Siamese DGCNN for the classification of adjacent archaeological fragments.

Alessio Parmeggiani[1,*], Alessio Sfregola[1,*]

[1]*Sapienza University of Rome, Italy*

Abstract

In this paper we propose a novel approach for the problem of determining whether two fragments belonging to the same broken object were adjacent before the fragmentation. This work is developed in collaboration with "Ente Parco Archeologico dei Fori Imperiali" and is part of a bigger project that aims at tackling the task of the reconstruction of fragmented ancient artifacts. The innovation we propose consists in combining Siamese neural networks with DGCNNs using point clouds to perform a classification task. This model takes as input two point clouds sampled from two synthetically produced archaeological fragments and outputs the probability that they were adjacent in the original object.

1. Highlights

- Use of custom dataset to simulate fragments of ancient artifacts
- Novel approach using Siamese Neural Networks with a DGCNN for point clouds
- Use of normals as additional features for point clouds
- Tested performances of different architectures and identified the best one
- Reached satisfying performances (90% F1-score)

2. Introduction

Archaeology is the study of human history through the analysis of artifacts and other physical remains. It's a field whose importance has been recognized only in recent centuries, and today understanding the political and societal state of ancient societies is considered of fundamental importance. To have a clearer perception of these issues, the analysis of what past civilizations left us is crucial.

The problem is that, as one would imagine, these remains have been left exposed to the elements for hundreds if not thousands of years, and this has led to them accumulating dirt, breaking and fragmenting, leaving archaeologists to a huge load of time consuming work that needs to be made in order to bring these artifacts back to their original conditions, or at least in a shape that allows us to analyze them.

In this regard, one of the most time consuming tasks is the reconstruction of broken objects. This is a practice that may take weeks or even months of hard work for a single relic, slowing down the advancements that can be made in the field. This is why in the latest years, researchers have been trying to find modern solutions to this problem, and this is where our work fits.

Our task is part of a bigger project in collaboration with the Institution of "Ente Parco Archeologico dei Fori Imperiali". This project involves the development of a deep learning approach to help researchers in reconstructing solid objects from their fragments. It's a very complex task, so we focused on a small subproblem, continuing the work made by Nardelli and Valleriani, who developed a new technique to detect whether some fragments of the original objects were internal or external.

We took their research one step forward and tackled the problem of determining whether two fragments were adjacent in the original artifact or not. Various approaches have been developed over time to tackle similar problems, such as extrapolating visual information from 2D images, or considering the whole 3D shape of the fragments as independent models. In the case of Roman relics, that can date back up to almost 3 thousand years ago, tiny fragments or internal fragments, most of the visual elements that can help in reconstructing the relics are not available, so we can only rely on the actual shapes of the fragments.

To do this, we needed a dataset containing a large amount of fragments of ancient relics, with data about adjacent fragments, but unfortunately, to our knowledge, a dataset of this kind doesn't exist. That's why we used a synthetic dataset provided to us by Prof. Marco Serra, which contains a large amounts of 3D models representing fragments of three different types of objects: spheres, icospheres and cubes. When working with 3D models it's common practice to transform them in point clouds before feeding them to neural networks, and that's exactly what we did. Taking inspiration by the work of Nardelli and Valleriani we also extracted normals from the 3D models.

The novelty of our work resides in the architecture

* Both authors contributed equally to this research

parmeggiani.1799327@studenti.uniroma1.it (A. Parmeggiani);
sfregola.1798423@studenti.uniroma1.it (A. Sfregola)

we used. We built a modified Siamese network, such that the two point clouds were fed to two identical DGCNNs. Then, the resulting vectors would be concatenated, joined with the normals, and fed to a fully connected network which output a single value, corresponding to the probability of the two fragments being adjacent in the original artifact. A Siamese neural network is an artificial neural network in which two identical models, that share the same weights, take two inputs and work in parallel to output some vectors, and are usually used to compare similar instances (see section 4.1). A DGCNN is an artificial neural network developed by Wang et al. [1] created as an evolution of PointNet [2], to extract local features from point clouds (see section 4.2).

To our knowledge, the use of these two architectures in conjunction has never been attempted before neither for a classification task, neither on point clouds, so we think that our work can bring interesting insights in the field. We tested our model with a large number of different hyperparameters, the results can be seen in section 5. Other than this, we also compared the performance of our model with a simpler architecture (PointNet, see section 4.6) we used as a baseline. Overall, we consider ourselves satisfied with the results we obtained.

