layer with the representations of the other images of the same model. The MVCNN [SHM*15] network has been trained for 30 epochs with a learning rate equal to $5e^{-5}$. The same setting is used for the Single View Convolutional Network (SVCNN) variant that consider each image a separate input data.

Figure 8 highlights the different types of inputs for the different approaches.

Results of the comparison of those methods with our approach for the classification task are shown in Table 5 for the Traceparts

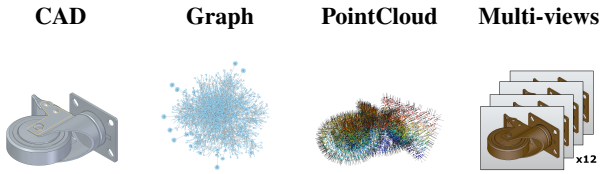

Figure 8: Summary of the different input types of the tested approaches for a single CAD model.

dataset and in Table 6 for the Configurators dataset. Regarding the first dataset, our GCN is able to achieve an accuracy score of 100% outperforming the PointNet and PointNet++ networks trained on the same data, slightly beating the Single-View Convolution Neural Network (SVCNN) variant and achieving the same perfect accuracy score as the MVCNN method.

Method	Accuracy
PointNet	28.7%
PointNet++	31.6%
SVCNN	99%
MVCNN	**100%**
GCN (ours)	**100%**

Table 5: Comparison on the Tracepart dataset of our approach based on STEP data to others state-of-the-art methods that works on different data formats.

In the second dataset, our approach confirms its validity achieving the value of 97.5% of accuracy and therefore exceeding the results of the MVCNN method. This is due to the fact that, unlike the 2D view method, the graph-based approach is able to exploit information that is not visible externally and is not affected if two models belonging to different classes are visually similar, as for example, happens for the classes 0 and 1 or for the classes 3, 4 and 5.

Method	Accuracy
PointNet	18.7%
PointNet++	28.5%
SVCNN	91.4%
MVCNN	93.3%
GCN (ours)	**97.5%**

Table 6: Comparison on the Configurators dataset of our approach based on STEP data to others state-of-the-art methods that works on different data formats.

The GCN training accuracy and loss for both of the datasets are shown in Figures 9 and 10, respectively.

4.3. GCN-based CAD model retrieval

The GCN network trained for the classification task was then used for the retrieval problem. A layer is chosen from the network, and a vector of features containing the model's salient features is extracted for each element of the dataset. The vectors extracted from

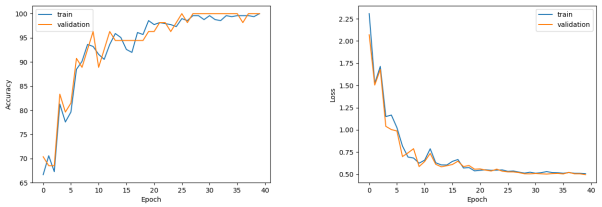

Figure 9: GCN training accuracy and loss obtained on the Traceparts CAD dataset.

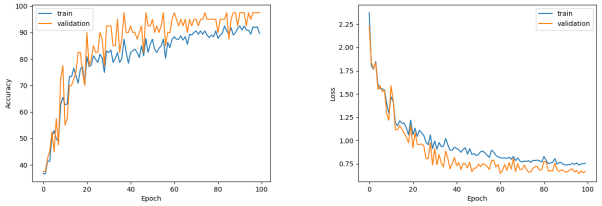

Figure 10: GCN training accuracy and loss obtained on the Configurators CAD dataset.

the test set are then compared with those extracted from the train set through various metrics such as cosine distance, Euclidean distance or histogram intersection. We expect that models of the same class will be represented by similar feature vectors. Then, by comparing the feature vectors, we should obtain a smaller distance if they belong to the same class and a greater one if they belong to different classes. We then use these distances to solve the retrieval task.

As a distances between extracted features, we compared the Euclidean distance, the cosine distance and the histogram intersection. The *mean average precision* (mAP) obtained for the first dataset through the metrics with the final softmax layer as output layer is shown in Table 7. The cosine distance resulted in the best score, so we used that distance in the experiments.

Metric	**mAP**
Euclidean distance	0.976
Cosine distance	**0.989**
Histogram intersection	0.976

Table 7: Comparison between different metrics when matching feature vectors extracted from the last FC layer of the GCN model for the first dataset.

We also studied which layer produces a more representative feature vector, as shown in Table 8. The last layer, *i.e.*, the softmax layer, performed the best mAP score equal to 0.989.

