SAPIENZA
UNIVERSITÀ DI ROMA

Point cloud transformers for 3D fragment matching

Faculty of Information Engineering, Informatics, and Statistics
Master in Data Science

Alessandro Sottile
ID number 1873637

Advisor
Prof. Simone Scardapane

Co-Advisor
Alessandro Baiocchi

Academic Year 2023/2024

Point cloud transformers for 3D fragment matching
Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: sottile124@gmail.com

To my dad, my mother and my sister,
who never stopped believing in me.

Contents

Chapter 1

Introduction

In recent years, machine learning has experienced extraordinary growth and has expanded into a wide range of sectors and domains. This is largely attributed to continuous advancements in research and technology development, as well as the increasing accessibility to data and computational resources. Each generation of Graphics Processing Unit (GPU) becomes increasingly powerful, enabling the training of heavier and more complex models. These models prove particularly effective and valuable when a large amount of data is available. They excel at identifying complex patterns in datasets that are often challenging for the human eye to discern.

A sector that is beginning to benefit from machine learning is archaeology. The synergy between archaeology and technology is a constant feature of the discipline, which has witnessed significant innovations over time, such as the development of the carbon-14 dating system in the 1940s, allowing for precise determination of the age of artifacts [1]. Another example is the use of satellite data, which has revealed hidden or submerged archaeological sites, opening new perspectives in research [2]. In the forensic field, 2021 saw the remarkable ability to recreate the faces of mummies [3]. Furthermore, the combination of 3D scanners and drones has enabled the creation of detailed digital models of archaeological sites and artifacts, enhancing the cataloging and study of the findings [4].

During their investigations, archaeologists often find themselves discovering ancient objects, some dating back thousands of years. These precious artifacts bear the unmistakable signs of the time that has passed, requiring immense expertise and mastery from the team members who come into contact with them. Frequently, these artifacts are found fragmented, adding an additional layer of difficulty to the challenge. As the number and complexity of fragments increase, archaeologists need greater skill. Moreover, they may find themselves working in situations where

fragments belong to different statues. In this context, the use of deep learning, leveraging modern technologies that enable high-precision 3D scanning of these artifacts, and thus obtaining point clouds, could be a promising solution. The integration of these algorithms into archaeology represents a significant evolution. This technology aims not to completely replace human capabilities but rather to assist and guide archaeologists in their research. Point clouds, as the term suggests, are a representation of points sampled from the surfaces of a 3D object, providing both local and global information about the object. Various neural networks are used to process this type of data, and the one used in this work is the Point Cloud Transformer (PCT). This architecture leverages the powerful Attention technique deployed in Transformer models and, to more effectively capture local information, makes use of Farthest Point Sampling and Nearest Neighbor Search.

This work extends a previous preliminary investigation done on data from the "Ente Parco Archeologico del Colosseo"[1], whose primary objective is the reconstruction of ancient artifacts from their fragments. Within the site, artifacts dating back over 2000 years have been recovered, which may belong to one or even multiple statues depicting Apollo. These fragments have undergone meticulous scanning, cataloging, and archiving in anticipation of the reconstruction process. Specifically, there are over 500 artifacts of various sizes, ranging from a few centimeters to a diameter of about ten centimeters.

The key contributions of this work can be divided into both theoretical and practical aspects: from a theoretical perspective, there is a noticeable scarcity of models and papers focusing on the reconstruction of artifacts. This work represents a novel approach in this field by employing the Point Cloud Transformer architecture. This contribution adds a unique theoretical dimension to the existing body of knowledge, introducing innovative methods for artifact reconstruction. On the practical side, the significance of this work is evident. The model offers valuable support to archaeologists in the intricate task of reconstructing fragmented artifacts. By doing so, it opens new horizons for investigation and enhances our understanding of the past. The practical contribution underscores the real-world applications and implications of the proposed model in the field of archaeology.