3. Related work

For the analysis of fragments of objects in the archaeological field there are two main approaches that have been studied in these years, algorithmic procedures and deep neural networks.

3.1. Algorithmic approaches

The first approach consists in the research of mathematical procedures to analyze the structure of the fragments, for example Elnaghy et al. [3] developed a procedure to segment fragments into facets by estimating the angle of fracture, estimate the fracture lines and then classify each facet as fracture or skin. Another example could be seen in the work of Filippas et al. [4] that developed an algorithmic procedure to perform our same work, detecting is two fragments are adjacent. They used a novel algorithm that works on point cloud and work on global matching and partial matching of the fragments. A complete reassembling pipeline has been proposed some years ago by Huang et al. [5], their algorithm is able to detect features of the objects, find matching pairs of fragments and use graph optimization for the multi-piece matching. All these approaches rely on user defined thresholds that could affect the quality of the final results, moreover the example proposed by the previous paper works with a low amount of fragments and their dimension is fairly big, i.e. they contain a lot of features.

3.2. Neural networks

More recently the introduction of deep neural networks in the study of 3D objects has obtained excellent results. While the use of Siamese neural networks with DGCNN in the archaeological field is a novelty, the use of point clouds to approximate 3D objects and its analysis with neural networks is an approach that has been used in a lot of works. The main difficulty that arises in using deep neural networks for point cloud data is that this type of data doesn't have a regular structure, while standard models, for example based on a CNN architecture, rely on an underlying regular structure like in images.

In the point clouds the positions of the points are continuously distributed in the space, and any permutation of their ordering does not change the spatial distribution.

Initially most researchers focused on transforming point clouds to regular data such as 3d voxel grids or a collection of images.

One common approach to process point cloud data using deep learning models is to first convert raw point cloud data into a volumetric representation,a set of occupancy grids, and then use a CNN architecture to do predictions on this 3d grid. This approach,used for example by Maturana et al.[6], however, usually introduces quantization artifacts and uses a lot of memory, making it difficult to go to capture small features.

An alternative approach to the volumetric representation of the point cloud is given by multi view CNNs, for instance in the case of the work by Qi et al.[7]. First, a 3D shape is rendered into multiple images then image features are extracted for each view using convolutional layers and he features are combined across views through a pooling layer, followed by fully connected layers. However with this approach is difficult to extend the work to other tasks, such as point classification, in fact here we don't consider the points of the point cloud anymore but a different representation.

One of the first architecture that takes into account directly the irregularity of the point clouds is PointNet, developed by Qi et al.[2]. As opposed to the previous approaches, PointNet can directly manipulate the point cloud without the use of an intermediate regular representation. This architecture achieves permutation invariance by operating on each point independently and accumulating later the related features.

By design PointNet does not capture local structures induced by the metric space points live in, limiting its ability to recognize fine-grained patterns and generalizability to complex scenes. Qi et al.[8] so improved the previous architecture based on the idea of dividing the point cloud into local regions and extract the local features of each region using the same architecture of PointNet, and then this features are grouped to be processed again in order to get higher level features. This

new model, called PointNet++, improved the results of PointNet but its performances have been surpassed by graph based models.

Another approach in the handling of point clouds is to consider the structure as a graph and using a neural network that can work on data in this form, as in the Graph Neural Network (GNN) architecture developed by Scarselli et al.[9] The GNN is designed to be able to handle all types of graphs, cyclic, directed and undirected.

The GNNs have been used also with point clouds, in the case of Point-GNN by Shi et al.[10].

Our architecture derives from the DGCNN model by Wang et al. that improve the GNN and PointNet models by introducing the EdgeConv operation and the dynamic graph update, to generate edge features that capture local geometric structure. How this type of model is built is explained in detail in section 4.2.

Valleriani and Nardelli used this type of network to classify fragments as external or internal.

Regarding the use of two inputs at the same time, one of the most successful architecture is the Siamese network that uses two models that share the same weights. Navarin et al.[11] used an approach that combines a Siamese network and the DGCNN model for the unsupervised pre-training using as input graphs of various chemical structures.

