Artificial neural networks exploiting point cloud data for fragmented solid objects classification

IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 045025 https://doi.org/10.1088/2632-2153/ad035e

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
6 June 2023

REVISED
5 October 2023

ACCEPTED FOR PUBLICATION
13 October 2023

PUBLISHED
6 November 2023

PAPER

Artificial neural networks exploiting point cloud data for fragmented solid objects classification

A Baiocchi[2,*], S Giagu[1], C Napoli[2], M Serra[3], P Nardelli[2] and M Valleriani[2]

[1] Department of Physics, 'Sapienza' University, Rome, Italy
[2] Department of Computer, Control and Management Engineering, 'La Sapienza' University, Rome, Italy
[3] INFN Rome, Department of Physics 'Sapienza' University, Rome, Italy
[*] Author to whom any correspondence should be addressed.

E-mail: alessandro.baiocchi@uniroma1.it

Keywords: classification, fragments of artifacts, PointNet, DGCNN, graph neural network, solid objects dataset, point clouds

Abstract

This paper presents a novel approach for fragmented solid object classification exploiting neural networks based on point clouds. This work is the initial step of a project in collaboration with the Institution of 'Ente Parco Archeologico del Colosseo' in Rome, which aims to reconstruct ancient artifacts from their fragments. We built from scratch a synthetic dataset (DS) of fragments of different 3D objects including aging effects. We used this DS to train deep learning models for the task of classifying internal and external fragments. As model architectures, we adopted PointNet and dynamical graph convolutional neural network, which take as input a point cloud representing the spatial geometry of a fragment, and we optimized model performance by adding additional features sensitive to local geometry characteristics. We tested the approach by performing several experiments to check the robustness and generalization capabilities of the models. Finally, we test the models on a real case using a 3D scan of artifacts preserved in different museums, artificially fragmented, obtaining good performance.

1. Introduction

Restoration of archaeological artifacts broken in a large number of fragments is of high interest in archaeology. Manual restoration is a complex and time-consuming task, prone to errors in classifying the characteristics of the fragment surface and in the reassembling of objects from the fragments. The latter can, in fact, either be missing or deteriorated over time, and possibly not all coming from the same object. In recent years, several techniques based on modern Machine Learning methods have been proposed to assist archaeologists in these processes. In computer science, the 3D object classification and reassembly problem can be approached by means of point clouds of scanned 3D objects. A point cloud is an unstructured representation of 3D points in space sampled around the surface of the 3D object. The point cloud provides local and global information about the geometrical features of the object itself. Modern deep neural networks can leverage these features for classification, object detection, and segmentation tasks. In this work we explored the possibility of using two neural network architectures suitable for the point cloud data type, PointNet [1] and dynamical graph convolutional neural network (DGCNN) [2], that have been proved to be very successful in classifying and segmenting 3D objects. A Dynamical Graph CNN is a particular CNN that recomputes the graph using nearest neighbors in the feature space produced by each layer. The DGCNN exploits local geometric structures by applying a generalization of the convolution operation over a graph called EdgeConv operation.

The main contributions of this work are listed as follows:

(a) proposing the use of realistic artificial datasets (DSs), produced by a 3D object simulation, to train the neural network model;
(b) producing an open-access benchmark DS for further studies on the subject. This DS is an attempt to build a big archive of virtual archaeological fragments for the classification and reconstruction task;

(c) proposing a novel algorithm for classifying fragments of solid objects depending on global and local features.

One of the principal positive aspects of the provided benchmark DS is that it contains 3D solid objects. Thus, it does not limit the researcher to use a specific type of representation such as point clouds, but the objects can be represented in an arbitrary way. As local features for training the model, we propose the combined use of the coordinates of the point cloud, together with the normal vectors to the triangles of the mesh, and the areas of the triangles. The former acts as a proxy of surface gradients, while the latter acts as a proxy of the surface curvature. The fragments produced are divided into two categories: completely contained within the original object (tag: internal) or partially overlapping with the original external surface (tag: external). To the best of our knowledge, we are the first to address the task of classifying internal and external fragments, a task that would be of great help to archaeologists.

This paper is structured as follows. In section 2 we review the related works with emphasis on traditional and machine learning techniques applied for object classification and reconstruction in archaeology and cultural heritage, in section 3 we describe the benchmark DS 'Broken3D' we have produced. Data preprocessing and details of the PointNet and DGCNN models are given in sections 4 and 5 respectively. Finally, section 6 deals with experiments and results, and conclusions come in section 7.

2. Related works

With the development of technology, many researchers have been interested in finding possible solutions to reassemble objects with computer assistance. In particular, the classification and reconstruction of archaeological remains, starting from fragments, are two of the most studied and relevant tasks. The last decade has seen an evolution of the adopted techniques, from the analysis of 2D images to the study of 3D geometries, from the use of simple algorithms to the application of modern deep neural networks, and so on.

In the early 2000s, [3, 4] solved the classification and reconstruction problem by analysing 2D images. In particular, the first ones proposed a solution to the color classification of ceramic fragments assuming that the spectral reflectance of archaeological fragments varies slowly in the visible spectrum. Instead, the second ones presented a novel approach to the problem of puzzle solving related to the archaeological fragment reconstruction task. Smith *et al* [5] investigated the classification of thin-shell ceramics based on color and texture descriptors and formulated a new feature descriptor based on total variation geometry. Oxholm and Nishino [6] presented a method that reassembles objects using only the fragments' boundary contours. First, there is a preprocessing step to encode the scale variability of the boundary contours as multi-channel 2D images, then an identification step for matching sub-contours, and finally a least square formulation that minimizes the distance between the adjoining regions while simultaneously maximizing the resulting geometric continuity across them.

Over the years, the research has made considerable progress both in terms of acquisition devices (cameras, scanners, etc) and in terms of algorithms, and methods. Thus, for the task of recomposing 3D objects, the idea of working with 3D data ('geometry-based' approach) instead of 2D data ('texture-based' approach) has become increasingly predominant. Although the high complexity of 3D elements, the association of heterogeneous information to 3D data can help to better characterize the problem.

Both [7, 8] deal with the reconstruction of broken artifacts by using a 3D acquisition set-up, transforming the objects into point clouds without the aid of neural networks. In particular, the first work aims to find the individual rigid transformations that better align all the fragments. A pre-processing phase is applied to the point clouds obtaining key points and descriptors useful for a pair-wise search algorithm. Then, thanks to a many-to-many search, the original artifact is reconstructed. Instead, the second one starts from the acquisition of a fragment to the 3D reassembly of several fragments with an estimate of its likelihood. To do so, the scanned 3D data is segmented into facets of the lateral part of the fragments followed by a 3D matching.