The work is organized as follows: in the second chapter, a review of related works is presented, focusing on the application of machine learning techniques in the archaeological field, and an overview of scientific literature on models dealing with

[1] https://colosseo.it/

3D data is provided. The third chapter will be dedicated to the description of the data and the methods used. Following that, the fourth chapter will provide a detailed account of the experimental setup, describing the model and strategies employed. In the fifth chapter, the results of the conducted experiments are presented, with the conclusions outlined in the final chapter.

Chapter 2

Related Works

There is a growing trend in scientific papers published annually that simultaneously include tags related to both artificial intelligence and archaeology. [5] An example is a study that trained a Convolutional Neural Network (CNN) to identify signs on bone surfaces with remarkable precision. Surface bone modifications are crucial for the accurate identification of human traces, such as chewing, cutting, and trampling marks.

[6] Proposed a CNN capable of classifying images of ceramic fragments into different artistic styles. The study focuses on a specific type of ancient painted pottery from the American Southwest, Tusayan White Ware. The results demonstrate that a deep learning model can classify style types with accuracy comparable to that of four expert archaeologists.

Several studies explore the application of deep learning in combination with satellite data to identify potential archaeological sites, such as burial sites [7], mounds [8], Qanat[1] [9], or more generally, structures [10]. These papers use neural networks, such as U-Net, which perform well in these tasks.

An innovative approach is presented by [11], proposing a new methodology for the reconstruction of 2D puzzle fragments. Their method consists of two phases and does not use neural networks. The first step involves finding pieces with a similar edge through partial curve matching. Subsequently, they rank to find the piece that fits optimally. [12] Has a similar purpose to the previous one but uses 3D data. Their approach involves dividing each fragment into facets and, through the likelihood of 3D matching, attempting to reconstruct the object.

[13] Proposes a seed-region-growing CNN (SRG-CNN) for unsupervised semantic segmentation of point clouds based on 3D scans of Terracotta warriors. The SRG

[1] An ancient system of deep underground tunnels and wells built in the Middle East to channel water from a mountain to a dry lower region

algorithm is used to exploit point clouds and obtain normal features, which are then used in a CNN-type network. Also, on the Terracotta warriors, [14] presented work based on deep learning and template guidance to classify 3D fragments of statues. Specifically, they apply Pointnet to classify, and then misclassified fragments are categorized based on the best match to complete the terracotta model scan.

In the context of 3D data analysis, it becomes imperative to have specific models for their management. The application of Pointclouds to common convolutional networks is not straightforward. This type of network requires a regular structure, as it needs to arrange data in a grid for convolution operations, but point clouds do not have such a structure. One of the early approaches was to use data in the form of "Voxel," or Volume Pixels [15]. In this approach, point clouds are divided into small cubes, called voxels, creating the grid for convolution operations. However, this idea significantly increases the required memory, as many created cubes represent empty spaces. Additionally, as the point cloud resolution increases, the required voxels increase disproportionately.

Subsequently, the literature landscape dedicated to neural network architectures designed to handle this specific category of data is explored. [16] Introduced an innovative type of neural network called PointNet, capable of processing point clouds directly. The network proposes a unified architecture suitable for a wide range of applications, from object classification to part segmentation and semantic scene analysis. Despite its simplicity, PointNet proves highly efficient and effective. The downside of PointNet is its inability, due to its structure, to capture local information in data, providing only a global overview of the analyzed point cloud. To overcome this limitation, PointNet++ was later developed [17]. This model leverages the local structures of points by recursively applying a PointNet to subgroups of the initial data.

Another architecture that operates directly on point clouds is the Dynamic Graph Convolutional Neural Network (DGCNN) [18]. Instead of working on individual points, this model uses the geometric structure by building a local neighborhood graph and applying a convolution operation on the edge connecting the pair of points in the neighborhood to update node values, called edge convolution. The graph is not fixed; it is dynamically updated after each layer of the network, ensuring superior performance.