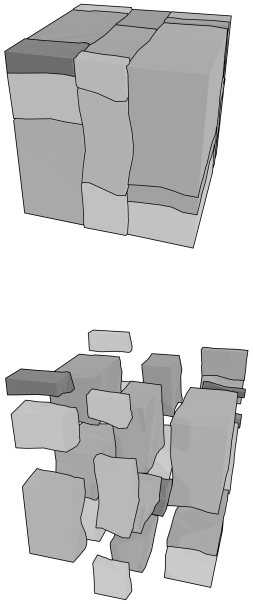


Figure 1: Fragments generated from a cube using 3 cuts for each axis,for a total of 27 fragments.

4. Dataset

To train the model we need a dataset of fragments and information about which of these objects were adjacent in the original artifact. Prof. Marco Serra provided us with a dataset that was generated using three types of object: cubes, spheres and icospheres that were cut many times along the X,Y or Z axis. The total number of fragments for each model would be equal to $x \cdot y \cdot z$, where x, y, z are the numbers of cuts along the X,Y or Z axis respectively. The cut are not perfectly straight and there is noise applied so that the fragments generated can be similar to real fragments of a bigger object. (see Figure 1 for an example.).

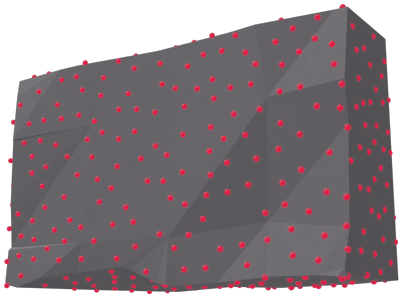
One of the problems of the dataset is that is highly unbalanced due to the nature of the work, in fact with n fragments, we would have 2^n possible pairs and the number of adjacent pairs would be much smaller than the number of non adjacent pairs. The first dataset had 48 objects subdivided into 9210 fragments, so we would have over some millions pairs but only 90000 of those are adjacent pairs. Moreover each pair would appear two times, given two fragments I and J, we would have the pair (I,J) and the pair (J,I) but we decided to include only one appearance of each pair to avoid having the same data repeated two times.

During the creation of the dataset, to avoid having an unbalanced dataset we decided, for each set of fragments belonging to the same object, to take only $\alpha \cdot N$ non adjacent pairs, and all the N adjacent pairs. So with $\alpha = 1$ we would have a perfectly balanced dataset.

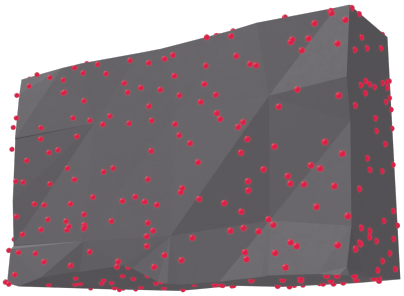
To build the dataset, for each set of fragments, we create the point cloud of each fragment, we generate all the possible pairs of fragments and we choose only a subset of them as specified above. These pairs will be split between the train, validation and test dataset with a user-defined probability, in our case 70% of the samples will be used for training, while the test set and validation set will contain each 15% of the samples.

To generate each point cloud we use the Python library Open3d and a function that allows to select a fixed amount of random points on the surface of a model using the Poisson disk method [12] where each point has approximately the same distance from the other. It would also be possible to sample points uniformly but we wanted to cover the most amount of surface possible for each model (see Figure 2 for a comparison.).

We noticed that each fragment was in the same position of the original model, so adjacent fragments would be closed to each other, introducing some form of bias. To avoid this, we decided to move each point cloud to the origin, so our model would learn independently from the position and look only at the geometric features of a fragment. We decided not to normalize the points of



(a) Points sampled using the Poisson algorithm.



(b) Points sampled uniformly.

Figure 2: Sampling of the points for the point cloud using two algorithms.

each point cloud because the difference in the size is an important feature that must be considered during the training.

Using Open3D we also extracted from the original model the normals to use them later in the model.

5. Our approach

As mentioned above, we used a Siamese network in which the twin models are DGCNNs, the output of which is concatenated, joint with normals, and passed through a fully connected network to output a probability that the pair of point clouds given as input were adjacent.

5.1. Siamese Neural Network

A *Siamese neural network* is an architecture, first introduced by Bromley et al. [13], in which two twin subnetworks share the same weights, accept distinct inputs and the two outputs are joined at the end. These kinds of networks are usually used to compute the similarity between two inputs. Usually the inputs are fed into the two twin networks, some output feature vectors are computed, and then some loss function is used (such as contrastive loss), to determine how similar the two samples are. In our case, we won't use them to perform this kind of task. Instead, we'll take the feature vectors, concatenate them with their corresponding vector of normals, and then feed everything to a fully connected neural network that works as a classifier, to determine whether the two point clouds were adjacent in the original object or not. Regarding the two twin subnetworks, in general there are no restriction on the models that can be used. In our case, we chose to use *DGCNNs*.

