Dynamical Graph Convultional Neural Network for fragmented solid objects classification

MARTINA VALLERIANI* and PIETRO NARDELLI*, La Sapienza University of Rome, Italy

This paper presents a novel approach for fragmented solid object classification based on a Dynamical Graph Convolutional Neural Network (DGCNN). This work is the initial step of a project in collaboration with the Institution of "Ente Parco Archeologico dei Fori Imperiali" which has the aim to reconstruct ancient artifacts. In particular, our dataset is built from scratch by Marco Serra and is composed of fragments of three different 3D objects: cubes, spheres and icospheres. Our task is to classify internal and external fragments: the internal ones have only internal faces while the external ones have at least an external face. In order to achieve this goal, we have implemented a DGCNN that takes as input the point clouds and uses also additional features. Point clouds are useful for the network because it needs the sampled representation of the fragments in order to work with solid objects. The innovative idea is the concatenation of the normal vectors (and of the areas) associated to the point clouds in the middle of the architecture. We prove that this information is very relevant for this task, indeed our model can gain very high performances. Moreover, we make several experiments to check the robustness of our model showing that the network is able to correctly perform also in presence of input noise (jitter) on the test set and in the event of different solid objects.

1 INTRODUCTION

One of the most important aspects of our life is the knowledge of our history, and thus the reconstruction of ancient artifacts is a very fascinating and challenging task. Indeed, in computer science the classification and recomposition of 3D solid objects starting from fragments is a problem which has high room for improvement. In fact, it is necessary to explore novel techniques in order to increase the performances.

Our project is a module of a bigger one, in collaboration with the Institution of "Ente Parco Archeologico dei Fori Imperiali" [1]. The bigger problem has the task of reconstructing the broken archeological remains of the Roman Forum starting from fragments. Since this is a very complex and articulated work, it is split into multiple sub-problems. In particular, our project has the aim of classifying internal and external fragments. With these expressions we mean that, when a fragment contains at least an external surface of the original object, it is classified as "external", while when there is no external surface, it is classified as "internal".

In order to solve our task, we need a specific and appropriate dataset, containing sufficient fragments of archaeological finds. Unfortunately, to the best of our knowledge, at the moment there is no available dataset of this kind and so the first step is the creation of a synthetic and simpler dataset. Starting from simple 3D objects, which are the sphere, the icosphere and the cube, some cuts are applied in order to get the fragments (see figure 1). In section 3 we will explain in detail how it was built.

Usually the 3D objects classification (and recomposition) problem is solved 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. Other than the point clouds, from the 3D objects we also extract other information such as the vector of the normals and the areas of the mesh triangles. However, the pre-processing techniques are applied only to the point clouds, which are the input of the network. As pre-processing techniques we use normalization, shuffle, translation and jitter. In particular, the last two are data augmentation techniques and, while the first one helps the network to better generalize, the latter one tries to reproduce the wear of time which the archaeological artifacts are subject to and, for this reason, it is applied only to the test set.

The point clouds are fed as input to the neural network, which is a DGCNN. A Dynamical Graph CNN is a particular CNN which recomputes the graph using nearest neighbors in the feature space produced by each layer [Wang et al. 2019b]. We exploit local geometric structures by constructing a local neighborhood graph and applying the EdgeConv operator. The main differences and improvements with respect to the DGCNN developed by [Wang et al. 2019b] are the use of a different loss function, which takes into account the unbalanced dataset problem, and the concatenation of several features fed as input to the fully connected network. Specifically, the additional features are the vectors of the normals and those of the areas, that are a very relevant information for achieving the objectives of our research, in fact they bring an increase in performance. According to the knowledge we have, it is the first time that this kind of information is added to the features used in a DGCNN whose task is the classification of solid 3D fragments.

*Both authors contributed equally to this research.
[1]https://parcocolosseo.it/

Authors' address: Martina Valleriani, valleriani.1709963@studenti.uniroma1.it; Pietro Nardelli, nardelli.154680@studenti.uniroma1.it, La Sapienza University of Rome, Rome, Italy.

2021.

Besides our best model, we tried some variations both in the architecture and in the usage of additional techniques. As shown in section 5.3, we developed a custom Graph CNN, which does not have the dynamic graph update technique typical of the DGCNN. Moreover, we implement the possibility to use Differential Learning Rate (DLR), Stochastic Gradient Descent with Restarts (SGDR) or the K-Mean algorithm in support of the feature extraction of the convolutional layer (see figure 6 to visualize the corresponding layer).

At the end, in section 6, we show all the experiments done. They concern both the trials to reach the best model and the trials to test the robustness of the network. In particular, the first ones cover the attempts done to choose the techniques and the parameters that allow us to obtain the best performances. Instead, the second ones are related to two tests to check whether the network is robust to jitter (input noise) on the test set and is able to generalize regardless of the kinds of solid objects used for the training and the test phases. Especially these two last tests are very relevant for our research because they allow us to understand the behaviour of our model in real life cases. In particular, the jitter test (section 6.7.1) simulates the wear of time and make us understand how good the scanner quality should be. In both cases we obtain satisfactory outcomes proving the effectiveness of our model.

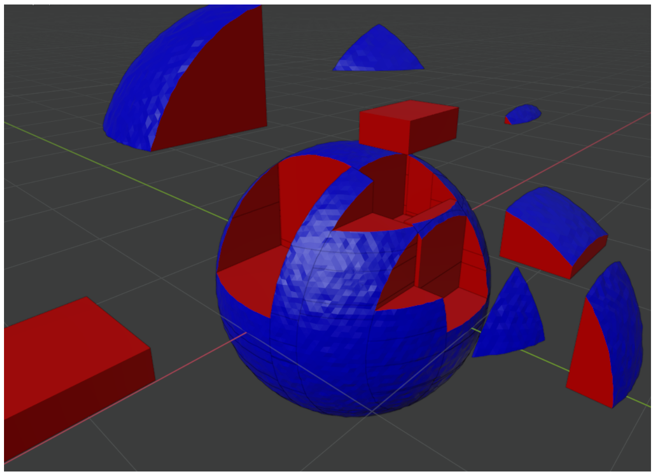

Fig. 1. Fragments obtained from a sphere using random cuts.

2 RELATED WORKS

2.1 Archaeological fragment classification/reconstruction

In the early nineteenth century the term archaeology was defined as the study of ancient finds with a strong emphasis on the artistic meaning. Thus, the archaeological finds were associated to the classical concept of beauty and the context in which they had been produced was not taken into account. Over time, archaeology acquired a meaning less related to the history of classical art. In fact, it went on to indicate the excavation activity and all the specific techniques for the search of the finds of antiquity. Over the years, we ended up with a more correct idea of archaeology: the aim of finding is to have more information on the historical-political, economic and cultural condition of the period to which the ancient finds belong [Mantovan and Nanni 2020].

With the development of technology, many researchers have been interested in finding the possible solutions to the resemble of objects with computer assistance in order to automatize this process and gain simpler and faster operations. In particular, the classification and reconstruction of archaeological remains starting from fragments are two of the most studied and relevant processes. Of course, over the years the several techniques adopted have evolved: from the analysis of 2D images to the study of 3D geometries, from the use of few algorithms to the application of deep neural networks, and so on.

In the early 2000s, Kampel and Sablatnig (2000) and McBride and Kimia (2003) 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. Their main focus is the use of partial curve matching to find similar portions of the fragments' boundaries. Smith et al. (2010) investigated the classification of thin-shell ceramics based on color and texture descriptors and formulated a new feature descriptor based on total variation geometry. Li-Ying and Ke-Gang (2010) proposed a classification method for ceramic cultural relics which first initializes center of clustering by re-division on multi-variant finite model and then applies a kernel-based fuzzy clustering algorithm in which parameters of different degree are used to control iterations of clustering. Zhou et al. (2011) described a classification algorithm for ancient porcelain classification, composed of the following principal steps: a color quantization method in HSI color space, a color-texture feature extractor based on Gabor filter, principal component analysis to reduce the features and, last, a nearest neighbor method to classify shards. Oxholm and Nishino (2013) 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, methods and, in general, theoretical aspects. Thus, for the task of recomposing 3D objects, the idea of work 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 characterize, describe and better interpret the object under study [Grilli and Remondino 2019]. Of course, the texture-based approach is still a valid option depending on the task and on the kind of solution which want to be used.

Both Sanchez-Belenguer and Vendrell-Vidal (2014) and Jampy et al. (2015) 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 preprocessing phase is applied to the point clouds obtaining keypoints 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.

Rasheed et al. (2017) developed a method for classifying pottery fragments into groups. The method begins with conversion of images from RGB color to HSV color, then a Self-Organization Map Neural Network is used for the classification task, relying upon the HSV color features. Also the research of Chetouani et al. (2018) is centered on the analysis of images. In particular, they proposed a CNN model to classify automatically ceramic shards.

Grilli and Remondino (2019) 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 of their research is the need to identify and map different states of conservation phases or employed materials in heritage objects. Gao and Geng (2020) 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 (2020b) implemented an algorithm with the help of a Backpropagation Neural Network to perform reconstruction of ceramic fragments using the slope features. The main principle of this method is to extract the edges of the fragments, which are lines, corners, and curves. Rasheed and Nordin (2020a) 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. To reconstruct ancient documents Ostertag and Beurton-Aimar (2020) presented a solution based on a Graph Neural Network, using pairwise patch information to assign labels (Up, Down, Left, Right or None) to edges representing the spatial relationships between pairs. Hu et al. (2020) 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, then they combine this algorithm with a custom CNN.

2.2 Deep Learning Methods for 3D Data

The kernel used in the convolution operator of the 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.