In the same years, a revolution was taking place in the field of Natural Language Processing (NLP) thanks to Transformers [19]. This innovative architecture,

equipped with both an encoder and a decoder, relies solely on the attention mechanism, allowing the capture of complex relationships and significant distances within sequences of data. [20] Adapted this type of model to operate in 3D environments, introducing the Point Cloud Transformer (PCT). This neural network aims to encode input points into a new higher-dimensional feature space, allowing the representation of semantic affinities between points as the basis for various point cloud processing tasks. The authors present two variants of this model. The first, called Simple Point Cloud Transformer, maintains a similar philosophy to the original Transformer, differing only in the absence of positional encoding. The second, called Point Cloud Transformers, introduces substantial improvements, including the adoption of Offset-attention instead of traditional self-attention and the introduction of Neighbor Embedding to capture local features more accurately.

[21] Implemented Point2Vec, a neural network based on the transformer with the addition of pretraining self-supervised learning. These architectures, similar to those used in other fields (Data2Vec, Word2Vec), are considered state-of-the-art for many tasks, such as Point2Vec for point cloud classification. These models are based on a student–teacher approach and are as effective as they are computationally demanding and expensive to train.

[22] Introduced the idea of fragmented solid object classification using neural networks based on point clouds. This work represents a preliminary investigation done on data from the "Ente Parco Archeologico del Colosseo". The first significant contribution lies in constructing various synthetic datasets containing fragments of 3D objects, usable for those intending to tackle the task. These data are publicly available under the name *Broken3D*[2]. The different datasets distinguish themselves in terms of creation methods, including variations such as the type of cut and simulated modifications to reflect the effects of aging on artifacts. The other important contribution of the work lies in performing the task of internal/external fragment classification. Indeed, the generated fragments can be divided into two categories: those completely contained within the original object (internal) and those that partially overlap with the original external surface (external). To address this task, Pointnet and DGCNN architectures were employed, with modifications made to optimize performance within the specific project. The obtained results were positive, both when applying the networks to the aforementioned synthetic data and fragments generated through the scanning of the bust statue of Nefertiti at the Neues Museum.

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

Chapter 3

Methods

3.1 Dataset

The data used in this work is of the Point Cloud (PC) type. This type of object is obtained through the use of specialized scanners that employ rapid laser pulses to collect hundreds of thousands of extremely accurate measurements per second. In recent years, LIDAR, an acronym for "Light Detection and Ranging" or "Laser Imaging Detection and Ranging," has gained significant popularity as a data acquisition technique in various sectors, including topography. A point cloud is a 3D digital representation of a physical object or space, composed of a set of individual points, each with its coordinates (x, y, z).

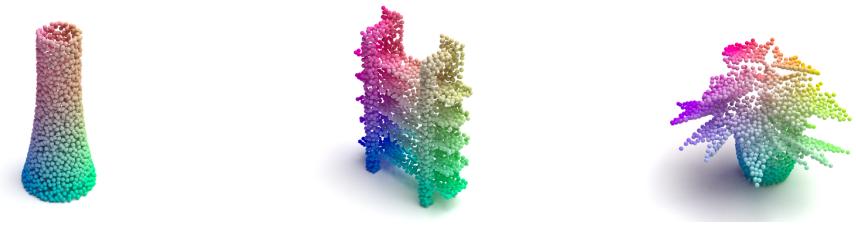


Figure 3.1. *Some examples of point clouds from ModelNet40 (Reproduced from Kaggle ModelNet40 - Princeton 3D Object Dataset).*

The data used in this work is the *Rotated cuts dataset* (RCD) that comes from the *Broken3D* archive. In the collection, the dataset was obtained from 3D solid objects which were cut to create fragments. The datasets vary from one another in terms of the type, orientation, and position of the cutting surface, as well as whether aging effects have been applied or not.

In addition to the classic structure of point clouds, these datasets have four additional features per point: nx, ny, nz, and A. The first three represent the normals regarding x, y, and z, respectively, while the last one is the area of the triangle

associated with the individual point. Through the use of normals, the orientation of the surface can be obtained, while the triangle's area allows us to understand if the surface is flat or undulated, as flat surfaces are represented with a few large triangles, while undulated ones have a high number of small triangles.