Grilli and Remondino [9] explored the applicability of supervised machine learning approaches to cultural heritage proposing a reliable and efficient pipeline that can be standardized for different case studies. Starting from 3D point clouds they obtain UV maps which are segmented using machine learning algorithms. Then, they project the 2D classification results onto the 3D object space by back-projection and collinearity model. The reason for their research is the need to identify and map different states of conservation phases or employed materials in heritage objects. Gao and Geng [10] presented a method based on a deep learning network combined with template guidance to classify 3D fragments of the Terracotta Warriors. Initially, PointNet is used to classify, then misclassified fragments are secondly categorized based on their best match to a complete Terracotta Warrior model. Rasheed and Nordin [11] are focused on two tasks: classification of ancient fragments and reconstruction of ancient objects. For the first task, they use 2D

images exploiting the color and texture properties of the fragment surfaces in order to achieve the goal, while for the second one, they work with 3D objects by removing noise, extracting contours, and identifying the matching part using a neural network. Hu *et al* [12] presented a seed-region-growing CNN (SRG-Net) for unsupervised part segmentation with 3D point clouds of terracotta warriors. In the SRG algorithm, they estimate normal features to exploit the point cloud coordinates, and then they combine this algorithm with a custom CNN.

The kernel used in the convolution operator of traditional CNNs perfectly works for images and, in general, regular and ordered sets of 3D data. Instead, when we are dealing with point clouds, we have to modify the CNN architecture or we have to choose a different kind of network. This is due to the fact that a point cloud is an unstructured representation of 3D points in space, thus it is an irregular and unordered set of vectors. Therefore, in order to solve this problem, many options have been adopted.

Between 2017 and 2018, the literature was principally divided in two different directions to handle 3D geometry by using deep learning: on one side the exploration of neural networks that directly consume point clouds, while on the other side the development of Graph CNNs. In the first case, the turning point was the work done by [1], which proposed PointNet, a novel architecture suitable for utilizing unordered sets of 3D points. Masci *et al* [13] introduced geodesic convolutional neural networks, an extension of CNNs to non-Euclidean manifolds based on a local geodesic system of polar coordinates to extract 'patches'. Li *et al* [14] proposed PointCNN, which is a generalization of typical CNNs to feature learning from point clouds. The key idea of this method is theχ-transformation from the input points for both weighting and permutation. This transformation could potentially be more adaptive to local structures. Another framework for applying CNNs to point clouds is point convolutional neural network, presented by [15], whose main idea is to use extension and restriction operators for translating volumetric convolution to arbitrary point clouds. In the second case, instead, MoNet [16], ECC [17] and GAT [18] are some of the most famous and relevant works. In particular, MoNet is a general framework allowing to design of convolutional deep architectures on non-Euclidean domains. The novelty is the parametric construction for extracting 'patches', in which the convolution-like operation can be defined. They studied a family of functions represented as a mixture of Gaussian kernels. Another type of convolution-like operation is ECC (Edge Conditioned Convolution), which is generalized to arbitrary graphs. Filter weights are conditioned on the specific edge labels over local graph neighborhoods. Instead, graph attention networks (GAT) operates on graph-structured data, taking advantage of masked self-attentional layers. GAT can be seen as a particular instance of MoNet, but uses node features for similarity computations rather than the node's structural properties. This means that GAT does not assume the knowledge of the graph structure in advance.

As known in literature, the inclusion of additional features, besides the point cloud coordinates, often brings useful information that may lead to achieving better performances. In fact, in the work done by [19] a comparison between several deep neural networks for 3D data is shown and it demonstrates how these architectures outperform their standard versions when normal vectors are considered.

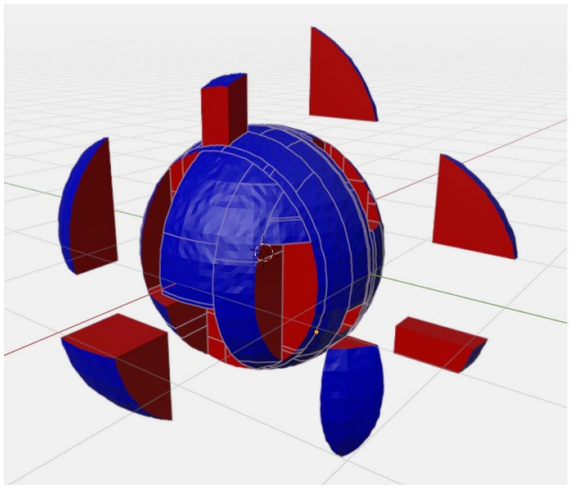
Another common usage of normal vectors is to add them to the input features, as done by [20, 21] and the already mentioned [12]. All these researches have in common the way in which the input is composed: the 3D point cloud coordinates plus the surface normal vectors. Thus, each point of the point clouds is composed of xyz coordinates, normal values, and possibly other features. In particular, [22] presented NormalNet, a voxel-based CNN designed for 3D object recognition. The network uses normal vectors of the object surfaces as input, which demonstrates stronger discrimination capability than binary voxels.

Then, both [23, 24] implemented a DGCNN approach adding meaningful information such as normal vectors and other features. More particularly, the latter ones calculated the normal vectors on cloud compare and gave in input a block composed of 12 features including normal vectors, which pass through the EdgeConv layers.

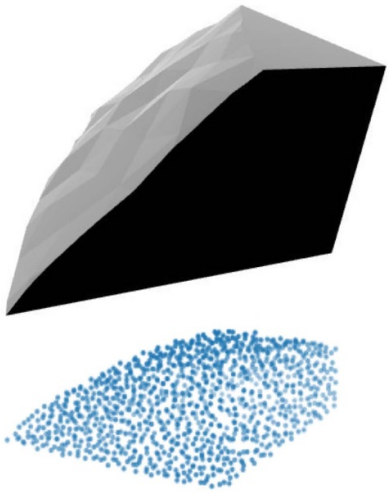
3. DS

We created a realistic simulated DS of fragmented objects in order to train and test the model for our task under different conditions. In fact, since the literature lacks any publicly available DS, we built one from scratch using the open-source 3D creation suite *Blender*[4]. Using the available Python APIs we implemented a procedure that first generates simple 3D objects like a cube, a sphere, an icosphere, or a torus and then subdivides them into fragments via a dedicated algorithm (which includes flat and deformed cutting surfaces on different axes). Initially, to build our DS we represented various kinds of objects and fragments by means of triangular mesh surfaces. Then, we implemented a procedure to randomly change the position of the

[4] www.blender.org/.



(a) Example of a fragmented sphere.



(b) Example of a fragment and the associated sampled Point Cloud.

Figure 1. A sample from the IGD dataset, on the left (a) a visualization of the fragmented object. On the right (b) a single fragment and the Point Cloud sampled on it.

mesh triangle vertexes (with adjustable parameters), to simulate the aging and accidental breakages that can occur in artifacts. At the end of the simulation, the fragments of the objects were stored in individual files in standard triangulation language (STL) format.

The format selection was determined by convenience in order to be easily converted in *Point Cloud* (PC) format. A PC is a collection of data points in space, and an object can be represented by a collection of sampled points around its surface. Such PC will be the input for our network models. We used a representation based on the *xyz* coordinates of the points.

In order to convert STL files to PC, we used the Open3D library [25] APIs. In this way, the fragments were sampled and represented as point cloud. While Open3D is computationally heavier than other packages, it outperforms other algorithms to produce samples almost homogeneous on the surface, so each point has approximately the same distance to the neighboring points. Moreover using Open3D it was possible to determine the number of points used to sample the surface. In our case, we used 1024 points, in figure 1(b) an example of a point cloud sampled from a fragment is shown.

Since we planned to classify different fragments by exploiting local geometrical structures, we used Open3D specific methods to compute other useful features associated with each point of the point cloud. In particular, Open3D allows us to get:

(i) all the vertices of each fragment;
(ii) the triangles of the mesh;
(iii) the normal vectors associated with the triangles.