Scarselli et al. (2009) first proposed a neural network model, called Graph Neural Network model, that extends existing neural network methods for processing the data represented in graph domains. This network can process most of the useful kinds of graphs: acyclic, cyclic, directed and undirected. Instead, Bruna et al. (2014) extended the CNN architecture through the graph Laplacian, an operator which provides an harmonic analysis on the graphs. Also Maturana and Scherer (2015) exploited the CNN by proposing VoxNet, a novel architecture which integrates a volumetric occupancy grid representation with a supervised 3D CNN. An alternative method was presented by Su et al. (2015), which defined Multi-View CNN that combines information from multiple views of a 3D shape into a single and compact shape descriptor. Wu et al. (2015) proposed a different model: 3D ShapeNets. Given a depth map of an object and its convertion into a volumetric representation, 3D ShapeNets can understand object label, complete full 3D shape, and can predict the next best view if needed. Besides 3D ShapeNets and VoxNet, Qi et al. (2016) defined in general the volumetric CNNs to exploit it by proposing some interesting variations, trying to make the performances as good as those of the multi-view CNNs.

Between 2017 and 2018, the research 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 consumes point clouds, while on the other side the development of Graph CNNs. In the first case, the turning point has been the work done by Charles et al. (2017), that proposed PointNet, a novel architecture suitable for utilizing unordered sets of 3D points. Masci et al. (2018) 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. (2018) proposed PointCNN, which is a generalization of typical CNNs to feature learning from point clouds. The key idea of this method is the $\mathcal{X}$-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 Atzmon et al. (2018), whose main idea is to use extension and restriction operators for translating volumetric convolution to arbitrary point clouds. In the second case, instead, MoNet [Monti et al. 2017], ECC [Simonovsky and Komodakis 2017] and GAT [Veličković et al. 2018] are some of the most famous and relevant works. In particular, MoNet is a general framework allowing to design 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.

2.3 Normal vectors

From the literature we know that the inclusion of additional features, besides the point cloud coordinates, often brings useful information which may lead to achieve better performances. In fact, in the work done by Uy et al. (2019) 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.

Rabbani et al. (2006) presented a method for segmentation of point clouds using local surface normals and point connectivity. The normal for each point is estimated by fitting a plane to some neighboring points using either k-nearest or fixed distance neighbours.

Some normal estimation methods for point clouds have been proposed by Guo et al. (2016), Ben-Shabat et al. (2018) and Hyeon et al. (2019). The first ones compared ten popular local feature descriptors which need the normal vectors to be computed, the second ones developed a normal estimation method based on mixture of experts and scale prediction and the third ones presented NormNet, a point-wise normal estimation network.

Another spread usage of normals is to add them to the input features, as done by Lan et al. (2018), Zhao et al. (2019) and the already mentioned Hu et al. (2020). All these researches have in common the way in which the input is composed: the 3D point cloud coordinates plus the surface normals. Thus, each point of the point clouds is composed by xyz coordinates, normal values and possibly other features. In particular, Wang et al. (2019a) presented NormalNet, a voxel-based CNN designed for 3D object recognition. The network uses normal vectors of the object surfaces as input, which demonstrate stronger discrimination capability than binary voxels.

Then, both Widyaningrum et al. (2020) and Pierdicca et al. (2020) implemented a DGCNN approach adding meaningful information such as normals and other features. More particularly, the latter ones calculated the normals on Cloud Compare and give in input a block composed of 12 features including normal vectors, which pass through the EdgeConv layers.

2.4 Our contributions

First of all, one of our main contributions is the fact that our project has a key role for the reconstruction of ancient Roman archaeological artifacts. In particular, it is the first time that a group of researchers works with the aim of recomposing the remains of the Roman Forum. Since Rome and, in general, Italy are rich of Roman finds, the success of this final project would be a big breakthrough for archaeology and for our cultural heritage. The final project has a very high complexity and it is unthinkable to solve it in short time, so the best solution is to divide it into sub-problems and we work on one of these sub-tasks. For this reason, it is easily to understand how much our research is relevant and fundamental for the final outcome.

In addition to this, many contributions are presented in this research, starting from the dataset which was realized from scratch by Marco Serra. This dataset is an attempt to build a big archive of virtual archaeological fragments for the classification and reconstruction task. One of the principal positive aspects 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.

Furthermore, having regard the previously mentioned literature, we can state that we are the first ones to address the task of classifying internal and external fragments.

Another important innovation is the different usage of the normal vectors with respect to the state-of-the-art research (see section 2.3). In fact, these projects exploit the normals either doing normal estimation or feeding them as input together with the point cloud. This means that these additional features will be subject to all the layers of the network. On the contrary, our idea is to concatenate them only on the middle of the architecture, before the fully connected network. In this way, they will not be subject to the convolutional layers which will receive as input only the features regarding the point clouds.

As far as we know, no research has used the areas of the triangle meshes as additional features. From the literature the closest topic is the point cloud's density or the density of a mesh. Yan et al. (2020) proposed a rotated density-based network (RD-Net) in which they defined point-density information by calculating the point-density of each point. Instead, the density of a mesh is a function of the surface detailing, so the meshes will consistently have lower density in flat areas and higher density near curvatures [Bassier et al. 2020]. Our idea is very close to this reasoning, in fact the areas of the triangle meshes was taking into account as additional features. Since we work with 3D data and so the meshes are available, we think that the choice of the areas is better than the point cloud's density because this could produce error in the computation of the features, being the point cloud sampled randomly. Instead, with the choice of the areas, we add a useful information for the network in order to better distinguish between internal and external fragments. In fact, our dataset has flat areas for internal surfaces which means lower density and higher areas, and more regions with curvatures for external surfaces, thus higher density and lower areas.

Finally, another contribution regards the tests done to check whether the network is robust. Among the projects which have at least something in common with our work, none of them checked the network behaviour when the jitter is applied to the test set neither when the training and test phases are done on different solid objects. Instead, according to us, these tests are very useful because both of them tell us the robustness of the network. The jitter simulates the wear of time and tells us how good the scanner quality should be, while the test with different solid objects lets us understand if the model is able to generalize and correctly perform even if it is tested on a dataset with other kinds of objects. We believe that this information can help us to go deeper in the model behaviour and to extract even more knowledge.

3 DATASET

3.1 Fragments creation

Since there is no available dataset for this specific task, it was built from scratch. The first step was the production of simple 3D objects like the cube, the sphere and the icosphere, using the free and open source 3D creation suite Blender [2]. Thanks to this software, the fragments were generated by applying some cuts which could be exact or random, depending if the distance between a cut and another one is always the same or is random. At the beginning only exact cuts were applied, but then also random cuts were employed in order to get a more general and realistic dataset.

At the end of this procedure, we have got 12720 stl files which are the fragments of the starting 3D solid objects.

3.2 Imbalanced dataset

Before reasoning about the architecture or the preprocessing techniques, it is very important to look at the structure of the dataset. The main problem of our data lies in the imbalance. First, there are 2728 fragments obtained from exact cuts and 9992 fragments obtained from random cuts. Although imbalance, this difference is not a real problem for the network performances, while the critical issue is an unequal distribution of classes. In fact, due to the number of cuts and to the distances between cuts, there are 3223 internal and 9497 external fragments, so about the 25% and the 75% of the dataset, respectively.

Imbalanced dataset could lead to poor predictive performances because the model would train with biased information giving less relevance to the minority class. As explained in section 5.2, we handle this issue using a weighted loss and evaluating the model with a balanced accuracy and macro f1 score.

Notice that to solve this imbalance, we could either apply oversamplig or undersampling. With the first technique we would introduce some duplicates for the minority classes, while with the second approach we would remove some samples related to the majority class. Both the methods have some possible drawbacks: oversampling can easily introduce undesirable noise with overfitting risks [Huang et al. 2016] and may lead to a slower computational time, while undersampling may result in a lack of potentially useful information.

[2]https://www.blender.org/

For these reasons we prefer to find other smart and sophisticated techniques to solve the problem.

3.3 Dataset creation

3.3.1 TriMesh VS Open3D. In order to load stl files in Python, there are two possible libraries, either TriMesh [3] or Open3D [4]. Both libraries allow you to load 3D objects and to use some APIs to manipulate them. In this way we can load the fragments, visualize them and sample them to get the point clouds, which are the input of the network. A point cloud is a set of data points in space and the points represent the external surfaces of a 3D shape or object.

While Trimesh is faster, Open3D requires at least triple the computational time but the sampling is more homogeneous. In fact with Open3D each point has approximately the same distance to the neighbouring points [Open3D [n.d.]], while Trimesh randomly sample surface and volume of meshes [Trimesh [n.d.]].

3.3.2 From stl files to point clouds. The fragments are in stl format and we have to transform them into point clouds to make them feasible for the network. We can see an example of sampling of an internal fragment in figure 2, while in figure 3 is shown an example of sampling of an external fragment. Point cloud is a set of data points in space. The point cloud of an object is the set of 3D points sampled around the surface. It is represented by the xyz coordinates of the points and, in addition to its simplest form, other information can be used [Bello et al. 2020].

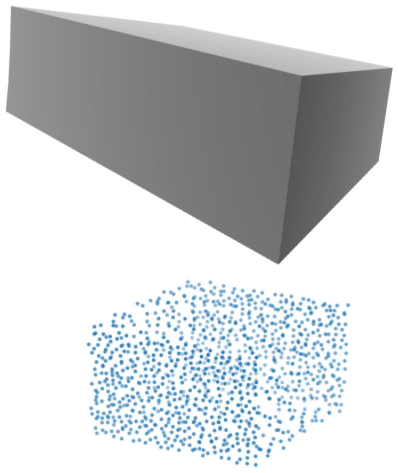
Fig. 2. Visualization of an internal fragment.

We have used Open3D methods to load stl files and to sample the mesh. The sampling points could be 1024 or 2048 and we have mainly worked with the dataset obtained with 1024 points. In fact, as we will see in section 6, the performances with the two datasets are quite similar but, since both the computational time for creating the dataset with 2048 points and the computational time for training with this dataset were much higher, the most convenient choice is the dataset with 1024 sampling points. In particular, the most significant cost is the first one because it requires ~12 hours.