The RCD (Randomized Cutting Dataset) was constructed from 2180 solid objects of the types cube, sphere, and torus. The cutting surfaces have random orientations, allowing for fragments with realistic shapes. These surfaces have been designed to be irregular, adding further realism. Furthermore, a random noise was applied to the vertices of the triangle meshes. This noise was introduced to mimic the effects of aging and accidental breakages that can occur in artifacts. In Figure 3.2, a fragmented object from the RCD dataset is depicted, while Figure 3.3 displays a fragment along with its corresponding sampled point cloud.

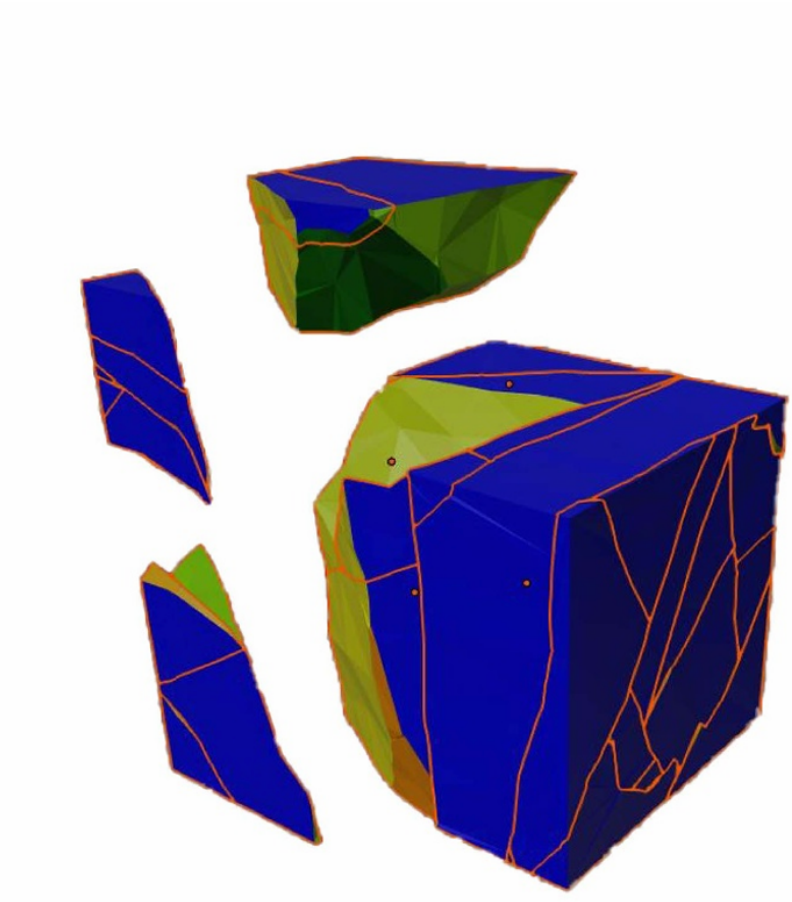


Figure 3.2. *Example of a fragmented cube from the RCD dataset in the Broken3D repository (Reproduced from A. Baiocchi et al., Artificial neural networks exploiting point cloud data for fragmented solid objects classification, 2023).*