So, in order to assign a vector representing the local curvature for each point of the point cloud we used the Open3D normals: we searched the nearest vertex of the nearest triangle to the point (i.e. the triangle which contains the nearest vertex to the point) and we collect from Open3D the normal to this triangle. Moreover, the area of the mesh triangle was computed. This parameter was then linked to each point. Therefore, we identified for each point the nearest triangle. The Area A_k of this triangle was computed by means of the Heron's Formula:

$$\begin{aligned} A_k &= \sqrt{s_k\left(s_k - a_k\right)\left(s_k - b_k\right)\left(s_k - c_k\right)} \\ K &= [1, N_P] \end{aligned} \tag{1}$$

where $s_k = \frac{a_k+b_k+c_k}{2}$ is the semi-perimeter, while a_k, b_k and c_k are the sides of the triangle associated to the kth point in a set of N_p points.

Therefore, in our DS we stored a point cloud for each fragment, and a vector and area pair related to each point.

The inclusion of normal vectors and areas as features is important for two main reasons: the normal vectors represent the orientation of the surface (see [26]) therefore they are useful to distinguish between a

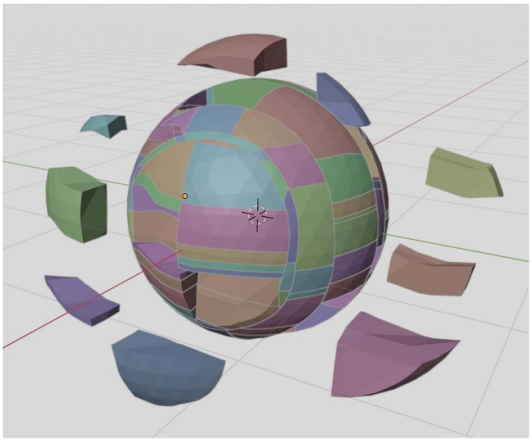


Figure 2. Fragments obtained from an icosphere using realistic cuts (RFD dataset). It can be noticed that cuts are roughly perpendicular to the cardinal axes.

flat and a curved surface and to better capture the different orientations of the surfaces of the fragments. On the other hand, areas are useful to distinguish between flat and corrugated surfaces, as flat surfaces are represented with a lower number of larger triangles. Corrugated surfaces contain instead a higher number of smaller triangles. Of course, the network's ability to use the vector and area features may depend on the specific shape of the various fragments and on the specific procedures used to simulate the aging and accidental breakages.

For the different DS productions, we always started from a group of objects of four types (cubes, spheres, torus, and icospheres), which were then broken into fragments in variable numbers and shapes depending on the positioning of the cutting surfaces. These surfaces have been designed to be flat or irregularly shaped, in different numbers (depending on the broken object), and to be positioned randomly or equidistantly along different axes. The fragments produced can be divided into two categories: either completely contained within the original object (tagged as *internal*) or partially overlapping with the original external surface (tagged as *external*). As the last step of the production, all the fragments of the various objects were grouped together and divided between external and internal to facilitate the preprocessing procedures for the assignment of labels. With our simulation, we tried to represent a realistic situation in which an artifact breaks and only a part of the fragments (externals) still retains traces of the visible part of the original sculpture.

3.1. DS description

Different DS have been produced by gradually representing a more realistic situation and introducing more complex elements (type of cutting surface, aging, etc) as detailed in the following.

The DS produced through our simulations - *Broken3D*—are publicly available[5].

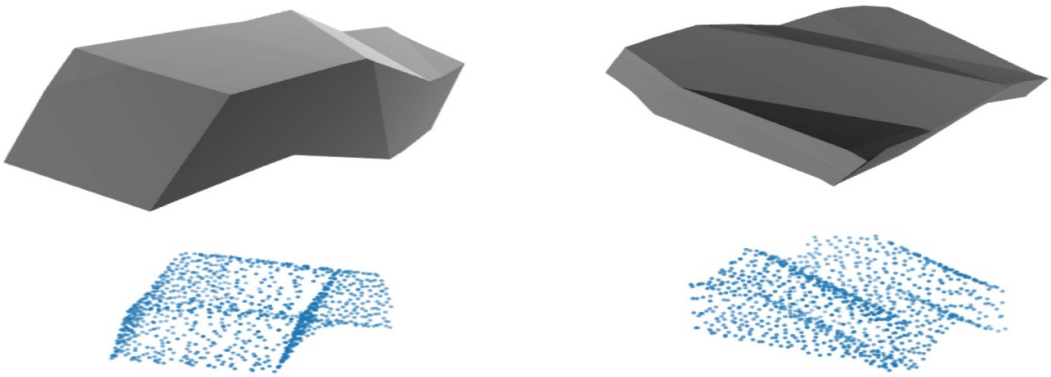
3.1.1. Ideal geometry dataset (IGD)

The IGD DS contains 12 670 STL files (stored in two different folders named *internal* and *external*), which contains the fragments corresponding to 36 solid objects (from three slightly uneven different geometrical shapes: cube, sphere, icosphere). In particular, 3223 are *internal* fragments while 9447 are *external*. In this case, the cutting surfaces were flat and positioned along the axes either equidistant (about half of the sample) or randomly. The number of cuts for each axis is chosen randomly in the range from 3 to 9. In figure 1 an example of a sample from this DS is shown.

3.1.2. Realistic fractures dataset (RFD)

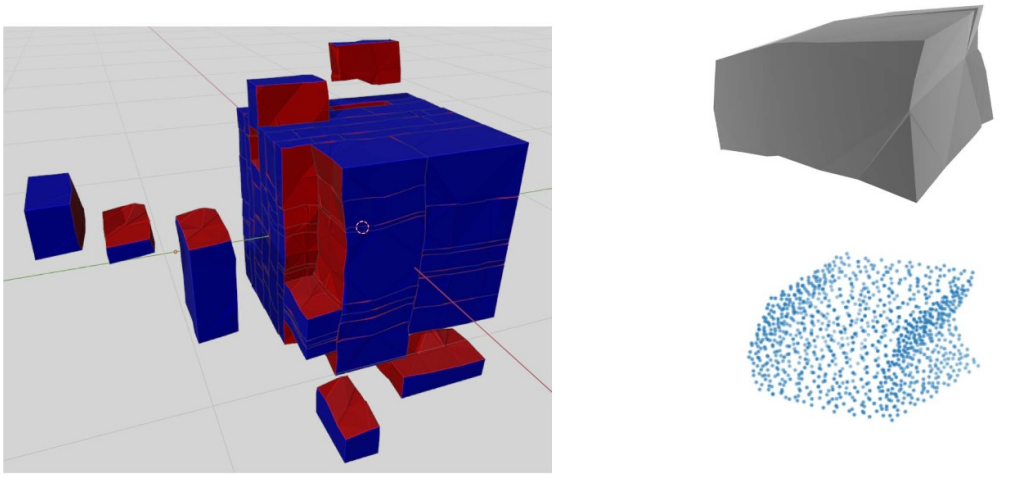
The RFD DS contains 9032 fragments (from 42 solid objects of 3 kinds—cube, sphere, and icosphere—with regular surface), 6371 of which are *externals* and 2661 *internals*. Also in this case the number of cutting surfaces on the axes ranged from 3 to 9 while their positions were always chosen randomly. The cutting surfaces were more realistic and irregular. Examples from this DS are shown in figures 2 and 3.