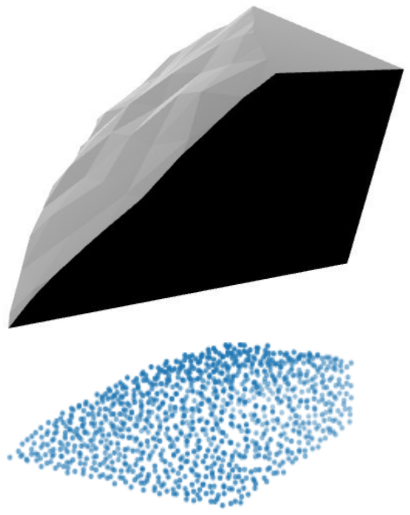
Fig. 3. Visualization of an external fragment.

3.3.3 Additional information. Since we need to obtain other information such as normals, we have used Open3D because it provides some specific and useful methods. In particular, Open3D allows us to get:

(1) all the vertices of each fragment;
(2) the triangles of the mesh represented with the indices of the vertices;
(3) the normals associated with the triangles.

We have to take into account the fact that we want to assign a normal to each point of the point cloud. Thus, for each point, we have searched the nearest vertex of the triangle and we have associated to this point the normal of the nearest triangle (i.e. the triangle which contains the nearest vertex to the point).

Another important information that we have added is the area of the mesh triangles. Again, we want to associate to each point this information other than to calculate it. First of all, in order to compute the area of a triangle, we use the Heron's Formula:

$$Area = \sqrt{s(s-a)(s-b)(s-c)}$$

where $s = \frac{a+b+c}{2}$ is the semi-perimeter, while a, b and c are the sides of the triangle.

[3]https://trimsh.org/index.html
[4]http://www.open3d.org/docs/release/

Then, as done for the normals, for each point we have searched for the nearest triangle and we have associated with it the corresponding triangle area.

As we will see in section 5.2, both for the normals and the areas, this information has been concatenated to the input of the first linear layer.

The addition of these vectors is very relevant for two different reasons. The normals represent the orientation of the surface [Cantor Rivera et al. 2012] so they are useful to distinguish between a flat and a curved surface and better capture the different orientations of the surfaces of the fragments. For how our dataset was built, we have that only the external surfaces are curve, while the internal ones are flat. So we can easily understand how much the normal vectors associated to each vertex are meaningful. Instead, the areas are bigger for bigger triangles, that is for surfaces which have been divided in a lower number of triangles, that is when the surface is flat or less curve. Of course, the addition of the area features may be susceptible to changes of dataset. So with a different dataset this information could become meaningless because, if the internal surfaces are faceted, they are represented by more triangles and so the areas of the triangles become lower. In this way, we would not have anymore the distinction between the areas of the internal and external triangles. Instead, in our case, the area vectors are quite different between internal and external surfaces, thus they provide a useful information, as well as the normal vectors.

3.3.4 Data splitting. Since we have a unique dataset, we have split it into three parts: train, validation and test set. In particular, the proportions are 70% for the train set, 15% for the validation set and 15% for the test set.

4 PREPROCESSING

Once the creation of the dataset is completed with the aforementioned steps, we apply some preprocessing techniques, particularly normalization and offline data augmentation techniques. Offline augmentation refers to editing and storing data on the disk. While transforming data on the fly can save memory, it will result in slower training [Shorten and Khoshgoftaar 2019] and so we had preferred the offline method rather than the online one.

Now we explain the several adopted preprocessing techniques.

4.1 Normalization

We apply normalization on all the point clouds. We first save the minimum and the maximum values among all the point cloud coordinates and then we normalize according to this formula:

$$x = \frac{(x - xmin)}{(xmax - xmin)}$$

where x is the dataset, $xmin$ is the minimum and $xmax$ is the maximum value, respectively.

4.2 Shuffle

We adopt two different kinds of shuffle:

(1) the first one is applied to the points of each point cloud, just by reordering them randomly;
(2) the second one is applied to the entire dataset using the *shuffle* parameter of the Dataloader, thanks to which the data reshuffle at every epoch.

4.3 Translation

We translate each point cloud just by adding to it a vector of the same size of a point and whose elements are random numbers between -0.2 and $+0.2$. So to each point cloud we sum up a different 3D random vector.

4.4 Jitter

This data augmentation technique is applied only on the test set because it has the role of testing the robustness of the model on a set affected by noise. This noise introduced by the jitter wants to reproduce the fragments worn by time, making them similar to the archaeological remains. Another meaning of jitter is the simulation of noise due to the type of scanner quality.

In order to apply the jitter, for each point cloud we sum up a clipped matrix. The clipped matrix is obtained by clipping between $-clip$ and $+clip$ a matrix of the same size of the point cloud, but sampled from a standard normal distribution and multiplied by a factor σ.

5 MODEL

Now we explain the adopted model which is based on the original paper about the DGCNN [Wang et al. 2019b], in particular the relevant differences are the kind of loss function and the different features fed as input to the fully connected layers.
Once the architecture and the functions used for our best model are made clear, we show some variations adopted based on typical machine learning techniques such as Differential Learning Rate (DLR), Stochastic Gradient Descent (SGD) with restarts and so on. These variations don't lead to better performances, however we find that it could be worthwhile for the community to know that these techniques could not be useful for this particular and specific task.

5.1 State-of-the-art architecture

As we previously said, the architecture from which we take inspiration is the network developed by Wang et al. (2019b) , called DGCNN, which is a convolutional neural network with dynamic graph update. They exploit local geometric

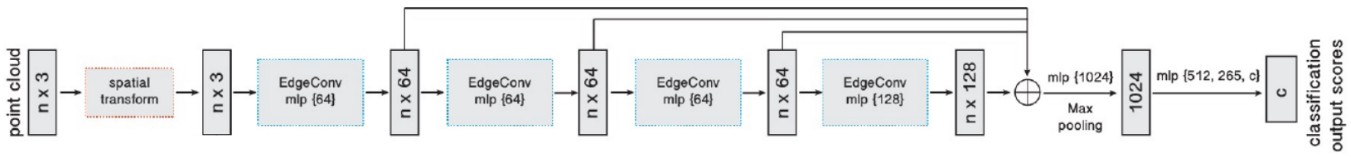

Fig. 4. DGCNN architecture by Wang et al.

structures by constructing a local neighborhood graph and applying convolution-like operations on the edges connecting neighboring pairs of points, in the spirit of graph neural networks [Wang et al. 2019b]. One of the most important aspects of this network is the Edge Convolution (EdgeConv) operator that now we will see in detail.

5.1.1 Edge convolution. Let's consider an F-dimensional point cloud with n points. At the beginning $F = 3$ because each point contains 3D coordinates, but in general F represents the feature dimensionality of a given layer. We compute the direct graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as the k-nearest neighbor (k-NN) graph representing local graph structure.

Edge features between node i and j are defined as $e_{i,j} = h_\Theta(x_i, x_j)$ where $h_\Theta : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$ is a non-linear function with a set of learnable parameters Θ that produces an output of dimension different from the input.

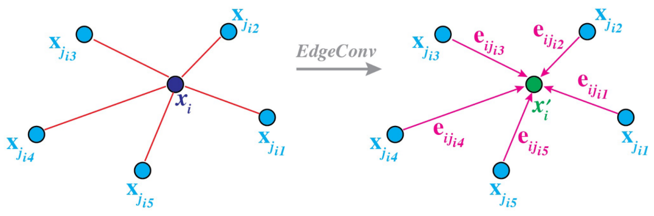

Fig. 5. The output of the edge feature is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

Finally, we can define the EdgeConv operation by applying an aggregation operation $\square$ such as summation or maximum. In particular, the operation used is the maximum and, as we can see from figure 5, the output of the i-th vertex is given by the following formula:

$$x'_i = \max_{j:(i,j)\in\mathcal{E}} h_\Theta(x_i, x_j).$$

Notice that, given an F-dimensional point cloud with n-points, the EdgeConv operator produces an F'-dimensional point cloud with the same number of points.

The choice of the edge function and the aggregation operation has a crucial influence on the properties of EdgeConv. The authors of this model adopt the following asymmetric edge function:

$$h_\Theta(x_i, x_j) = h_\Theta(x_i, x_j - x_i).$$

This explicitly combines global shape structure, captured by the coordinates of the patch centers x_i, with local neighborhood information, captured by $x_j - x_i$.
In particular, we can define our operator by notating

$$e'_{ijm} = ReLU(\theta_m \cdot (x_j - x_i) + \phi_m \cdot x_i)$$

which can be implemented as a shared MLP, and taking

$$x'_{im} = \max_{j:(i,j)\in\mathcal{E}} e'_{ijm}$$

where $\Theta = (\theta_1, ..., \theta_M, \phi_1, ..., \phi_M)$.

5.1.2 Dynamic graph update. Unlike graph CNNs, the graph is not fixed but rather is dynamically updated after each layer of the network. The architecture learns how to construct the graph $\mathcal{G}$ used in each layer rather than taking it as a fixed constant constructed before the network is evaluated. With dynamic graph updates, the receptive field is as large as the diameter of the point cloud, while being sparse. In the implementation, a pairwise distance matrix is computed in feature space and then the closest k points are taken for each single point.

5.1.3 Properties. There are two important and useful properties related to the choices made about the EdgeConv operation:

(1) *permutation invariance* —the ouput x'_i of the EdgeConvolution operator is invariant to permutation of x_j because the maximum operation is a symmetric function. Also the global max pooling is permutation invariant.
(2) *translation invariance* —supposing that points are shifted by a value of T, for the translated point cloud we have:

$$e'_{ijm} = \theta_m \cdot (x_j + T - (x_i + T)) + \phi_m \cdot (x_i + T) =$$

$$= \theta_m \cdot (x_j - x_i) + \phi_m \cdot (x_i + T).$$

It can be shown that part of the edge feature is preserved while a total translation invariance can be obtained with $\phi_m = 0$. In this case, the model is not capturing the global shape structure anymore given that it recognizes objects based only on an unordered set of patches (ignoring their positions and orientations). Instead, with both $x_j - x_i$ and x_i as input, it can take into account local and global shape information.