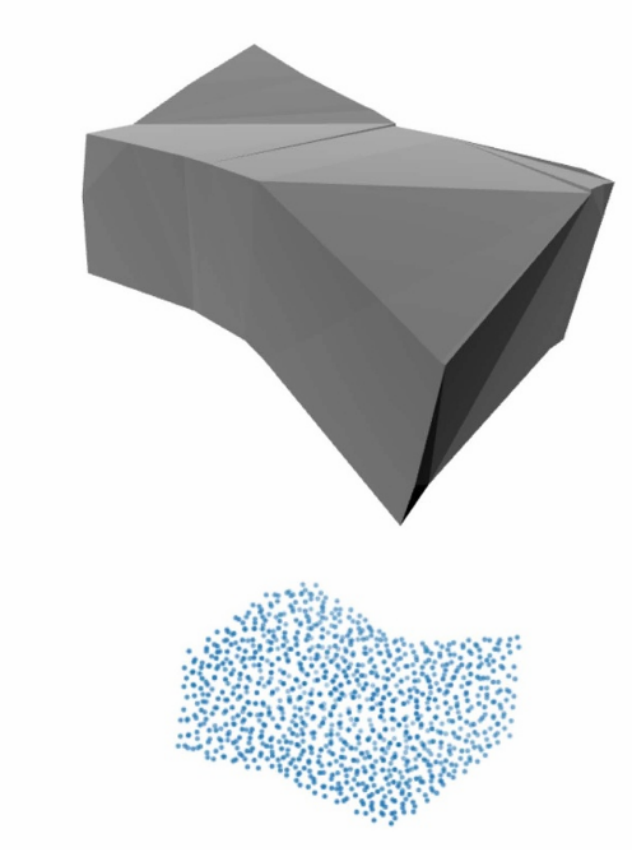


Figure 3.3. *Fragment and a point cloud sampled on its surface (Reproduced from A. Baiocchi et al., Artificial neural networks exploiting point cloud data for fragmented solid objects classification, 2023).*

Going into more detail, the dataset is divided into clusters, each having two macro elements $[n\_clusters, 2]$. The first, with the shape $[n\_frags, 1024, 7]$, represents the set of fragments belonging to the i-th cluster, each composed of 1024 points and 7 features per point. These, when combined appropriately, recreate the original 3D object. These fragments represent the input data for the model, i.e., $\mathbf{X}$.

The second macro element is the adjacency matrix associated with the cluster, providing crucial indications on how the pieces should be joined to form the original object. This matrix has the shape $[n\_frags, n\_frags]$ and takes binary values associated with each pair of fragments; it assumes the value 0 if the two pieces are not adjacent and 1 if they are. The values within the matrix represent the model's labels, i.e., $\mathbf{Y}$.

The dataset is divided into three parts: Train set, Validation Set, and Test Set, with proportions of 70%, 15%, and 15% of the total, respectively.

It is worth noting that the cluster sizes are not constant within the data, as can be seen in Figure 3.4. The average size seems to be between 20 and 40. Some

clusters are small, while there are a few very large clusters.