5.2. DGCNN

A *DGCNN* (Dynamic Graph Convolutional Neural Network) is an architecture that can be seen as an intersection between PointNet and conventional CNNs, allowing for the exploitation of local geometric structures of graphs. The originality of this work resides in the fact that graphs, unlike images, usually don't have an underlying grid, so the usage of operations such as convolutions is not possible. For this reason, the authors defined a new operation called *EdgeConv* (edge convolution), an operation similar to convolution that can be used on unstructured data. Other than this, Wang et al. [1] also noticed that considering the whole graph as a constant structure was a constraint that could be removed by dynamically updating the graph after each layer of the network, allowing for different representations of the graph in the input space and in feature space. The ability of *dynamically* learning the structure of the graph is what gives the name to this architecture.

5.3. Edge convolution

Suppose we have an F-dimensional point cloud with n points, denoted by $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\} \in \mathbb{R}^F$. For simplicity, let's assume $F = 3$, as in a point cloud sampled on the surface of a 3D object, but it could be any number of dimensions. We build a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges, from our point cloud (e.g. by taking the k-nearest neighbors for each point). define edge features as $e_{i,j} = h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$, where $h_\Theta : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^{F'}$ is a nonlinear function with a set of learnable parameters Θ. The EdgeConv operation for the vertex $\mathbf{x}_i$ is defined as:

$$\mathbf{x}'_i = \underset{j:(i,j)\in\mathcal{E}}{\square} h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$$

that is, the application of a channel-wise symmetric aggregation operation $\square$ (such a summation or max operation) on the edge features associated with all the edges that start from x_i (see Figure 3). The similarity between this operation and the classic convolution of a grid of elements around a central pixel in an image is glaring.

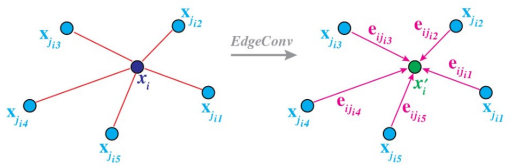


Figure 3: The EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

5.4. Dynamic graph update

As we said, another key innovation of Wang et al. is the choice of recomputing the whole graph after each layer. This allows for the receptive field to be as large as the diameter of the point cloud, while being sparse. The major advantage is letting the architecture learn how to build the graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$ used in each layer, rather than taking it as a fixed constraint. To build it, we simply compute a pairwise distance matrix in the feature space and take the k closest points for each vertex of $\mathcal{G}^{(l-1)}$.

5.5. Properties

The application of the aforementioned techniques allow for some interesting properties.

Permutation Invariance. Defining EdgeConv such that it involves the use of a symmetric aggregation function means that the order in which the edges are processed is completely irrelevant, meaning that the operation is permutation invariant.

Translation Invariance. The EdgeConv operation is not actually invariant to translation, but we can choose some families of edge functions $h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$ such that it's easy to expose the part of the function which is translation-dependent and optionally disable it.

5.6. Our model

After seeing the structure of the networks we started with, let's see how we modified them to suit our task.

Given the nature of our problem, we needed to feed the network with two different point clouds. We explored different possibilities, such as concatenating the point clouds and considering them as one input sample, or creating a new point cloud consisting of the two elements side by side, but we ended up with using a completely different approach.

In the end we settled for a Siamese network that takes as input two distinct point clouds and feeds them to two identical DGCNNs that also share the same weights, and output two distinct feature vectors. These vectors are then concatenated, first with their respective normals and then with each other, to form a unique feature vector that is fed to a linear classifier.

In Figure 4 we can see the main components of our model. The first point cloud is given as input to the first EdgeConv layer. Here, as explained in section 4.2, edge features are computed and then aggregated. The output is then fed into the second EdgeConv layer, and the same process is performed for all 4 layers. The four vector resulting from these intermediate steps are concatenated and passed through a final convolutional layer and max-pooling and average-pooling are performed. Then, we take the normals of the points that make up the point cloud and concatenate them with the feature vector resulting from the previous steps,then we feed it into a linear layer, a batch normalization layer and an activation function (leaky ReLU). Lastly, we perform some dropout to reduce overfitting. Now, the same procedure is repeated for the second point cloud, and at the end, we concatenate both of them in a single vector. As a final step, this vector is given as input to a linear classifier, composed by 2 components, each of which consists of a linear layer, a batch normalization layer, and an activation function (leaky ReLU). Dropout is then applied to it, and finally, the result is given to a last linear layer, which outputs a single value, the final result.