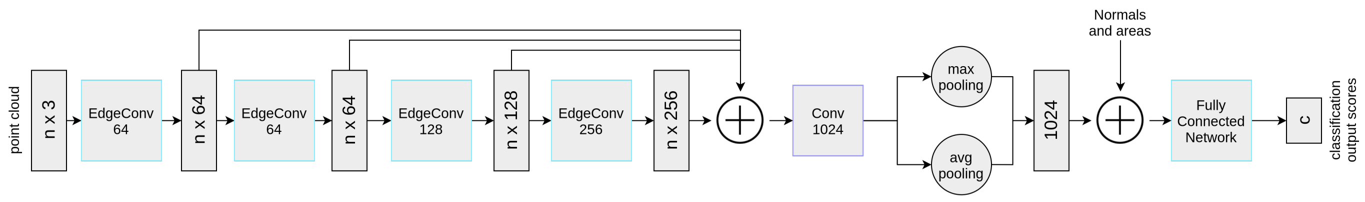

Fig. 6. Our model architecture.

5.1.4 Architecture. As shown in figure 4, the architecture receives as input a point cloud of dimension $n \times 3$. Four different EdgeConv layers produce features that are all fed as input to the next layer and, through skip-connections, are concatenated. Then the max pooling is applied and this result is the input of a multi-layer perceptron which generates classification scores for c classes.

5.2 Our model

Now that we have explained the basic model from which we take inspiration, we can illustrate our model with all its additions and modifications with respect to the basic one.

5.2.1 Output linear layer. First of all, we add as output layer a linear layer which returns a binary output. The reason is quite obvious since we have a binary classification task: 0 as label for internal and 1 for external fragments. Of course there would also be another possible choice: modifying the dimension of the old last layer rather than adding a new layer (as we do). We prefer the addition because we don't want to reduce the layer dimensions too fast.

5.2.2 Weighted loss. As seen in section 3.2, we have an imbalanced dataset and so we have to take into account this issue. A very important technique to avoid this problem is the choice of a weighted loss function, that is a loss function which explicitly weights penalties for two types of errors (i.e. false negative and false positive errors) in a binary classification problem [Phan et al. 2017]. 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.

Typically, a class-balanced loss assigns sample weights inversely proportionally to the class frequency [Cui et al. 2019]. There are different weighting schemes that can be used to compute this weighted loss, such as the Inverse of Number of Sample (INS), the Inverse of Square Root of Number of Samples (ISNS) or the Effective Number of Samples (ENS). We use another kind of weighted loss according to which the weight of a class is the size of the smallest class divided by the size of that class. Thus, the weights for the internal and the external classes are the following ones, respectively:

$$w_{int} = \frac{n_{int}}{n_{int}}$$

$$w_{ext} = \frac{n_{int}}{n_{ext}}$$

where n_{int} is the number of internal fragments, while n_{ext} is the number of external fragments.

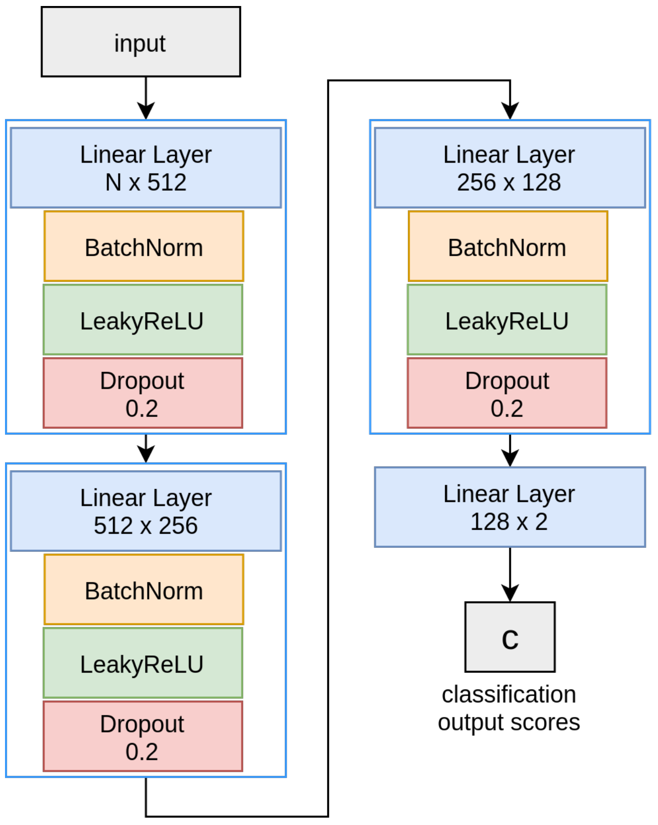

Fig. 7. Fully Connected Network of our model.

5.2.3 Architecture. As shown in figure 6, our model architecture is composed by:

- 4 EdgeConv layers;
- 1 convolutional layer;
- features concatenation;

- fully connected network.

Thus, a point cloud is fed as input into a first EdgeConv layer (see figure 8). A 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 produce the edge features. As 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 we recompute the graph features of the previous output, since we want to apply 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. Now, to the output of the conv layer, we apply the max pooling and average pooling operators and concatenate these vectors with the normals and the areas associated with the point cloud.

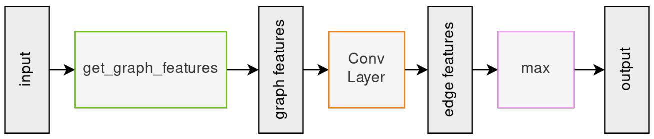

Fig. 8. Structure of the EdgeConv operator.

As last step, a fully connected network (see figure 7) is applied in order to obtain the classification output scores.

The fully connected network is composed by 3 modules and each of them is made of:

- a *linear layer*, which applies a linear transformation to the incoming data;
- *batch normalization*, which normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation, in order to increase the stability of a NN;
- *leaky ReLU*, which is an activation defined as follows:

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x \leq 0 \\ negative\_slope \times x, & \text{otherwise} \end{cases}$$

  where $negative\_slope$ is set to 0.2.
- *dropout*, which is set to 0.2.

After these three modules, there is a final linear layer which produces the classification output scores.

5.3 Model variations

As usually happens in computer science, we first build a base model and then tried to make improvements both in the architecture and in the usage of some deep learning techniques.

The variations that we adopt are the following ones:

- Graph CNN;
- K-Mean;
- Differential Learning Rates;
- SGD with Restarts.

While the first one is a completely different architecture which differs from our model (section 5.2) for the absence of the dynamic graph update, the other three are advanced deep learning techniques which aim to solve specific issues which usually appear in neural networks. From the literature, we already know that the Graph CNN provides poorer performances than the DGCNN, instead the other variations are not always the right or more efficient solutions, it depends on multiple factors.

However, it was only right to try them and show to other researchers that the following variations give us worse results for our specific task.

5.3.1 Graph CNN. We also deployed a different architecture that is not based on the dynamic graph update technique. As we know, the Convolutional Neural Networks based on Graphs are the forerunners of the DGCNN. In fact, the main difference introduced by the authors of the DGCNN [Wang et al. 2019b] lies in the Dynamic Graph Update technique.

In literature we can see that the DGCNN performs better than the other graph CNNs (without the dynamic graph update), as shown by Wang et al. (2019b) and Hong et al. (2020). In fact, these papers illustrate how to obtain better performances using the DGCNN, while with the previous graph CNNs like KD-Net [Klokov and Lempitsky 2017] the results are slightly lower. However, the DGCNN architecture is relatively young because it was developed in 2019 and thus not so much research has been done on this architecture. This means that we cannot be totally sure about the effectiveness of this model. Thus, we decide to develop a custom Graph CNN without the Dynamic Graph Update in order to check whether this technique actually leads to better results also for our task.

In figure 9 there is our custom Graph CNN architecture, that is made of the following steps:

- graph feature extraction;
- 4 convolutional layers;
- feature concatenation;
- fully connected network.

Thus, first of all we extract the graph features from the point cloud which is the input of the model. Notice that we compute these graph features only once at the beginning, in this way we have a static graph instead of a dynamic one. This is the key point of this network. These features are the input of a network composed by a sequence of four convolutional layers. To the resulting output of these layers we apply the max, that is the aggregation operator, and then the max pooling and the average pooling operators. Finally, we concatenate these two pooling vectors with the initial point cloud, its normals

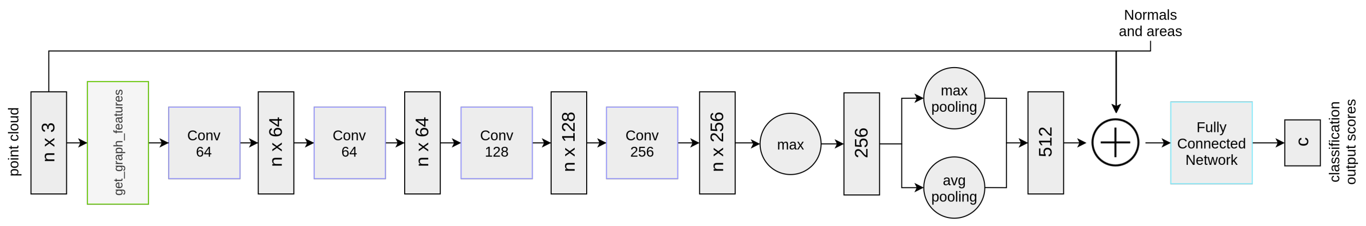

Fig. 9. GCNN architecture.

and its areas. Now these are the features fed as input into the fully connected layers thanks to which we obtain the classification output scores.

5.3.2 The K-Means clusters as additional features. The K-Means [MacQueen 1967] is a well-known unsupervised learning algorithm that aims to solve the clustering problem. In other words, given the input data $D = \{z_n\}$, the unsupervised clustering has the purpose of finding multiple classes from data where target values are not available.