Based on the above tests, we fixed the cosine metric and the softmax layer as the output layer and generated the precision-recall curve shown in Figure 11. The method is able to obtain excellent

Output layer	mAP
Attention layer	0.872
First FC layer without ReLu	0.902
First FC layer with ReLu	0.919
Second FC layer	0.948
Softmax layer	**0.989**

Table 8: Comparison of mAP values with the cosine metric and different output layers for the generation of feature vectors for the first dataset.

values of mAP and only for class 2, characterized by the greater variance in the number of nodes of the graphs, it does not reach the perfect score.

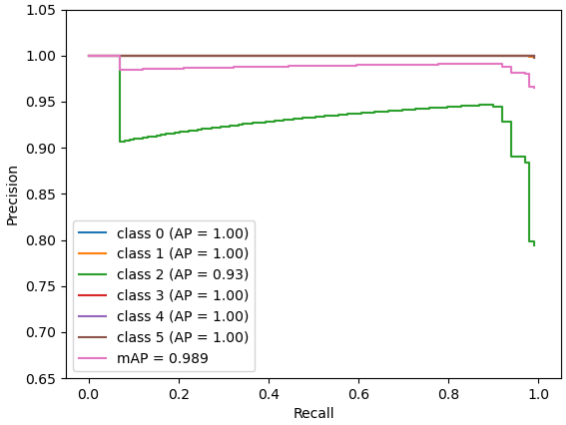

Figure 11: Precision-recall curve for the retrieval of CAD model with features vectors extracted from the softmax layer and cosine distance as metric for comparing features vectors of the first dataset.

4.4. Limitations

The large amount of information present in the CAD files involves some redundancy and therefore a large amount of memory is needed to store the dataset models as graphs. To avoid having to convert the models to graphs at each run, we used the *networkx* python library [swa] to save the graph data as a *.graphml* file. As shown in Table 9, the size of the storage files increases almost linearly as the sum of nodes and arcs increases.

This implies that for complex models corresponding to large graphs, heavy files are generated. In contrast to other formats such as Point Clouds, voxel grids and multiple 2D scans, in which the number of elements can be fixed as well as the size of the files, in the graph approach, it is not possible to regulate the size of the input data, therefore big graphs can be difficult to manage for GPU computation and RAM memory storage.

# nodes	# arcs	space
1K	1K	200 KB
3K	35K	600 KB
9K	10K	2,3 MB
15K	17K	3,5 MB
20K	25K	4,1 MB
30K	35K	7,0 MB
35K	40K	8,3 MB
40K	43K	11,0 MB

Table 9: Increase in size of graph storage *.graphml* files as the number of nodes and arcs increases.

5. Conclusions

In this paper, we have introduced a new approach to solve the retrieval and classification problems of 3D CAD data that operates directly in native format without the need to convert the models to other extensions. We exploited the linked structure of STEP files, which represents a standard format in the CAD domain, to create a graph in which the nodes are the primitive geometric elements and the arcs are the connections between them. The graphs created are then evaluated through convolutional graph neural networks.

Since there is no dataset containing a sufficient number of 3D models in a native CAD format, we created two datasets by downloading data from the *Traceparts* model library and by collecting 3D models from a real-world software modeling company and dividing them into classes. We have used these datasets to validate our approach and compare it with state-of-the-art methods that consider other 3D formats such as *PointNet++* for the point clouds and *MVCNN* for the models expressed as multiple 2D views. The obtained results demonstrate the validity of our method for both the classification task, where we obtained 100% and 97.5% accuracy for the relative datasets, and for the retrieval task, where we achieved results close to 100% of mAP for the first dataset. Our approach resulted in an equivalent or better performance with *MVCNN*, greatly exceeding the performance of *PointNet++* on the same data expressed in the respective formats.

This preliminary work has introduced a new approach for the analysis of CAD models, but its full potential is still to be discovered. In fact, the collected datasets did not allow us to exploit the information on the hierarchy of the components of the models and in the classification task the information of the nodes was represented only by their geometric type without considering the related parameters. Theoretically, using the hierarchy of components would allow us to improve the classification of complex objects, while introducing the parameters of the nodes would make it possible to distinguish even very similar objects. In future work, it will be critical to exploit these two characteristics in order to conduct experiments on larger datasets, allowing us to maximize the potential of classification and retrieval tasks of CAD files.

Acknowledgment

Removed for anonymous review.

References

[BDB*] BAI, YUNSHENG, DING, HAO, BIAN, SONG, et al. "SimGNN: A Neural Network Approach to Fast Graph Similarity Computation". () 5.

[CDT*03] CHEN, DING-YUN, TIAN, XIAO-PEI, et al. "On Visual Similarity Based 3D Model Retrieval". (2003) 2.

[CFG*] CHANG, ANGEL X., FUNKHOUSER, THOMAS, GUIBAS, LEONIDAS, et al. "ShapeNet: An Information-Rich 3D Model Repository". (). Technical Report arXiv:1512.03012 Stanford University - Princeton University - Toyota Technological Institute at Chicago 2015 6.