This is the base architecture of our model. We then performed some modifications to see if the performances could be further increased.

5.6.1. Modifications

The model described above is the one that produced the best results, but we also tried different approaches.

We tried changing the number of layers in the final classifier, increasing or decreasing them, and we also tried using normals differently, such as not feeding them into a linear layer after concatenating them with the features of the corresponding point cloud, but concatenating the two point clouds with respective normals together into one big vector and feeding it directly into the final classifier. This meant using one less linear layer, that we tried adding to the classifier, but this process did not increase our performances. Some of the architectures we tried can be seen in Figure 5

5.7. PointNet

After extensively testing our model we decided to see how its performances resulted when compared with a baseline. For that purpose, we chose to use PointNet [2], an architecture similar to the DGCNN we used, but much simpler. In PointNet, the points that make up a point cloud are processed identically and independently, as opposed to DGCNNs, in which when we analyze a

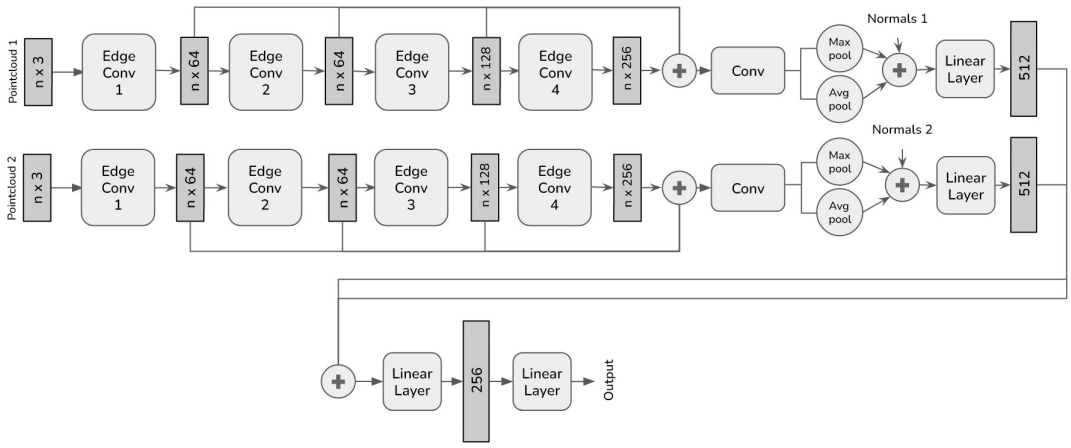


Figure 4: The architecture that obtained overall the best results.

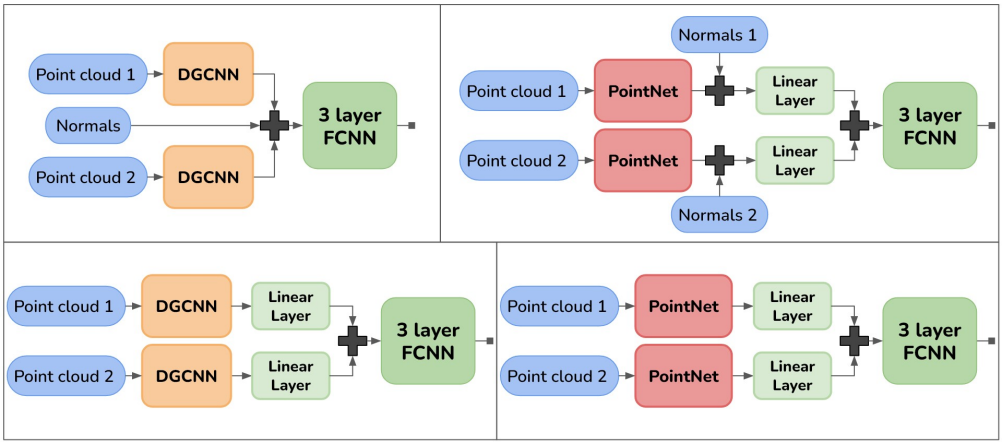


Figure 5: Some of the alternative architectures we tried

point we also always analyze its neighborhood, which allows for the utilization of local features. All the work done in PointNet revolves around the use of a symmetric aggregation function, max pooling, thanks to which the network learns the most informative points in the cloud. This approach is similar to what is done in DGCNNs, but the concept of dynamically updating the network is completely absent in PointNet. We chose this architecture as a baseline because in the last couple of years it has been giving consistently good results in tasks regarding point cloud classification similar to ours, being a good compromise between complexity in the implementation and performance. (e.g. [14])