The main idea is to define k centroids, one for each cluster. This algorithm aims at minimizing a squared error function that is the sum of the distances between each cluster centre and the data points:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| z_i^j - c_j \right\|^2$$

where z_i^j is the i-th data point for the cluster j and c_j is the j-th cluster centre [Faraoun and Boukelif 2006].

The K-Means is defined by the following steps:

(1) choose the value of k, i.e. the number of clusters;
(2) set an initial partition assigning the training samples randomly or systematically, that is:
    (a) take the first k training samples as single-element clusters;
    (b) the remaining $(N - k)$ training samples are assigned to each cluster with the nearest centroid, then recompute the cluster centre after each assignment;
(3) compute the distance of each sample from the centroid of each cluster and check if all the samples are correctly classified in the right classes, otherwise switch the misclassified ones in the correct clusters;
(4) update the centroids involved in the switch;
(5) repeat the two previous steps until convergence, that is until no new assignment is possible.

However we have to take into account the following issues. First of all, the number of clusters k must be determined beforehand, then it is sensitive to initial condition when few data are available. Moreover, it is not robust to outliers [Bishop 2006] and very far data from the centroid may pull the cluster center away from the real one.

While in some cases k-means clusters are used as pseudo-labels [Gupta et al. 2020], in our case the role of the k-means algorithm is to enhance the learning of the features in order to drive the network on the right classification direction. This can be achieved by clustering the feature space x produced at the end of the convolutional layer. Since we want to reduce the input space we apply the Principal Component Analysis (PCA) that extracts only the most relevant information, simplifying the complexity in high-dimensional data while retaining trends and patterns. If the variables use different scales, it may be appropriate to standardize them such that each variable has unit variance [Lever et al. 2017]. For this reason, first we apply normalization to each element x_i of the feature space x in the following way:

$$x_{norm} = \frac{(x_i - x_{min})}{(x_{max} - x_{min})}$$

where x_{min} and x_{max} are respectively the minimum and the maximum value. Then, we apply PCA to x_{norm} so that we obtain z_i. Finally, the K-Means algorithm is computed on z_i with number of clusters equals to three. At the end, we obtain as much labels as the number of input points (equal to the number of sampling points), which are the features that has to be concatenated to the other information.

5.3.3 Differential Learning Rates. Another technique that we try to adopt is the Differential Learning Rates, also called Discriminative Fine-Tuning [Howard and Ruder 2018]. Essentially, in the Stochastic Gradient Descent (SGD) optimization algorithm, rather than the same learning rate for all layers, we associate to each linear layer of the network a different learning rate value, in particular:

- linear1: $\eta/4$;
- linear2: $\eta/3$;
- linear3: $\eta/2$;
- linear4: η,

where η is the learning rate value. Thus we create four lists, each of them containing both the weights and the bias of that layer.

In general, in neural networks, higher layers are more specialized than the lower layers. This means that the layers could be fine-tuned to different extents [Yosinski et al. 2014].

For this reason, we choose a lower value of η for the first linear layer. In this way, we are giving less importance to the update of the weights of this layer, because it is the less specialized. Layer by layer we increase the learning rate and for the fourth one we set it to the highest value, placing greater emphasis on the weights of this last layer because it is the most specialized.

Usually the regular stochastic gradient descent update of a model's parameters θ at time step t looks like the following [Ruder 2017]:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta)$$

where $\nabla_\theta J(\theta)$ is the gradient with regard to the model's objective function. Instead, the SGD update with discriminative fine-tuning is the following:

$$\theta_t^l = \theta_{t-1}^l - \eta^l \cdot \nabla_{\theta^l} J(\theta)$$

where θ^l contains the parameters of the model of the l-th layer and η^l is the learning rate of the l-th layer.

5.3.4 SGD with restarts. Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by the model's parameters $\theta \in \mathbb{R}^d$ by updating them in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ with respect to the parameters [Ruder 2017].

SGD is a stochastic approximation of gradient descent optimization, which uses an estimate of the gradient calculated from a randomly selected mini-batch, instead of the actual gradient computed from the entire dataset. In particular, we use SGD with Momentum because the momentum method is a technique for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations [Sutskever et al. 2013].

Stochastic Gradient Descent with Restarts (SGDR) is a variant of learning rate annealing where in each restart the learning rate is initialized to some value and is scheduled to decrease [Loshchilov and Hutter 2017]. SGDR is based on the Cosine Annealing, that is a method to adjust the learning rate based on the number of epochs, and add to it the restarting technique. SGDR is used to essentially put the parameters out of the minimum to which they previously converged, so it avoids local minima.

The Cosine Annealing schedule is the following one:

$$\eta_t = \eta_{min} + \tfrac{1}{2}(\eta_{max} - \eta_{min})\left(1 + cos\left(\tfrac{T_{cur}}{T_{max}}\pi\right)\right)$$
$$\text{with } T_{cur} \neq (2k+1)T_{\max};$$

$$\eta_{t+1} = \eta_t + \tfrac{1}{2}(\eta_{max} - \eta_{min})\left(1 - cos\left(\tfrac{1}{T_{max}}\pi\right)\right)$$
$$\text{with } T_{cur} = (2k+1)T_{\max},$$

where η_{max} is set to the initial learning rate and T_{cur} is the number of epochs since the last restart in SGDR.

Although model performance temporarily suffers when the restart of the learning rate cycle is applied, the performance eventually surpasses the previous cycle after annealing the learning rate [Huang et al. 2017].

Thus, with SGDR we can start with high values of learning rate in order to quickly find a local minimum and then we use lower values to move towards the minimum. In order to find a more stable local minimum, we can reset the scheduler (that is, increase the learning rate) from time to time. These are the "restarts" which have the aim of encouraging the model to escape from a local minimum.

6 EXPERIMENTS AND RESULTS

In this section we analyze the experiments done for finding the best model for our task and for showing the effectiveness of the several techniques adopted.

In particular, first we analyze the experiments done from the baseline to our best model illustrating the techniques and hyperparameters which allow us to obtain better results. Then, we describe the trials in which we use some techniques explained in previous sections that don't make improvements and we see why. At the end, two kinds of tests are shown: the first one attempts to simulate the worn due to time by adding noise to the point cloud, instead the second one wants to check the robustness of the model by training and testing on different solid objects.

6.1 Classification metrics

As we have seen in section 3.2, we work with an imbalanced dataset and so we have to choose two kinds of classification metrics which take into account this issue. In particular, these metrics are the balanced accuracy and the macro F1 score. The first one is defined as the average of recall obtained on each class[5], while the second one calculates metrics for each label and find their unweighted mean[6]. Thus, macro F1 is defined as follows:

$$\text{macro F1} = \frac{1}{N}\sum_{i=0}^{N} \text{F1}_i$$

where

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

In this way, macro F1 score does not take label imbalance into account.

6.2 From baseline to best model

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

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

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

- table 1 which illustrates the performances, that is, the balanced accuracy (*avg acc*) and the macro F1 score (*macro-F1*);
- table 2 which displays the differences between a model and the previous one, that is the variations of the performance values;
- table 4 which shows the differences of table 2 but expressed in percentage, so that the performance improvements are more visible.

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

Table 1. Model results from baseline to best model.

Model	avg acc	macro-f1
$M_{baseline}$	0.7864	0.8310
$M_{weighted}$	0.8927	0.8512
$M_{normals}$	0.9532	0.9320
$M_{normals\_areas}$	0.9778	0.9418
M_{best}	0.9768	0.9523

Table 2. Differences between models from baseline to best model.

Model	Δ_{avg_acc}	Δ_{macro_F1}
$M_{baseline}$	-	-
$M_{weighted}$	+0.1063	+0.0202
$M_{normals}$	+0.0605	+0.0808
$M_{normals\_areas}$	+0.0246	+0.0098
M_{best}	−0.0010	+0.0105

About the hyperparameters, they are written in table 3. However, we want to specify that during the experimental phase we changed some hyperparameters but the values shown in table 3 have been the best ones. Thus, the following trials are based on these best values.

Moreover, it is necessary to specify that for each trial, even if the number of epochs is set to 20, the resulting model is saved only if the performances increase between an epoch and the previous ones.

6.2.1 *Trial 1 - $M_{baseline}$.* We start reproducing the architecture of the original DGCNN paper [Wang et al. 2019b]. This preliminary step is fundamental because we want to have a comparison with the state-of-the-art model for the DGCNN. With this network we have quite good results in performances, as we can see from table 1. However, looking at the confusion matrix in figure 10, we can see that this model is not accurate because the internal fragments are not correctly classified for about half of the times - it is like a coin flip. We actually expected this bad result for the minority class, i.e. the internal fragments, because the loss function used by the authors of the DGCNN does not take into account label imbalance.

Table 3. Initial hyperparameters (model26_2).

Hyperparameter	Value
epochs	20
learning rate	0.001
momentum	0.9
dropout	0.2
k	20
t_{max}	20
restart epoch	20
sampling points	1024
embedding dims	1024

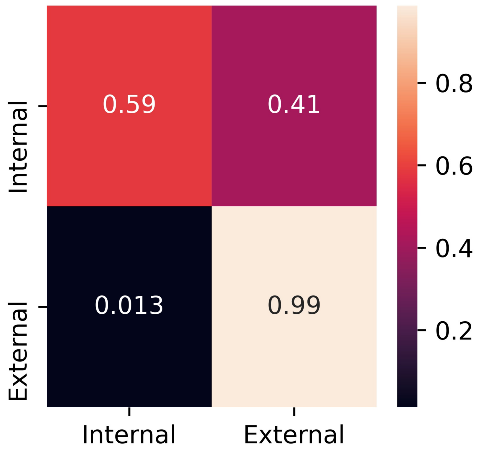

Fig. 10. Confusion matrix of $M_{baseline}$.