[5] https://deeplearninggate.roma1.infn.it/.



(a) External fragment. (b) Internal fragment

Figure 3. Visualization of an internal fragment (a) and an external fragment (b) from the RFD dataset. Here, the roughness of the cutting surfaces is apparent, producing more realistic fragments with respect to the IGD fragments.



(a) Fragmented cube belonging to the AED dataset. (b) Fragment and Point Cloud sampled on its surface.

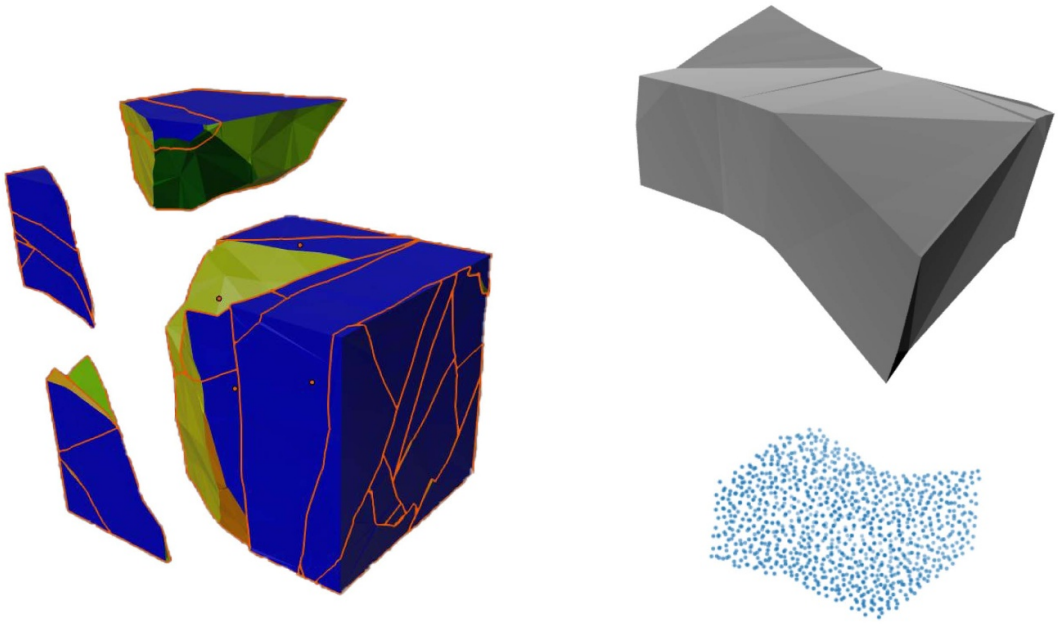
Figure 4. A sample from the AED dataset, on the left (a) a visualization of the fragmented object. On the right (b) a single fragment and the Point Cloud sampled on it.

3.1.3. Ageing effects dataset (AED)

The AED DS is made of 28 678 *internal* fragments and 86 835 *external* (from 565 solid objects of three kinds—cube, sphere, and torus), for a total of 115 513 STL files. The cutting surfaces were realistic, like the ones used for the second production. Moreover, a random noise was applied to the vertexes of the triangle meshes. The noise was introduced aiming to reproduce the wear due to time. Figure 4 shows some samples from this production.

3.1.4. Rotated cuts dataset (RCD)

The RCD DS is made of 8160 *internal* fragments and 83 081 *external* (from 2180 solid objects of three kinds—cube, sphere, and torus), for a total of 91 241 STL files. The cutting surfaces in this DS are not perpendicular to the axes of the reference system of the solid objects and have instead random orientations. The irregularity of the cutting surfaces and the noise applied after the cuts are equivalent to those used in the AED production. A sample from this DS is shown in figure 5.



(a) Fragmented cube belonging to the RCD dataset. It can be seen that the cuts are no longer along cardinal axes but have random orientations.

(b) Fragment and Point Cloud sampled on its surface.

Figure 5. A sample from the RCD dataset, on the left (a) a visualization of the fragmented object. On the right (b) a single fragment and the Point Cloud sampled on it.

3.1.5. Real statues models DS

While creating the framework that allowed us to produce the fragments we kept open the possibility to import solids from an external source. In particular for the experiments in this work, In order to enable us for a common benchmarking ground, we also imported the publicly available 3D scans of some real artifacts, such as the *Caligola's Bust*[6] and the *Nefertiti's Bust*[7].

This latter is a scan of an artifact of fundamental importance preserved in the Neues Museum of Berlin (Germany): an old Egyptian bust of Nefertiti. Applying the fragmentation procedure already used for the RCD dataset we have obtained external and internal fragments, as shown in figure 6.

4. Preprocessing

As described in the previous section the fragmentation procedure produces imbalanced DSs with respect to *internal* and *external* fragments. To cope with the class imbalance in the DSs we used either a weighted loss function or undersampling. The weighted loss was used in the training and the model performances have been evaluated by means of both a balanced accuracy and macro f1 score. Otherwise given the abundance of data in the last two DSs (AED and RCD), balance was obtained by subsampling the external fragments.

The DSs have been split into three parts: train, validation, and test set, in the proportions of 70%:15%:15% respectively.

We applied normalization on all the point clouds and on all the areas. For each of these inputs, we first saved the minimum and maximum values. Finally, we normalized the features within the [0,1] range using the MinMaxScaler algorithm.

Additionally, at every epoch start the DS is randomly shuffled.

In some experiments, additional data augmentation was applied in the form of random rotations of each sample (on the three rotation axes). This choice of data augmentation is aimed at teaching the model the rotational invariance of the fragments.

[6] www.thingiverse.com/thing:239422.

[7] www.thingiverse.com/thing:3974391.

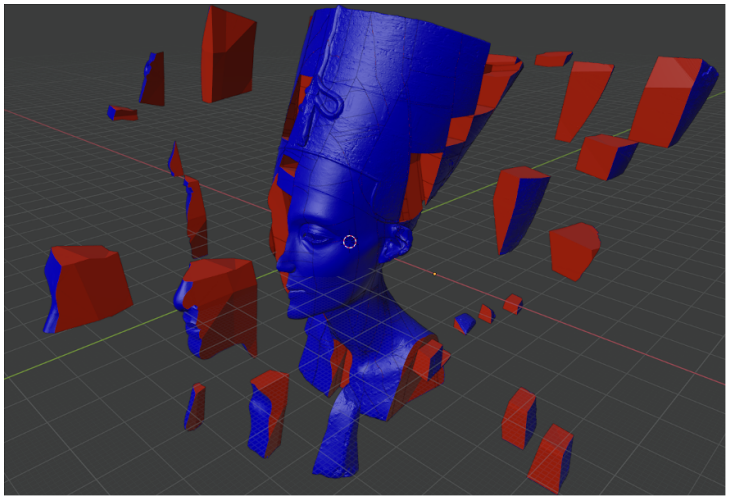


Figure 6. An example of a statue belonging to the statue models dataset. The bust of Nefertiti shown in the picture is fragmented in the same way the objects in the RCD dataset are.