[GCD*21] GÜMELI, CAN, DAI, et al. "ROCA: Robust CAD Model Retrieval and Alignment from a Single Image". (2021) 2.

[Gro] GROUP, TRACE. *Traceparts online library, Product Content Everywhere*. Online available: https://www.traceparts.com/it. 2, 6.

[HSKK01] HILAGA, M., SHINAGAWA, Y., KOHMURA, T., and KUNII, T. L. "Topology matching for fully automatic similarity estimation of 3D shapes". *In SIGGRAPH* (2001), 203–212 2.

[Ip05] IP, CHEUK YIU. "Automatic Classification of CAD Models". (2005) 2.

[IR05] IP, CHEUK YIU and REGLI, WILLIAM C. "Content-Based Classification of CAD Models with Supervised Learning". *Computer Aided Design and Applications* (2005) 2.

[KAM*16] KANEZAKI, ASAKO, MATSUSHITA, YASUYUKI, et al. "RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints". (2016) 3.

[KRLV17] KLOKOV, ROMAN, LEMPITSKY, and VICTOR. "Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models". (2017) 2.

[LWM*20] LI, WENJIN, MAC, et al. "Computer aided design (CAD) model search and retrieval using frequency domain file conversion". (2020) 2.

[MS15] MATURANA, D. and SCHERER, S. "VoxNet: A 3D Convolutional Neural Network for real-time object recognition". *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015). DOI: 10.1109/IROS.2015.7353481 2.

[NK04] NOVOTNI, M. and KLEIN, R. "Shape retrieval using 3D zernike descriptors". (2004). DOI: 36(11):1047âĂŞ1062 2.

[NWSL20] NIE, W., WANG, Y., SONG, D., and LI, W. "3D Model Retrieval Based on a 3D Shape Knowledge Graph". (2020). DOI: 10.1109/ACCESS.2020.3013595 2.

[QFL*14] QIN, FW., LI, et al. "A deep learning approach to the classification of 3D CAD models". (2014). https://doi.org/10.1631/jzus.C1300185 2.

[QQGB22] QIN, FEIWEI, QI, SHI, GAO, SHUMING, and BAI, JING. "3D CAD model retrieval based on sketch and unsupervised variational autoencoder". (2022) 3.

[QRS*16] QI, R., CHARLES, SU, et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". (2016) 2.

[QYSG17] QI, CHARLES RUIZHONGTAI, YI, LI, SU, HAO, and GUIBAS, LEONIDAS J. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". *Advances in Neural Information Processing Systems*. Ed. by GUYON, I., LUXBURG, U. VON, BENGIO, S., et al. Vol. 30. Curran Associates, Inc., 2017 2, 7.

[RTBD02] R., OSADA, T., FUNKHOUSER, B., CHAZELLE, and D., DOBKIN. "Shape distributions". *ACM Transactions on Graphics* (2002) 2.

[SHM*15] SU, HANG, MAJI, et al. "Multi-view Convolutional Neural Networks for 3D Shape Recognition". (2015) 2, 7.

[SK06] S., HOU and K, RAMANI. "Sketch-based 3D engineering part class browsing and retrieval SBM". (2006) 3.

[SK07] S., HOU and K, RAMANI. "Classifier combination for sketch-based 3D part retrieval Comput. Graph". (2007) 3.

[SKT*17] SFIKAS, KONSTANTINOS, THEOHARIS, et al. "Exploiting the PANORAMA Representation for Convolutional Neural Network Classification and Retrieval". (2017). DOI: 10.2312/3dor.20171045 2.

[SPT18] SFIKAS, KONSTANTINOS, PRATIKAKIS, IOANNIS, and THEOHARIS, THEOHARIS. "Ensemble of PANORAMA-based convolutional neural networks for 3D model classification and retrieval". (2018) 2.

[Srl] SRL, CONFIGURATORI. *Configuratori*. website: https://www.configuratori.com/ 2, 6.

[swa] <SWART@LANL.GOV>, ARIC HAGBERG <HAGBERG@LANL.GOV> DAN SCHULT <DSCHULT@COLGATE.EDU> PIETER SWART. "NetworkX Python Library". (). https://github.com/networkx/networkx 9.

[WSK*15] WU, Z., SONG, S., KHOSLA, A., et al. "3D shapenets: A deep representation for volumetric shapes". *IEEE Conference on Computer Vision and Pattern Recognition* (2015), 1912–1920 6.

[ZLLG18] ZHI, SHUAIFENG, LIU, YONGXIANG, LI, XIANG, and GUO, YULAN. "Toward real-time 3D object recognition: A lightweight volumetric CNN framework using multitask learning". (2018) 2.