6.2.2 *Trial 2 - $M_{weighted}$.* In this experiment we handle the problem of the label imbalance by building a weighted loss function, as previously explained in section 5.2.2. In this way, the performances are significantly increased: the macro F1 score increases of 2.4%, while the balanced accuracy has got an increment of 13.5%. This result is appreciable also looking at the confusion matrix (figure 11). Now the internal fragments have the 88% of accuracy, so the issue of the imbalanced dataset is solved having found a method to handle it.

6.2.3 *Trial 3 - $M_{normals}$.* The changes made are the addition of the normal vectors and the consequent elimination of the shuffle during the dataset creation. The normals represent

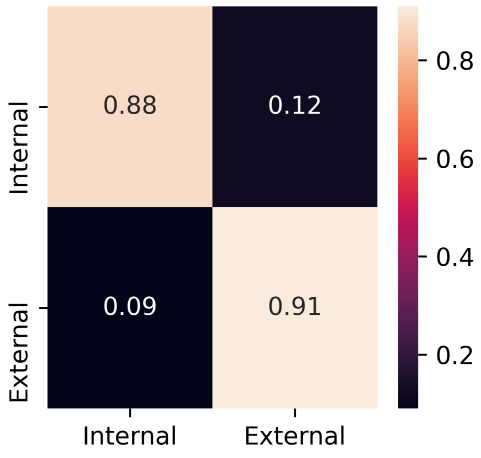

Fig. 11. Confusion matrix of $M_{weighted}$.

the orientation of the surfaces of the fragments. Thus, as explained in section 3.3.3, this is one of the most relevant and innovative aspects of our work. In fact, thanks to this additional information, the performances are increased of 6.8% and 9.5% for the two classification metrics. This is a remarkable improvement which gave a turning point to our research.

The reason why we have to delete the shuffle technique lies in an implementation choice. While until this moment we used the shuffle because the DGCNN is permutation invariant, now we cannot use it anymore. In fact, since we add the normal vectors during the dataset creation phase, we have to consider the correspondence between the points of a point clouds and their normals. This means that we cannot shuffle the point cloud, otherwise the mapping would be lost.

6.2.4 Trial 4 - $M_{normals\_areas}$. Since in trial 3 we noticed a very good improvement by adding more information to the network, we spent some time to find other useful and significant information. In particularm we found that the areas can help the network to extract more information from our dataset and so to better classify the fragments. Again, in section 3.3.3 we explained the reason why we add also the areas of the mesh triangles. Briefly, in our case, greater areas correspond to flat surfaces, while smaller areas correspond to more faceted surfaces. This means that the external surfaces are represented by more triangles with smaller areas, so the areas are helpful features for the network.

The result of this experiment is a slight increase of performances of about 2.6% and 1.1%.

6.2.5 Trial 5 - M_{best}. Finally, in this experiment we apply normalization and translation to the dataset and we get the

Table 4. Differences in percentage between models from baseline to best model.

Model	$\%\Delta_{avg\_acc}$	$\%\Delta_{macro\_F1}$
$M_{baseline}$	-	-
$M_{weighted}$	+13.5%	+2.4%
$M_{normals}$	+6.8%	+9.5%
$M_{normals\_areas}$	+2.6%	+1.1%
M_{best}	−0.1%	+1.1%

best model. With respect to the previous model, there is a slight decrease of average accuracy of about −0.1% and a macro-F1 score increase of about 1.1% . This can be attributed to the data augmentation technique, explained in section 4, that double the size of the dataset with new augmented samples, and in a small portion to the normalization that generally speeds up learning and leads to faster convergence.

Obviously, the increase of the performances is not particularly significant because our architecture fastly learns to classify the fragments and because the model is (partially) translation invariant, as described in section 5.1.2.

This model produce excellent results as we can see from the confusion matrix (see figure 12), in which the external fragments reach an accuracy of 97% while the internal ones are predicted almost perfectly (99%).

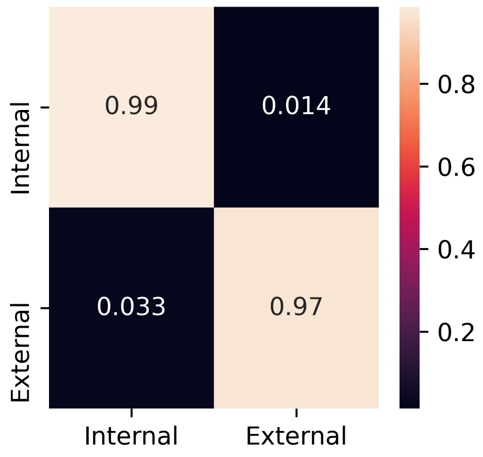

Fig. 12. Confusion matrix of M_{best}.

6.3 Input dropout

With respect to model $M_{normals}$ we also try the input dropout technique which randomly drops some input data that acts as a regularizer and makes the model more robust. It works

Table 5. Results of the models with the input dropout technique starting from the model $M_{normals}$.

Dropout	avg acc	macro-f1
0.2	0.9509	0.9266
0.4	0.9490	0.9280
0.7	0.9580	0.9019

when the next layers are able to reconstruct the data without any loss of information. Otherwise it is not useful. In our case, as we can see from table 5, the network works reasonably well even with the dropout on the input but it does not reach performances better than those obtained with $M_{normals}$. This means that, even if the network can reconstruct the dropped data, there is inevitably a loss of information. In fact, the greater the dropout value, the lower the performances, though slightly. We expected this performances decay because the paper [Srivastava et al. 2014] proves that for the input units the optimal probability of retention is usually closer to 1, which means that the dropout value has to be closer to 0.

6.4 K-Means

These experiments are focused on the application of the K-Means clusters as additional features. We have done several trials using K-Means clusters as additional features, but in each of them we did not obtain better results. Even when the performances of the model with the K-Means were similar to those of the model without it, the computational time was much higher in the first case. Since we always have quite the same results, we show only the most significant experiment, that in which we compare M_{best} with the same model with the K-Means in addition, $M_{best+kmean}$. Thus, in table 6 there are the results of the two metrics and the variation in percentage, which highlights how $M_{best+kmean}$ is not able to reach the same behaviour of M_{best}. Then, in figure 13 we can see the confusion matrix of $M_{best+kmean}$, which shows that the classifier is overall quite good, but not as good as that of M_{best} (see figure 12).

Thus, from these results we can understand how, in our case, the K-Means is not completely able to grasp the distinction between clusters. This may be due to a reduction of dimension, in fact, as seen in section 5.3.2, we first apply the Principal Component Analysis which reduces the dimension from 1024×1024 to 1024×3. Besides, another possible reason of these unsuccessful attempts may be one of the typical K-Mean issues: non-robustness to outliers (very far data from the centroid may pull the cluster center away from the real one). Since also the PCA is sensitive to outliers, we are strongly led to think that the sensitiveness to outliers could be emphasized by using these two techniques together.

Table 6. Results of the model with the K-Means technique compared with M_{best}.

Model	avg acc	macro-f1
M_{best}	0.9768	0.9523
$M_{best+kmean}$	0.9588	0.9397
%Δ	-1.8%	-1.3%

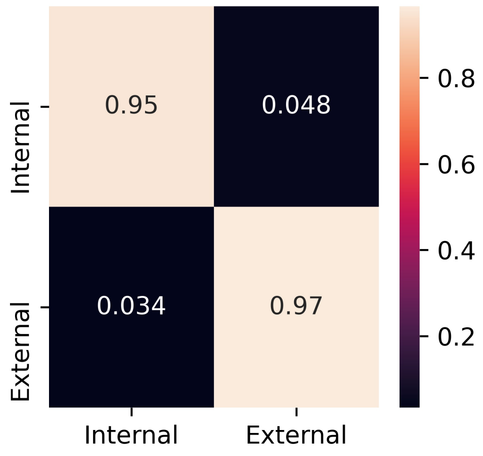

Fig. 13. Confusion matrix of the model $M_{best+kmean}$.

6.5 DLR

In this trial, with respect to $M_{normals\_areas}$ we add the differential learning rate technique, as explained in section 5.3.3. The results are shown in table 7. We can clearly see a performances decay when the DLR is applied. A possible reason of this result may be the following consideration: since we have a shallow fully connected network made of only four linear layers, the learning rates computed for the first layers might be too small, not enabling to sufficiently learn to classify. Maybe, in this way, the error is propagated up to the fourth layer so that the network worsens the performances.

6.6 Graph CNN

Another set of experiments is based on our custom Graph CNN explained in section 5.3.1. In particular, we built the network and illustrate only the most significant trials in comparison with the models based on the DGCNN (i.e. the models shown in table 1). Thus, also for the Graph CNN architecture, we start from a baseline model ($M_{gcnn\_baseline}$),

Table 7. Result of the model with the DLR technique compared with $M_{normals\_areas}$.

Model	**avg acc**	**macro-f1**
$M_{normals\_areas}$	0.9778	0.9418
$M_{normals\_areas+dlr}$	0.9687	0.9322
%Δ	-0.9%	-1%

then we change the loss function by using the weighted loss ($M_{gcnn\_weighted}$), we add the normal vectors ($M_{gcnn\_normals}$) and the areas ($M_{gcnn\_normals\_areas}$) and at the end we use the offline data augmentation technique besides normalization (M_{gcnn_best}). The results are shown in table 8.

Looking at the table 9, we can see that these models are not able to reach the performances of the models based on the DGCNN. In fact, we know that the Graph CNN has not got the Dynamic Graph Update technique which has a key role in improvements. Also the literature confirms that the absence of this technique may affect the performances, as shown by [Wang et al. 2019b] and [Hong et al. 2020]. However, the results are quite satisfactory especially considering that the network was built from scratch. It is relevant to notice that also for this architecture the additional features (normal vectors and areas) and the offline data augmentation (see sections 3.3.3 and 4) have a fundamental role for the improvement of the results, as illustrated in table 8.

Table 8. Results of the models with the Graph CNN architecture.