5. Models

The models we employed are inspired by two neural network architectures that have gained significant attention in the field of 3D object recognition: the DGCNN [2], and PointNet [1]. To enhance the performance of our models for our particular task, we made some modifications to the original DGCNN and PointNet models during implementation. Specifically, we incorporated additional features to the traditional input point cloud coordinates, including normal vectors associated with the triangles in the STL representation of the 3D fragments, as well as the area of these triangles. These enhancements were designed to assist the models in learning the local curvature of the 3D object.

5.1. DGCNN

DGCNN [2] is a deep learning model that has shown remarkable performance in 3D object recognition and classification tasks. DGCNN is a graph convolutional neural network that operates directly on point cloud data, which is a common representation of 3D shapes. Unlike traditional CNNs, DGCNN considers each point in a point cloud as a node in a graph, where the edges represent the spatial relationships between points. This enables the model to capture local and global geometric features of the object, making it highly effective for 3D classification tasks. DGCNN comprises multiple graph convolutional layers that learn local features by aggregating information from neighboring points. The model then employs a max pooling operation to obtain a global feature representation of the entire point cloud. DGCNN also includes a dynamic edge convolution layer, which dynamically adjusts the edge weights based on the distance between points, allowing the model to capture the local geometry of the object better.

The DGCNN architecture consists of several graph convolutional layers, which allow the model to learn local features by aggregating information from neighboring points in the point cloud. The graph convolutional layer is defined as follows:

(i) Given a set of points represented as a point cloud, the graph convolutional layer computes the pairwise distances between all pairs of points using Euclidean distance;
(ii) The pairwise distances are then used to construct a k-nearest neighbor graph, where each point is connected to its k-nearest neighbors;
(iii) The graph is then used to compute the edge features, which represent the spatial relationship between pairs of points. The edge features are computed using a dynamic edge convolution layer that adjusts the edge weights based on the distance between points;
(iv) Finally the edge features are combined with the point features using a learnable weight matrix to produce the updated point features, called edgconv.

After several graph convolutional layers, the DGCNN employs a max pooling operation to obtain a global feature representation of the entire point cloud. The global feature representation is then fed into a fully connected layer, which produces the final output of the network. Overall, the structure of DGCNN

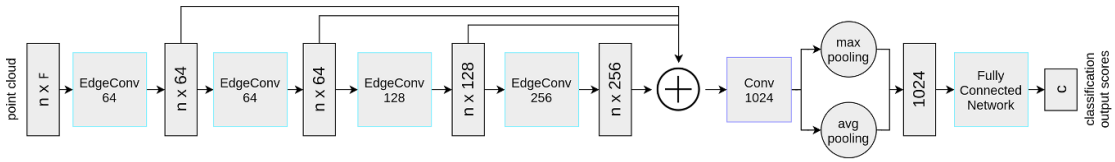


Figure 7. The DGCNN model architecture. A series of EdgeConv layers are stacked at the beginning of the model, then the extracted features get processed by a dense head.

enables it to effectively learn local and global geometric features of 3D shapes, making it a powerful tool for tasks such as 3D object recognition and classification.

5.1.1. DGCNN architecture

As shown in figure 7 our DGCNN model architecture is composed by:

- four EdgeConv layers;
- one convolutional layer;
- features concatenation;
- fully connected network.

Thus, a point cloud is fed as input into the first EdgeConv layer. An EdgeConv layer first processes the point cloud and extracts its corresponding graph features, then the graph features are used by a convolutional layer in order to compute the edge features. As the last step of the EdgeConv, the max operator is applied to the edge features obtaining the output of the edge convolution operator. This output is the input of the next EdgeConv layer. This means that the graph features are recomputed starting from of the previous output, by applying the dynamic graph update technique. These steps are iterated for a total of four EdgeConv layers whose outputs are concatenated and processed by a convolutional layer. To the output of each convolutional layer, the max pooling and average pooling operators are applied. Finally, the proposed system concatenates the obtained vectors with the normal vectors and the areas associated with the point cloud.

As the last step, a fully connected network is applied in order to obtain the classification output scores. This fully connected network is composed of three modules and each of them is composed of:

(1) a *linear layer*, which applies a linear transformation to the incoming data;
(2) a *batch normalization* layer, in order to increase the stability of the NN which normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation;
(3) a *leaky ReLU* activation.

5.2. PointNet

As a second model we implemented the original PointNet classification model proposed in [1]. PointNet is a unique neural network architecture that can directly consume raw point clouds as input. This makes PointNet highly efficient and scalable, as it does not require the costly pre-processing steps associated with other approaches. PointNet consists of a series of multi-layer perceptrons (MLPs) that process each point independently, followed by a max-pooling operation to aggregate global features. These features are then fed into a fully connected layer, which produces the final output of the network. The main advantage of this architecture is that it is invariant to the input point order, allowing it to capture the intrinsic features of the object regardless of its orientation or position. To achieve this invariance, PointNet includes a max pooling operation that aggregates the feature vectors of all the points in the point cloud, which produces a global feature vector that is permutation invariant. In addition, PointNet includes a transformation network that learns to estimate an affine transformation matrix for the input point cloud, which allows the model to be more robust to input variations such as rotation and translation. The PointNet transformation network is used to learn a spatial transformation that can be applied to the input point cloud data to improve its alignment and orientation in space. The transformation network takes as input a point cloud represented by a set of N points, each with its own feature vector, and outputs a 3×3 rotation matrix R and a three-dimensional translation vector t. The transformation network is implemented as a sequence of several fully connected layers with ReLU activation functions. The input to the transformation network is a concatenation of the point cloud features with an additional spatial coordinate vector that encodes the position of each point relative to the centroid of the point cloud. This allows the network to learn a transformation that is invariant to translation and rotation. Once the transformation matrix is learned, it can

be applied to the original point cloud data to align it in a canonical orientation for downstream processing tasks, such as segmentation or classification. This alignment can help improve the accuracy of subsequent processing steps by reducing the effects of spatial variability in the input data. Overall, the structure of PointNet enables it to effectively learn features of 3D shapes directly from the raw point cloud data, making it well-suited for our classification task

5.3. Weighted loss

As seen in section 3.1, we have an imbalanced DS and so we have to take into account this issue. A standard technique to avoid this problem—when classifying two classes—is to undersample the most populous data. A more general option is the choice of a weighted loss function, that is a loss function that explicitly weights penalties for two types of errors (i.e. false negative and false positive errors) in a binary classification problem [27]. We essentially want to assign a lower weight to the loss encountered by the samples associated with the majority class, that is the class of the *external* fragments. In this way, we give more relevance to the minority class. In our models, we used a weighted cross-entropy loss, with as weight of each class as the size of the smallest class divided by the size of that class. Thus, the weights for the *internal* and the *external* classes are $w_{\text{int}} = 1$ and $w_{\text{ext}} = n_{\text{int}}/n_{\text{ext}}$ respectively, where n_{int} is the number of internal fragments, while n_{ext} is the number of external fragments.

6. Experiments and results

In this section, we describe the experiments as well as the procedure to train our models.

A first set of tests was carried out with the Pointnet model, to show its effectiveness as a classifier. Later, other tests were carried out with the DGCNN model to verify whether using the pointcloud local information would make it possible to improve the previous results.