6. Results

For the training we found out that Adam[15] greatly outperforms the other optimizers that we tried, and we used a scheduler for the learning rate to improve the training trend; we found out that using a cosine annealing schedule produces good results but the learning rate became too small, and thus the training slow down. We settled with a scheduler that reduces the learning rate by some factor after a certain amount of epochs passed without an improvement in the loss function in order to escape from possible plateaus. As a loss function we used the standard Binary Cross Entropy.

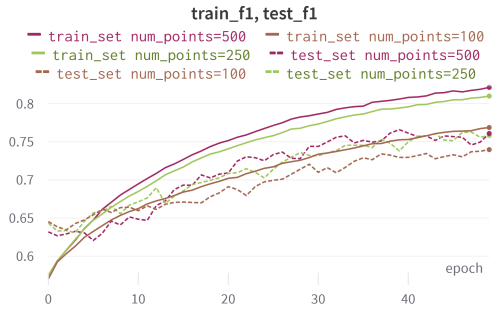


Figure 6: F1 score on the same model using different number of points for each point cloud.

6.1. Choosing the most important parameters

The most important parameter that we can set is how many points to consider for each point cloud. Having a lot of points will allow the model to learn all the necessary features of a point cloud but will slow down a lot the training. The model could also have too many weights to update and this could lead to underfitting. On the other hand, having fewer points will allow the training to run faster, to converge in less epochs, and it will also reduce overfitting, in fact, with less weights the model will be able to generalize better and obtain better predictions on new data. The problem of having too few points is that they are sampled with the same density on all the surface of the 3D model and this could generate a point cloud that is missing some key features of the original fragment, such as particular edges or little ridges; especially in the case of large fragments,in fact a point cloud generated from it would be more sparse than one generated from a little fragment.

We tested a basic model using 100,250 or 500 points for each fragment and we noticed that after the 250 threshold the performances improve only a bit while the computation time doubles, so we avoided testing also using more points. The number of points we decided to use is 250 for each fragment, this allows us to obtain good performances that are only slightly worse than using 500 points and an acceptable training time. A comparison of the results can be seen in Figure 6.

Another important parameter is the number of neighbours to consider for each point during the edge convolution step for the construction of the local graph. As expected the more points we consider the better the results, but increasing this parameter also increased a lot the computation time. We found out that there is a big improvement between using 20 neighbours or 30 neighbours (up to 1.8% on the test accuracy) but between 30 and 50 neighbours the accuracy improved by only 0.2 but overall the training is less stable and the runtime almost doubled (8.5 minutes against 15.8 minutes per epoch),see Table 1. So we settled with K=30 and continued to use this parameter from now on.

Table 1
Accuracy and F1 score for different values for the amount of neighbour of each point to consider(K) after 30 epochs.

K	test accuracy	test F1
5	0.69	0.658
10	0.695	0.688
20	0.72	0.708
30	0.732	0.722
50	0.734	0.711

Table 2
Accuracy and F1 score for different values of dropout after 40 epochs.

dropout	point dropout	train F1	test F1
0	0	0.94	0.74
0.3	0	0.83	0.74
0.5	0	0.77	0.71
0.3	0.2	0.79	0.75
0.3	0.5	0.75	0.70

6.2. Reducing overfitting

To avoid and reduce the overfitting we introduced dropout between layers to zero out some nodes at each step; another technique that we used was using a special type of dropout: instead of setting to zero random nodes in the network at each step, we set to zero random points (so both the x,y and z coordinates) of each point cloud at the start of each training step. We found that using 0.2 as a value for this point dropout (i.e. 20% of the points are dropped) gives the best results (see Table 2).

We tested the methods on one of the model that gave us the worst overfitting to check which approach worked better and if we could use both together. We noticed that using both methods reduces a lot the overfitting lowering the training accuracy, without improving or decreasing the results on the test set, allowing for a larger room of improvement of the performances in the future.