Model	**avg acc**	**macro-f1**
$M_{gcnn\_baseline}$	0.7315	0.7707
$M_{gcnn\_weighted}$	0.8356	0.8013
$M_{gcnn\_normals}$	0.9598	0.9294
$M_{gcnn\_normals\_areas}$	0.9728	0.9361
M_{gcnn_best}	0.9719	0.9464

Table 9. Differences between GCNN models and reference models (based on DGCNN).

Model	**Ref. model**	$\%\Delta_{avg\_acc}$	$\%\Delta_{macro\_F1}$
$M_{gcnn\_baseline}$	$M_{baseline}$	−7%	−7.3%
$M_{gcnn\_weighted}$	$M_{weighted}$	−6.4%	−5.9%
$M_{gcnn\_normals}$	$M_{normals}$	+0.7%	−0.3%
$M_{gcnn\_normals\_areas}$	$M_{normals\_areas}$	−0.5%	−0.6%
M_{gcnn_best}	M_{best}	−0.5%	−0.6%

6.7 Other experiments

In order to check whether the network is robust and performs in a satisfactory way also in different situations, we thought to make two kind of sets of trials. In the first one we introduce noise on the test set to simulate the fact that the objects are worn out by time, like for actual archaeological remains. In the second group of experiments each time we train and test our best model M_{best} on datasets containing different kinds of solid objects.

Table 10. Results of the evaluation on model M_{best} with jitter in the test set. Next to each metric is written the variation in percentage w.r.t M_{best}.

clip	σ	**avg acc**	$\%\Delta_{avg\_acc}$	**macro-f1**	$\%\Delta_{macro\_F1}$
0.02	0.01	0.9748	−0.2%	0.9539	+0.2%
0.04	0.01	0.9652	−1.2%	0.9494	−0.3%
0.02	0.08	0.9598	−1.7%	0.9489	−0.4%
0.04	0.02	0.9544	−2.3%	0.9483	−0.4%
0.04	0.08	0.9002	−5.8%	0.9331	−2%
0.09	0.09	0.7357	−24.7%	0.7902	−17%
0.1	0.09	0.6744	−30.6%	0.7243	−33.9%
0.2	0.2	0.5000	−48.8%	0.4515	−52.6%

Table 11. Results of the test using different solid objects for the training (O_{train}) and the test (O_{test}). Next to each metric is written the variation in percentage w.r.t M_{best}.

O_{train}	O_{test}	**avg acc**	$\%\Delta_{avg\_acc}$	**macro-f1**	$\%\Delta_{macro\_F1}$
cube	sphere	0.9582	−1.9%	0.9336	−2%
cube	icosph.	0.9592	−1.8%	0.9438	−0.9%
sphere	cube	0.8956	−8.3%	0.9089	−4.6%
sphere	icosph.	0.9535	−2.4%	0.9535	+0.1%
icosph.	cube	0.9188	−5.9%	0.9338	−1.9%
icosph.	sphere	0.9623	−1.5%	0.9532	+0.1%

6.7.1 Noise addition. In these experiments we introduce jitter on the test set, specifically on the point clouds of the test set. We can see jitter as a Gaussian noise applied on the input. Its aim is to reproduce the wear due to time that usually affects real archaeological fragments. Another important purpose of this technique is the simulation of the noise due to the 3D scanner quality. In fact, thanks to this set of trials, we can understand which has to be the quality of the scanner used for the dataset creation. If our model performs quite well also in presence of little amounts of input noise, we can state that it is not necessary to have the highest-level scanner on the market and moreover we do not need to work with very high and accurate quality 3D objects.

Thus, looking at the results shown in table 10, we can understand whether the network is robust to the jitter or not. First of all we have to remember that, as explained in section 4.4, the parameters of this kind of noise are $clip$ and σ. Having said that, from the table we can see that up to 0.04 for $clip$ and 0.08 for σ the network performs quite well, maintaining about and over the 90% of average accuracy and macro F1 score. Since we apply normalization before to add jitter and since we clip the noise matrix by a factor $clip$, we know that our model is able to correctly classify the fragments ($\sim$ 90%) even with a 4% error on the point clouds. Beyond 0.04 for $clip$ and 0.08 for σ, the performances decrease and for high values of these parameters we get that the results become as a flip coin.

6.7.2 Different solid objects. In each trial we train our best model M_{best} on a dataset containing only a type of objects and test it on another dataset related to a different kind of solid objects. For instance, we train on cubes and test on spheres, and so on. Essentially, by training on fragments from a certain solid and testing on another solid, we want to test the capability of the network to generalize. This means that we want to check whether the results of our model strongly depend on the kind of objects used for training or if it is able to correctly perform independently from the kind of objects.

In table 11 we show the results obtained in the various experiments, adding the percentage variations with respect to our best model trained and tested on the entire dataset. Of course, in most of the trials we have seen a slight degradation of the performances. The reason is quite obvious: while our best model M_{best} trained on the entire dataset (composed of cubes, spheres and icospheres) is able to correctly classify fragments of different kinds of solids, these experiments are specifically trained on a particular category and thus are less able to classify kinds of objects different from the ones used for the training. Nevertheless we are satisfied from the results obtained in these trials because most of the times the performances are about 95%. Only when we test on the cubes we have lower values (however about 90%), because the fragments of the cubes are very similar to the internal fragments of all the objects so it is very easy for the network to misclassify these elements.

Therefore we can state that our model M_{best} is not highly dependent from the kinds of objects on which it is trained and it performs very well even in these situations.

7 CONCLUSION

In this research we propose a slightly adapted DGCNN to solve the classification task of distinguish between internal and external ancient archaeological fragments.

The main contributions lie in the usage of a custom dataset, the concatenation of normal vectors in the middle of the architecture and the addition of areas as discriminative features. Finally, we examine the several tests done. Starting from the trials to get our best model, we illustrate some techniques attempted to improve the network and then we analysed additional experiments done to check the robustness of our best model.

From the results, we can see that our network achieves very high performances, even in presence of additional noise on the test set and in case of different solid objects used during the training and test phases. Thus, our network is robust and is able to generalize, which are both very important properties in view of the future reconstruction task.

8 ACKNOWLEDGMENTS

We gratefully acknowledge professor Stefano Giagu for the idea of introducing the normal vectors and for the theoretical support and professor Marco Serra to have let us use its own dataset.

REFERENCES

Matan Atzmon, Haggai Maron, and Yaron Lipman. 2018. Point Convolutional Neural Networks by Extension Operators. *arXiv:1803.10091 [cs]* (March 2018). http://arxiv.org/abs/1803.10091 arXiv: 1803.10091.

Maarten Bassier, Maarten Vergauwen, and Florent Poux. 2020. Point Cloud vs. Mesh Features for Building Interior Classification. *Remote Sensing* 12, 14 (July 2020), 2224. https://doi.org/10.3390/rs12142224

Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. 2020. Review: Deep Learning on 3D Point Clouds. *Remote Sensing* 12, 11 (May 2020), 1729. https://doi.org/10.3390/rs12111729

Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 2018. Nesti-Net: Normal Estimation for Unstructured 3D Point Clouds using Convolutional Neural Networks. *arXiv:1812.00709 [cs]* (Dec. 2018). http://arxiv.org/abs/1812.00709 arXiv: 1812.00709.

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]* (May 2014). http://arxiv.org/abs/1312.6203 arXiv: 1312.6203.

Diego Hernando Cantor Rivera, Diego Cantor, and Brandon Jones. 2012. *WebGL: beginner's guide ; become a master of 3D web programming in WebGL and JavaScript*. Packt Publishing Limited, Birmingham.

R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Honolulu, HI, 77–85. https://doi.org/10.1109/CVPR.2017.16

Aladine Chetouani, Teddy Debroutelle, Sylvie Treuillet, Matthieu Exbrayat, and Sebastien Jesset. 2018. Classification of Ceramic Shards Based on Convolutional Neural Network. In *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, Athens, 1038–1042. https://doi.org/10.1109/ICIP.2018.8451728

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-Balanced Loss Based on Effective Number of Samples. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Long Beach, CA, USA, 9260–9269. https://doi.org/10.1109/CVPR.

2019.00949

Kamel Faraoun and Aoued Boukelif. 2006. Neural networks learning improvement using the K-means clustering algorithm to detect network intrusions. *International Journal of Computational Intelligence* 3 (01 2006), 161–168.

Hongjuan Gao and Guohua Geng. 2020. Classification of 3D Terracotta Warrior Fragments Based on Deep Learning and Template Guidance. *IEEE Access* 8 (2020), 4086–4098. https://doi.org/10.1109/ACCESS.2019.2962791

Eleonora Grilli and Fabio Remondino. 2019. Classification of 3D Digital Heritage. *Remote Sensing* 11, 7 (April 2019), 847. https://doi.org/10.3390/rs11070847

Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, Jianwei Wan, and Ngai Ming Kwok. 2016. A Comprehensive Performance Evaluation of 3D Local Feature Descriptors. *International Journal of Computer Vision* 116, 1 (Jan. 2016), 66–89. https://doi.org/10.1007/s11263-015-0824-y

Divam Gupta, Ramachandran Ramjee, Nipun Kwatra, and Muthian Sivathanu. 2020. Unsupervised Clustering using Pseudo-semi-supervised Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJlnxkSYPS

Jinseok Hong, Keeyoung Kim, and Hongchul Lee. 2020. Faster Dynamic Graph CNN: Faster Deep Learning on 3D Point Cloud Data. 8 (2020), 10.

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 328–339. https://doi.org/10.18653/v1/P18-1031

Yao Hu, Guohua Geng, Kang Li, Wei Zhou, Xingxing Hao, and Xin Cao. 2020. SRG-Net: Unsupervised Segmentation for Terracotta Warrior Point Cloud with 3D Pointwise CNN methods. *arXiv:2012.00433 [cs]* (Dec. 2020). http://arxiv.org/abs/2012.00433 arXiv: 2012.00433.