As explained in section 3.1, we devised our model in order to work with an imbalanced DS (when undersampling was not used), therefore in order to show the effectiveness of the adopted technique we used two kinds of classification metrics: the balanced accuracy and the macro F1 score. The first one is defined as the average of recall obtained on each class[8], while the second one calculates metrics for each label and finds their unweighted mean[9]. Thus, macro F1 is defined as follows:

$$\text{macro F1} = \frac{1}{C}\sum_{i=0}^{C} \text{F1}_i \tag{2}$$

where

$$\text{F1}_i = 2 \cdot \frac{\text{precision}_i \cdot \text{recall}_i}{\text{precision}_i + \text{recall}_i}. \tag{3}$$

In the different experiments explained below, we trained the model between 20 and 160 epochs, saving the model when and if the performances were increasing between an epoch and the previous ones. The batch size was equal to 32. Each input element is composed of 1024 3D points.

For the PointNet model, we used the parameters described before, which correspond to the ones used in the original implementation [1]. The only variation is the modification of the regularization loss parameter α (with weight 0.0001) to use more features than the coordinates of the points (see previous paragraph).

For the DGCNN the number of neighbors on which the edge convolution is computed has been set to 5, 10, 20, 30. The selected optimizer was ADAM, with a learning rate of 0.001, momentum of 0.9, and weight decay of 0.0001. In addition to the weight decay, another regularization technique employed is dropout. In particular, we randomly dropped 20% of the neural units. In table 1, the hyperparameters obtained for the best model are reported.

6.1. From baseline to best model

For the models that we analyze in this section, we show the results in:

- table 2 which illustrates the performances, that is the balanced accuracy (*bal acc*) and the macro F1 score (*macro-F1*);
- table 3 which displays the differences between a model and the previous one, that is the variations of the performance values.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html.
[9] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

Table 1. Best model hyperparameters. These were obtained through a grid search of hyperparameters. The values taken into consideration for the grid search were chosen close to the hyperparameters used in the original paper of each model.

Model	Hyperparameter	Value
PointNet	batch size	32
	learning rate	1×10^{-3}
	dropout	0.3
	α	1×10^{-4}
DGCNN	batch size	32
	learning rate	1×10^{-3}
	dropout	0.5
	k	20

Table 2. PointNet experiments results. The experiments listed differ in the input features that were given to the model.

Model	bal acc	macro-f1
M_{baseline}	0.8127	0.7845
M_{normals}	0.8583	0.8326
M_{areas}	0.8183	0.7832
$M_{\text{normals\_areas}}$	0.8341	0.8278

Table 3. Differences (differences in percentage) between the several models and the baseline model, showing the increase in performances.

Model	Δ_{avg_acc}	Δ_{macro_F1}
M_{baseline}	—	—
M_{normals}	$+0.0456\,(+5.6\%)$	$+0.0481\,(+6.1\%)$
M_{areas}	$+0.0056\,(+0.7\%)$	$-0.0013\,(-0.2\%)$
$M_{\text{normals\_areas}}$	$+0.0214\,(+2.6\%)$	$+0.0433\,(+5.5\%)$

We refer always to these tables during the treatment of this section.

Most of the experiments on AED were performed using PointNet. Other experiments were performed using both Pointnet and DGCNN or only DGCNN. For each experiment, results are shown for all of the architectures used.

6.1.1. Trial 1—$M_{baseline}$

The AED DS was used in this first batch of experiments. In this trial, we train PointNet using only the point cloud coordinates. We obtained a macro F1 score of 0.78, while the balanced accuracy was 0.81. This is a very promising but not optimal result as can be seen also in the confusion matrix in figure 8(a). The accuracy for identification of internal and external fragments is 83%, and 78% respectively.

6.1.2. Trial 2—$M_{normals}$

In this experiment, we extended the approach including the normal vectors. They represent the orientation of the fragment's surfaces. Leveraging the additional information, the model performances have significantly increased by 5.6% and 6.1% respectively for macro F1 score and balanced accuracy with respect to the baseline model. These results are also shown in the confusion matrix in figure 8(b).

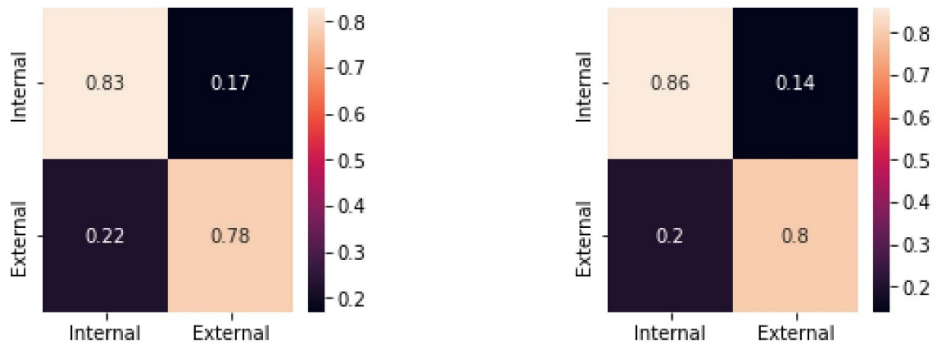
6.1.3. Trial 3—M_{areas}

We also explored the possibility of including the areas of the mesh triangles as additional information to better classify the fragments. The results of this experiment show a similar performance to the baseline. The relative difference in the evaluation metrics is indeed smaller than 1%.

6.1.4. Trial 4—$M_{normals\_areas}$

Finally, we trained the models using both the areas of the mesh triangles and the normal vectors. This model obtained an accuracy score of 0.834 and an F1 score of 0.827, which represents an improvement over the baseline model. However, the performances decrease with respect to the model using exclusively the normals.

Since adding the area as a feature brought no significant improvement in the performance of the model we performed all subsequent experiments using only coordinates and normals.



(a) Confusion matrix of the model $M_{baseline}$ (AED dataset).

(b) Confusion matrix of the best model $M_{normals}$ (AED dataset).

Figure 8. The matrix (a) on the left shows that the model is able to achieve promising results based on the coordinates of the points alone. The matrix (b) on the right, on the other hand, shows that adding the normals as input features improves the performance of the model even further.

6.2. Robustness and generalization tests

In order to test the network robustness and generalization capabilities we performed additional sets of trials.

Initially, the capability of the model to learn from different DSs was studied. The model was trained and evaluated on both simpler (RFD) and more realistic (RCD) DSs.

Moreover, we investigated the capability of the model to generalize to rotated data. We observed that the models originally did not learn rotation-invariant features, so we decided to retrain them using random rotations as data augmentation during training.

Lastly, in order to measure the ability of the model to generalize to more complex fragments, an evaluation was carried out using the data from the statue model DS. This last evaluation was made using only the DGCNN because of the slightly superior performance with respect to the PointNet in the previous tests.

6.2.1. Experiments on different DSs

We trained and tested the DGCNN model on the RFD and RCD DSs (described in section 3.1) with the aim of understanding if the model was able to be successfully applied to data with different levels of realism.

The performances obtained are reported in figures 9(a) and (b). For the RFD DS, no significant increase or decrease in the overall accuracy has been observed with respect to the best model. It is expected that the RFD DS would not be too challenging for the network to generalize to. In fact, the smooth external surfaces should make for an easier task.