To reduce the overfitting we also tried introducing a random translation for each point cloud, this could help the model to focus on geometric features rather than the actual position of a point and we noticed that this approach reduces a lot the overfitting and leads to an overall more stable training.

We also tested the shuffling of all the points of a point cloud to determine if the results depend on how the dataset is built, but as expected there are no significant

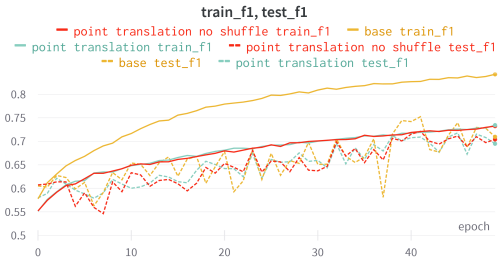


Figure 7: Results using a base model (yellow), translating point clouds (red), translating point clouds and shuffling the points (cyan).

changes in the training performances. This is due to the fact that the EdgeConv operation is built to be permutation invariant, i.e the order in which the edges are processed is irrelevant. See Figure 7 for details.

6.3. Model architecture parameters

We also searched for the optimal dimension of the output of the DGCNN model and the optimal number of nodes for the final classification layers, but after trying different combinations we found that the results didn't change much. This could indicate that the model has reached it's maximum potential, so to improve the performances we need to increase the amount of data on the training. To do this, our only option would be to use an unbalanced dataset, given that we already used all the available adjacent pairs. So we created another dataset that contains 3 times non-adjacent pairs with respect to adjacent ones, and to avoid biased results on the non-adjacent class, we used weights during the computation of the loss to give more importance to the prediction of the adjacent class . The performances surely improved, the overfitting reduced a lot but the F1 score decreased a bit, due to the decrease in recall.

During the creation of the dataset we considered only one of the two possible combinations for each pair, now we try to see if considering both the possible combinations (i.e. for two fragments I and J, considering both the pair (I,J) and the pair (J,I)) can improve the performances. Overall the performances improved, the accuracy remained almost the same but there is a big improvement in the f1 score, given mainly by the increase in the recall score. This could mean that a bigger dataset could greatly increase the performances.

6.4. Final results

Prof. Marco Serra provided us with a new dataset that contains double the number of elements, so 90000 adjacent pairs without considering both the possible combinations for each pair, and using this new dataset the performances increased a lot but the training needed a lot more epochs to converge.

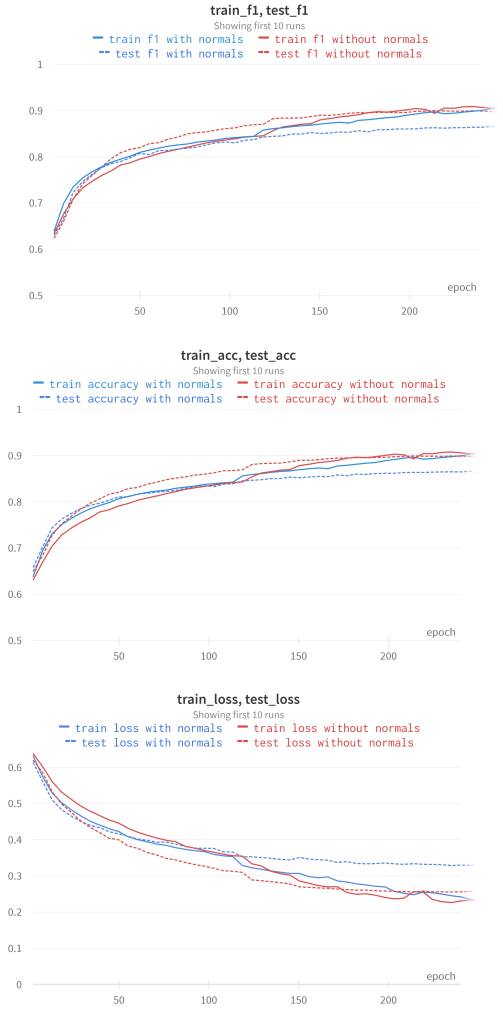


Figure 8: F1,accuracy and loss plots of the model that obtained the best results with and without the use of the normals.

With this new dataset we get the best results, in which the F1 score, reaches 0.901 on the test set. From the plot is possible to see how all the improvements discussed before and the bigger dataset contributed to having little to no overfitting.