Chen Huang, Yining Li, Chen Change Loy, and Xiaoou Tang. 2016. Learning Deep Representation for Imbalanced Classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Las Vegas, NV, USA, 5375–5384. https://doi.org/10.1109/CVPR.2016.580

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. Snapshot Ensembles: Train 1, get M for free. *arXiv:1704.00109 [cs]* (March 2017). http://arxiv.org/abs/1704.00109 arXiv: 1704.00109.

Janghun Hyeon, Weonsuk Lee, Joo Hyung Kim, and Nakju Doh. 2019. NormNet: Point-wise normal estimation network for three-dimensional point cloud data. *International Journal of Advanced Robotic Systems* 16, 4 (July 2019), 172988141985753. https://doi.org/10.1177/1729881419857532

F. Jampy, A. Hostein, E. Fauvet, O. Laligant, and F. Truchetet. 2015. 3D puzzle reconstruction for archeological fragments, Robert Sitnik and William Puech (Eds.). San Francisco, California, United States, 939308. https://doi.org/10.1117/12.2075655

M. Kampel and R. Sablatnig. 2000. Color classification of archaeological fragments. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, Vol. 4. IEEE Comput. Soc, Barcelona, Spain, 771–774. https://doi.org/10.1109/ICPR.2000.903031

Roman Klokov and Victor Lempitsky. 2017. Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Venice, 863–872. https://doi.org/10.1109/ICCV.2017.99

Shiyi Lan, Ruichi Yu, Gang Yu, and Larry S. Davis. 2018. Modeling Local Geometric Structure of 3D Point Clouds using Geo-CNN. *arXiv:1811.07782 [cs]* (Nov. 2018). http://arxiv.org/abs/1811.07782 arXiv: 1811.07782.

Jake Lever, Martin Krzywinski, and Naomi Altman. 2017. Principal component analysis. *Nature Methods* 14, 7 (July 2017), 641–642. https://doi.org/10.1038/nmeth.4346

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018. PointCNN: Convolution On $\mathcal{X}$-Transformed Points. *arXiv:1801.07791 [cs]* (Nov. 2018). http://arxiv.org/abs/1801.07791 arXiv: 1801.07791.

Qi Li-Ying and Wang Ke-Gang. 2010. Kernel fuzzy clustering based classification of Ancient-Ceramic fragments. In *2010 2nd IEEE International Conference on Information Management and Engineering*. 348–350. https://doi.org/10.1109/ICIME.2010.5477818

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv:1608.03983 [cs, math]* (May 2017). http://arxiv.org/abs/1608.03983 arXiv: 1608.03983.

J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations.

Lorenzo Mantovan and Loris Nanni. 2020. The Computerization of Archaeology: Survey on Artificial Intelligence Techniques. *SN Computer Science* 1, 5 (Aug 2020). https://doi.org/10.1007/s42979-020-00286-w

Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. 2018. Geodesic convolutional neural networks on Riemannian manifolds. *arXiv:1501.06297 [cs]* (June 2018). http://arxiv.org/abs/1501.06297 arXiv: 1501.06297.

Daniel Maturana and Sebastian Scherer. 2015. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Hamburg, Germany, 922–928. https://doi.org/10.1109/IROS.2015.7353481

Jonah C. McBride and Benjamin B. Kimia. 2003. Archaeological Fragment Reconstruction Using Curve-Matching. In *2003 Conference on Computer Vision and Pattern Recognition Workshop*, Vol. 1. 3–3. https://doi.org/10.1109/CVPRW.2003.10008

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. 2017. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Honolulu, HI, 5425–5434. https://doi.org/10.1109/CVPR.2017.576

Open3D. [n.d.]. open3d.geometry.sample_points_poisson_disk — Open3D 0.7.0 documentation. http://www.open3d.org/docs/0.7.0/python_api/open3d.geometry.sample_points_poisson_disk.html#open3d-geometry-sample-points-poisson-disk

Cecilia Ostertag and Marie Beurton-Aimar. 2020. Using Graph Neural Networks to Reconstruct Ancient Documents. *arXiv:2011.07048 [cs]* (Nov. 2020). http://arxiv.org/abs/2011.07048 arXiv: 2011.07048.

Geoffrey Oxholm and Ko Nishino. 2013. A flexible approach to reassembling thin artifacts of unknown geometry. *Journal of Cultural Heritage* 14, 1 (2013), 51–61. https://doi.org/10.1016/j.culher.2012.02.017

Huy Phan, Martin Krawczyk-Becker, Timo Gerkmann, and Alfred Mertins. 2017. DNN and CNN with Weighted and Multi-task Loss Functions for Audio Event Detection. *arXiv:1708.03211 [cs]* (Oct. 2017). http://arxiv.org/abs/1708.03211 arXiv: 1708.03211.

Roberto Pierdicca, Marina Paolanti, Francesca Matrone, Massimo Martini, Christian Morbidoni, Eva Savina Malinverni, Emanuele Frontoni, and Andrea Maria Lingua. 2020. Point Cloud Semantic Segmentation Using a Deep Learning Framework for Cultural Heritage. *Remote Sensing* 12, 6 (March 2020), 1005. https://doi.org/10.3390/rs12061005

Charles R. Qi, Hao Su, Matthias NieBner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. 2016. Volumetric and Multi-view CNNs for Object Classification on 3D Data. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Las Vegas, NV, USA, 5648–5656. https://doi.org/10.1109/CVPR.2016.609

T Rabbani, F.A. Heuvel, and George Vosselman. 2006. Segmentation of point clouds using smoothness constraint. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 36 (01 2006).

Nada Rasheed, Awfa Dakheel, Wessam Nadoos, Maysoon Khazaal, and Md Jan Nordin. 2017. Classification Archaeological Fragments into

Groups. *Research Journal of Applied Sciences, Engineering and Technology* 14 (09 2017), 324–333. https://doi.org/10.19026/rjaset.14.5072

Nada A. Rasheed and Md Jan Nordin. 2020a. Classification and reconstruction algorithms for the archaeological fragments. *Journal of King Saud University - Computer and Information Sciences* 32, 8 (Oct. 2020), 883–894. https://doi.org/10.1016/j.jksuci.2018.09.019

Nada A. Rasheed and Md Jan Nordin. 2020b. Reconstruction algorithm for archaeological fragments using slope features. *ETRI Journal* 42, 3 (June 2020), 420–432. https://doi.org/10.4218/etrij.2018-0461

Sebastian Ruder. 2017. An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]* (June 2017). http://arxiv.org/abs/1609.04747 arXiv: 1609.04747.

Carlos Sanchez-Belenguer and Eduardo Vendrell-Vidal. 2014. An efficient technique to recompose archaeological artifacts from fragments. In *2014 International Conference on Virtual Systems & Multimedia (VSMM)*. IEEE, Hong Kong, Hong Kong, 337–344. https://doi.org/10.1109/VSMM.2014.7136671

F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (Jan. 2009), 61–80. https://doi.org/10.1109/TNN.2008.2005605

Connor Shorten and Taghi M. Khoshgoftaar. 2019. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data* 6, 1 (Dec. 2019), 60. https://doi.org/10.1186/s40537-019-0197-0

Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. *arXiv:1704.02901 [cs]* (Aug. 2017). http://arxiv.org/abs/1704.02901 arXiv: 1704.02901.

Patrick Smith, Dmitriy Bespalov, Ali Shokoufandeh, and Patrice Jeppson. 2010. Classification of archaeological ceramic fragments using texture and color descriptors. 49 – 54. https://doi.org/10.1109/CVPRW.2010.5543523

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (06 2014), 1929–1958.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Santiago, Chile, 945–953. https://doi.org/10.1109/ICCV.2015.114

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 1139–1147. http://proceedings.mlr.press/v28/sutskever13.html

Trimesh. [n.d.]. trimesh.sample — trimesh 3.9.20 documentation. https://trimsh.org/trimesh.sample.html#sample-py

Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. 2019. Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data. *arXiv:1908.04616 [cs]* (Aug. 2019). http://arxiv.org/abs/1908.04616 arXiv: 1908.04616.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *arXiv:1710.10903 [cs, stat]* (Feb. 2018). http://arxiv.org/abs/1710.10903 arXiv: 1710.10903.

Cheng Wang, Ming Cheng, Ferdous Sohel, Mohammed Bennamoun, and Jonathan Li. 2019a. NormalNet: A voxel-based CNN for 3D object classification and retrieval. *Neurocomputing* 323 (Jan. 2019), 139–147. https://doi.org/10.1016/j.neucom.2018.09.075

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019b. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics* 38, 5 (Nov. 2019), 1–12. https://doi.org/10.1145/3326362

E. Widyaningrum, M. K. Fajari, R. C. Lindenbergh, and M. Hahn. 2020. TAILORED FEATURES FOR SEMANTIC SEGMENTATION WITH A DGCNN USING FREE TRAINING SAMPLES OF A COLORED AIRBORNE POINT CLOUD. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2020 (Aug. 2020), 339–346. https://doi.org/10.5194/isprs-archives-XLIII-B2-2020-339-2020

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yueguan Yan, Haixu Yan, Junting Guo, and Huayang Dai. 2020. Classification and Segmentation of Mining Area Objects in Large-Scale Spares Lidar Point Cloud Using a Novel Rotated Density Network. *ISPRS International Journal of Geo-Information* 9, 3 (March 2020), 182. https://doi.org/10.3390/ijgi9030182

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *arXiv:1411.1792 [cs]* (Nov. 2014). http://arxiv.org/abs/1411.1792 arXiv: 1411.1792.

Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. 2019. PointWeb: Enhancing Local Neighborhood Features for Point Cloud Processing. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Long Beach, CA, USA, 5560–5568. https://doi.org/10.1109/CVPR.2019.00571

Pengbo Zhou, Kegang Wang, and Wuyang Shui. 2011. Ancient Porcelain Shards Classifications Based on Color Features. In *2011 Sixth International Conference on Image and Graphics*. 566–569. https://doi.org/10.1109/ICIG.2011.190