For the RCD DS, which instead contains more realistic fragments, we observed a reduction of 11.8% in the average accuracy of the model. This is to be expected since more realistic fragments translate to a more difficult task for the network.

Despite achieving a worse performance with respect to the AED DS, we value the realisticity of the RCD DS, therefore the subsequent models are trained using the RCD data.

It is worth noticing here that we followed a conservative approach for the hyperparameters optimization, that have been tuned on the AED DS. Such parameters may result in being sub-optimal for the other DSs.

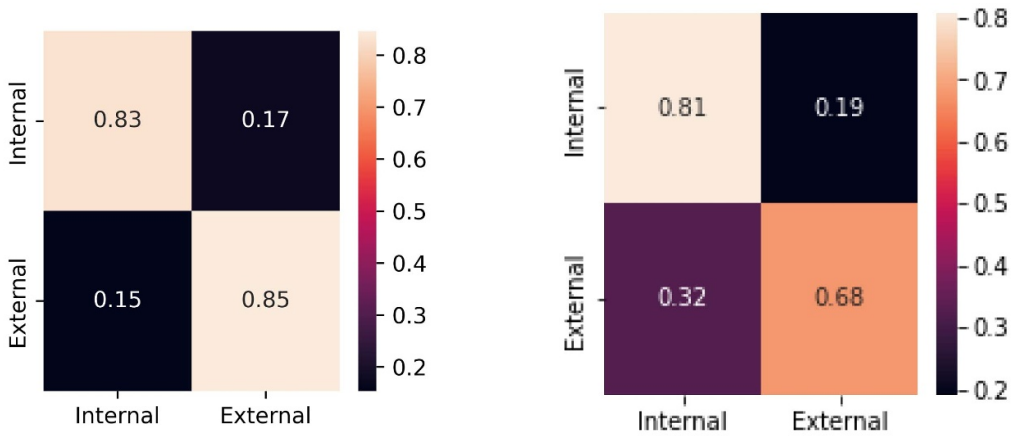
6.2.2. Rotational invariance

The models we employed in this study are not rotation-invariant. Therefore we studied the effect of random rotation applied to the test data. All of the subsequent experiments were carried out using the most realistic DS, RCD.

By applying random rotations on the test data the performances drop drastically, as shown in table 4. This shows that the features learned by the model are dependent on the orientation of the fragments, and as such are not desirable in a rotation-invariant task. By augmenting the data with randomly rotated samples we recover a model whose performances on rotated data are comparable to the ones obtained previously on the original test set (3rd row of table 4).

6.2.3. Test on the statue models DS

As a final test, we evaluated our model trained on the RCD DS by testing it on the real statues models DS (3.1.5). The confusion matrix is shown in figure 11. We obtained very good results even in this situation, consistently with the result obtained with the RCD DS (see figure 9(b)). The average accuracy was 0.81 and



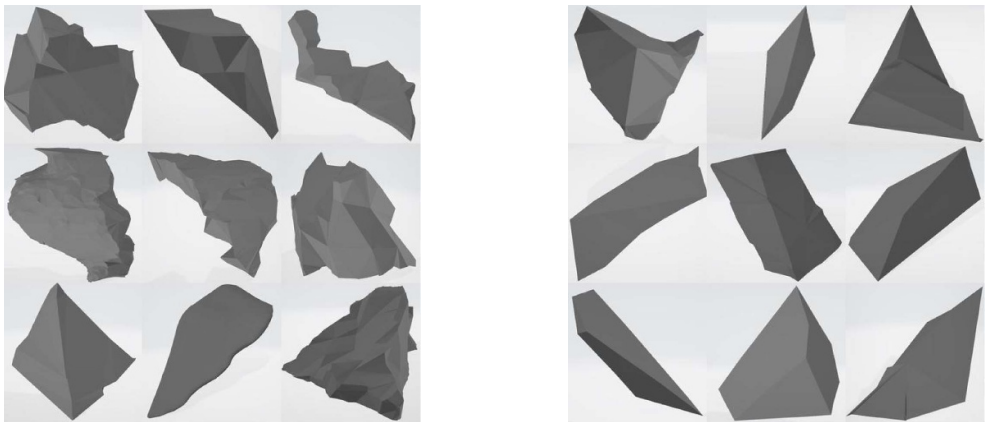
(a) Confusion matrix of the DGCNN on the RFD dataset.

(b) Confusion matrix of the DGCNN on the RCD dataset.

Figure 9. These matrices show how different levels of realism affect the capability of the network to solve the task effectively. On less realistic datasets (RFD) the network has even better performance than on the AED dataset, while on more realistic datasets (RCD) the model's performance decreases.

Table 4. Accuracy and F1 score for the DGCNN trained and evaluated on the RCD dataset with and without rotation data augmentation.

Train data	Test data	Accuracy	F1 score
Original data	Original data	0.7671	0.7646
Original data	Rotated data	0.6611	0.6492
Rotated data	Rotated data	0.7543	0.7632



(a) Examples of fragments classified as external by the model.

(b) Examples of fragments classified as internal by the model.

Figure 10. The fragments shown in these pictures are classified respectively as external (on the left) and internal (on the right). It can be seen that some, but not all, external pieces are easily recognized since they show very smooth and detailed surfaces.

the F1-score was 0.81. This is an already satisfactory result that could be improved by fine-tuning the trained model over realistic examples of real archaeological fragments. The slight performance improvement is due to the fact that the external surfaces of the statue models are much smoother than the internal cuts, and, as such, easier to distinguish. Some examples of fragments classified as external and internal are shown respectively in figures 10(a) and (b).

6.3. Human benchmark

State-of-the-art reconstruction for this kind of problem is usually done by human operators that manually identify fragments. In order to benchmark our approach to this problem we decided to perform some tests

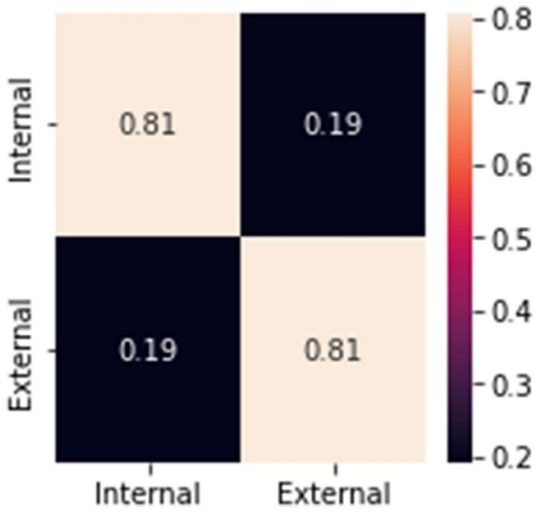


Figure 11. Confusion matrix obtained by testing the model M_{best}, trained on the RCD dataset, on the real statues dataset.