We also noticed that having more data reduced the efficacy of the normals,in fact this time the introduction of the normals didn't improve the performances, as if the model reached its maximum capabilities with the increase of the dimension of the dataset and the addition of new data led to some overfitting because of the need

for more nodes to handle the increased dimension of the input. These results can be seen in Figure 8.

6.5. Baseline comparison

Given that we didn't find any other work in literature that could be directly compared to ours, we chose to build another simpler baseline model, to see if the complex architecture we used gives significantly better results. To do this, we used PointNet as twin model in the Siamese network. We noticed that our original results perform a lot better than a partially tuned PointNet model, in fact it reaches only 0.72 on the test set and feature also a lot of overfitting. This means that the exploitation of local features and the dynamic update used by DGCNNs are a significant improvement over simpler models that operate solely on independent points of the point clouds.

7. Conclusions

In this research we propose a new model that combines the Siamese networks architecture and the DGCNN model for the classification of adjacent point clouds, using also the normals as additional features. As far as we know, this kind of architecture is a novelty in the field. The final goal of this project is to apply these technologies in the archaeological field, to help with the reconstruction of ancient artifact from its fragments. Our model is trained on a custom synthetic dataset that simulates different types of fragments, and after several trials with different hyperparameters and different configurations of the network, we obtained performances we are satisfied of, reaching an F1 score of 90%. Comparing our results with PointNet as a baseline, we can state that our work is significant, and can probably obtain even better results with a bigger dataset.

References

[1] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, Dynamic graph cnn for learning on point clouds, 2018. URL: https://arxiv.org/abs/1801.07829. doi:10.48550/ARXIV.1801.07829.

[2] C. R. Qi, H. Su, K. Mo, L. J. Guibas, Pointnet: Deep learning on point sets for 3d classification and segmentation, CoRR abs/1612.00593 (2016). URL: http://arxiv.org/abs/1612.00593. arXiv:1612.00593.

[3] H. ElNaghy, L. Dorst, Geometry based faceting of 3d digitized archaeological fragments, in: 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), 2017, pp. 2934–2942. doi:10.1109/ICCVW.2017.346.

[4] A. Georgopoulos, D. Filippas, Development of an algorithmic procedure for the detection of conjugate fragments, volume II-5/W1, 2013.

[5] Q. Huang, S. Flöry, N. Gelfand, M. Hofer, H. Pottmann, Reassembling fractured objects by geometric matching, ACM Trans. Graph. 25 (2006) 569–578. doi:10.1145/1141911.1141925.

[6] D. Maturana, S. Scherer, Voxnet: A 3d convolutional neural network for real-time object recognition, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 922–928. doi:10.1109/IROS.2015.7353481.

[7] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, L. J. Guibas, Volumetric and multi-view cnns for object classification on 3d data, CoRR abs/1604.03265 (2016). URL: http://arxiv.org/abs/1604.03265. arXiv:1604.03265.

[8] C. R. Qi, L. Yi, H. Su, L. J. Guibas, Pointnet++: Deep hierarchical feature learning on point sets in a metric space, CoRR abs/1706.02413 (2017). URL: http://arxiv.org/abs/1706.02413. arXiv:1706.02413.

[9] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (2009) 61–80. doi:10.1109/TNN.2008.2005605.

[10] W. Shi, R. Rajkumar, Point-gnn: Graph neural network for 3d object detection in a point cloud, CoRR abs/2003.01251 (2020). URL: https://arxiv.org/abs/2003.01251. arXiv:2003.01251.

[11] N. Navarin, D. V. Tran, A. Sperduti, Pre-training graph neural networks with kernels, CoRR abs/1811.06930 (2018). URL: http://arxiv.org/abs/1811.06930. arXiv:1811.06930.

[12] C. Yuksel, Sample elimination for generating poisson disk sample sets, Computer Graphics Forum 34 (2015) 25–32. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12538. doi:https://doi.org/10.1111/cgf.12538.

[13] J. Bromley, J. Bentz, L. Bottou, I. Guyon, Y. Lecun, C. Moore, E. Sackinger, R. Shah, Signature verification using a "siamese" time delay neural network, International Journal of Pattern Recognition and Artificial Intelligence 7 (1993) 25. doi:10.1142/S0218001493000339.

[14] H. Gao, G. Geng, Classification of 3d terracotta warrior fragments based on deep learning and template guidance, IEEE Access 8 (2020) 4086–4098. doi:10.1109/ACCESS.2019.2962791.

[15] D. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (2014).