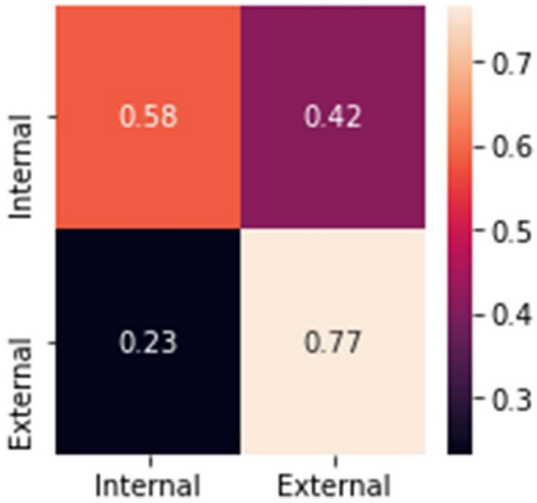


Figure 12. Confusion matrix obtained by the human subjects on our benchmark tests. The fragments used for this test are a balanced subset of 100 samples from the statue dataset. It can be seen that humans struggle to recognize the internal fragments and have significantly lower overall performance with respect to the model.

with humans: a small subset of the statue models DS was used for these tests. We asked several people to classify the 100 fragments contained in the DS, obtaining an approximate benchmark for human accuracy on this kind of problem. We must remember that our test subjects are not archeologists and only received a brief explanation, including a few examples before taking the test.

The results of this test are shown in figure 12. The overall accuracy for the human subjects was 67%, significantly lower than the accuracy obtained by the model on the same data. The results of this test also show that humans have more difficulty in recognizing internal fragments for this DS.

6.4. Further developments

Even though the approach we presented shows promising results in the classification of internal and external fragments, it is limited to processing each fragment separately. For the complete reconstruction of an artifact, it would be necessary to adapt the model to be capable of processing more than one fragment at a time to predict which fragments are adjacent. This could be possible by exploiting a twinned version of the DGCNN backbone to extract each fragment's features and then by aggregating them in a permutation-invariant way to predict whether the two fragments are neighbors.

7. Conclusion

This research paper presents a novel approach for fragmented solid object classification based on a DGCNN.

We built a synthetic DS of fragments of different 3D objects from scratch, including aging effects. We used this DS to train a deep learning model to classify internal and external fragments.

We tested the approach by performing several experiments to check the robustness and generalization capabilities of the model. Finally, we tested the model on a real case, using a 3D scan of the Nefertiti Bust from the Neues Museum, artificially fragmented, obtaining promising performance.

The codes to load the data and build, train, and evaluate the models, as well as pre-trained weights are available on request by contacting the authors.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://deeplearninggate.roma1.infn.it/.

Conflict of interest

The authors declare that they have no competing interests.

ORCID iDs

A Baiocchi https://orcid.org/0000-0002-0596-6707
S Giagu https://orcid.org/0000-0001-9192-3537
M Serra https://orcid.org/0000-0002-6093-8063
P Nardelli https://orcid.org/0000-0002-9093-1532

References

[1] Charles R Q, Su H, Kaichun M and Guibas L J 2017 PointNet: deep learning on point sets for 3D classification and segmentation *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* pp 77–85
[2] Wang Y *et al* 2019 Dynamic graph CNN for learning on point clouds *ACM Trans. Graph.* **38** 1–12
[3] Kampel M and Sablatnig R 2000 Color classification of archaeological fragments *Proc. 15th Int. Conf. on Pattern Recognition* vol 4 pp 771–4
[4] McBride J C and Kimia B B 2003 Archaeological fragment reconstruction using curve-matching *Conf. on Computer Vision and Pattern Recognition Workshop* vol 1 p 3
[5] Smith P, Bespalov D, Shokoufandeh A and Jeppson P 2010 Classification of archaeological ceramic fragments using texture and color descriptors *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition—Workshops* pp 49–54
[6] Oxholm G and Nishino K 2013 A flexible approach to reassembling thin artifacts of unknown geometry *J. Cult. Herit.* **14** 51–61
[7] Sanchez-Belenguer C and Vendrell-Vidal E 2014 An efficient technique to recompose archaeological artifacts from fragments *Int. Conf. on Virtual Systems & Multimedia (VSMM)* (IEEE) pp 337–44
[8] Jampy F, Hostein A, Fauvet E, Laligant O and Truchetet F 2015 3D puzzle reconstruction for archeological fragments *Proc. SPIE* **9393** 56–64
[9] Grilli E and Remondino F 2019 Classification of 3D digital heritage *J. Remote Sens.* **11** 847
[10] Gao H and Geng G 2020 Classification of 3D terracotta warrior fragments based on deep learning and template guidance *IEEE Access* **8** 4086–98
[11] Rasheed N A and Nordin M J 2020 Classification and reconstruction algorithms for the archaeological fragments *J. King Saud Univ.* **32** 883–94
[12] Hu Y *et al* 2020 SRG-Net: unsupervised segmentation for terracotta warrior point cloud with 3D pointwise CNN methods (arXiv:2012.00433)
[13] Masci J, Boscaini D, Bronstein M M and Vandergheynst P 2018 Geodesic convolutional neural networks on Riemannian manifolds (arXiv:1501.06297)
[14] Li Y *et al* 2018 PointCNN: convolution On X-transformed points (arXiv:1801.07791)
[15] Atzmon M, Maron H and Lipman Y 2018 Point convolutional neural networks by extension operators (arXiv:1803.10091)
[16] Monti F *et al* 2017 Geometric deep learning on graphs and manifolds using mixture model CNNs *Conf. on Computer Vision and Pattern Recognition (CVPR)* (IEEE) pp 5425–34
[17] Simonovsky M and Komodakis N 2017 Dynamic edge-conditioned filters in convolutional neural networks on graphs (arXiv:1704.02901)
[18] Veličković P *et al* 2018 Graph attention networks (arXiv:1710.10903)
[19] Uy M A *et al* 2019 Revisiting point cloud classification: a new benchmark dataset and classification model on real-world data (arXiv:1908.04616)
[20] Lan S, Yu R, Yu G and Davis L S 2018 Modeling local geometric structure of 3D point clouds using Geo-CNN (arXiv:1811.07782)
[21] Zhao H, Jiang L, Fu C and Jia J 2019 PointWeb: enhancing local neighborhood features for point cloud processing *Conf. on Computer Vision and Pattern Recognition (CVPR)* (IEEE) pp 5560–8
[22] Wang C , Cheng M, Sohel F, Bennamoun M and Li J 2019 Normalnet: a voxel-based cnn for 3d object classification and retrieval *Neurocomputing* **323** 139–47

[23] Widyaningrum E, Fajari M K, Lindenbergh R C and Hahn M 2020 Tailored features for semantic segmentation with a DGCNN using free training samples of a colored airborne point cloud *The Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLIII-B2-2020* pp 339–46
[24] Pierdicca R , Paolanti M, Matrone F, Martini M, Morbidoni C, Malinverni E S, Frontoni E and Lingua A M 2020 Point cloud semantic segmentation using a deep learning framework for cultural heritage *Remote Sens.* **12** 1005
[25] Zhou Q, Park J and Koltun V 2018 Open3d: a modern library for 3d data processing
[26] Cantor D and Jones B 2012 *Webgl: Beginner's guide; become a master of 3D web Programming in Webgl and Javascript* 1st edn (Packt Publishing)
[27] Phan H, Krawczyk-Becker M, Gerkmann T and Mertins A 2017 DNN and CNN with weighted and multi-task loss functions for audio event detection (arXiv:1708.03211 [cs.SD])