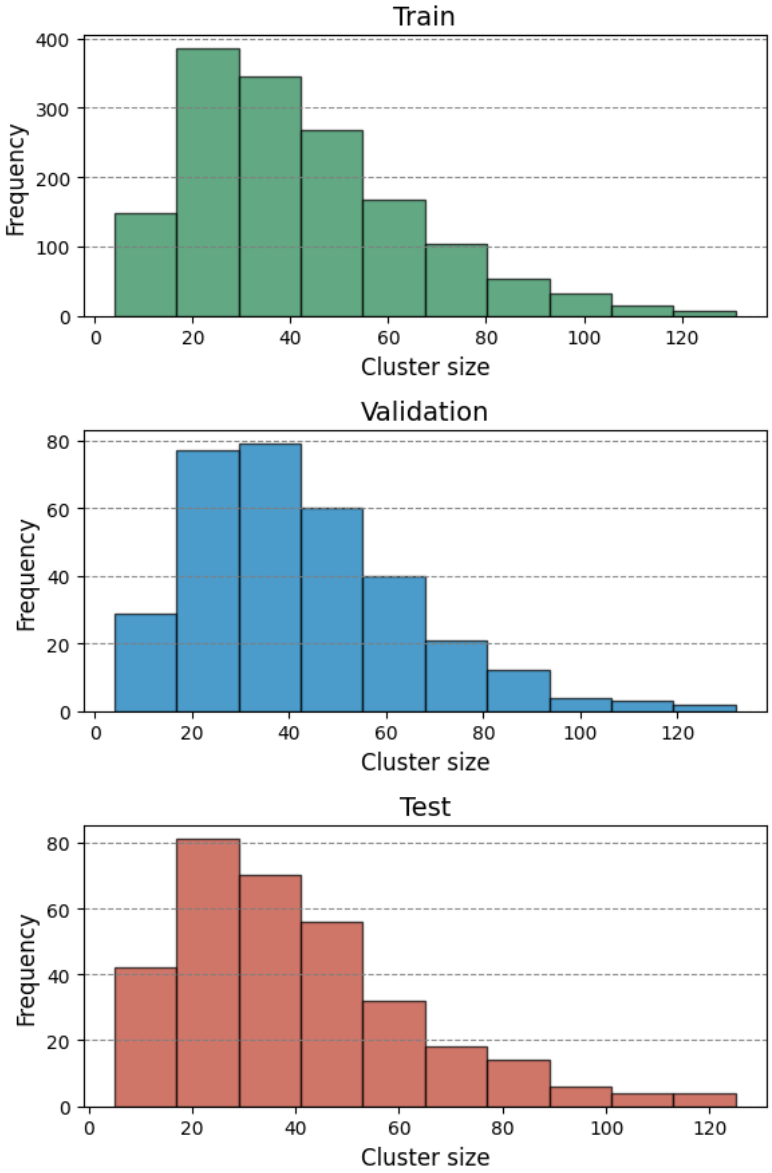


Figure 3.4. *Cluster size distribution.*

This means that some objects are divided into many pieces, while others into few, and the variability introduced should allow the network to reconstruct objects regardless of their division into fragments.

Label	Frequency
0	0.857
1	0.143

Table 3.1. *Distribution of the labels in the data.*

Furthermore, the Y values are heavily imbalanced; in Table 3.1, the distributions of 0s and 1s in each of the three subsets are reported. There are many more non-adjacent pairs (0) than adjacent pairs (1).

3.2 Models

In this section, the general concept of a neural network and its constituent components is initially introduced. Following that, the architecture of Transformers is presented. Subsequently, detailed descriptions of the main models employed for the point cloud classification task are provided.

3.2.1 Neural Networks

Neural networks, inspired by the functioning of the human brain, are computational models composed of interconnected nodes called neurons or hidden units, organized into distinct layers. These layers collaborate in processing and transmitting signals through existing connections, forming a versatile structure that finds applications in various domains.

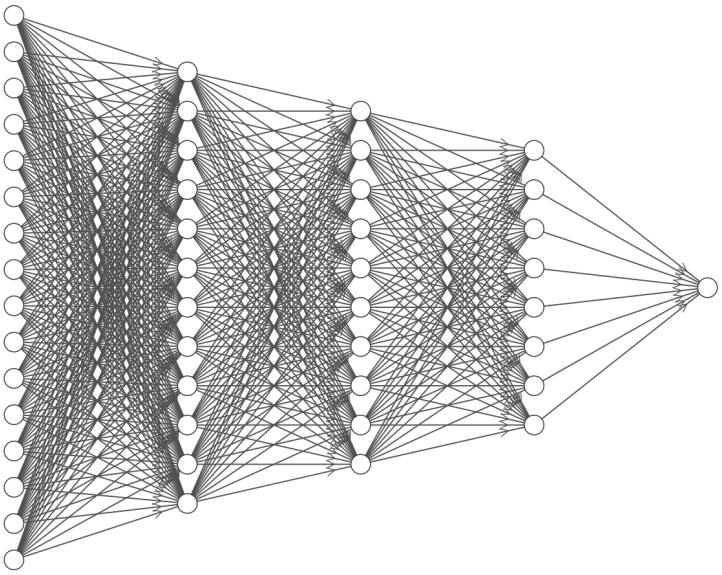


Figure 3.5. *A generical neural network.*

There are numerous variations of these architectures and new ones are invented every year. The simplest type of neural network is the fully connected neural network, which is nothing more than a stack of linear functions and non-linear functions (activation functions).

A linear function can be generally indicated as:

$$f(x) = Wx + b \tag{1}$$

where W represents the weight matrix, x is the input data to the function, and b is a generic bias. What sets neural networks apart from linear models is the activation function, which is applied directly to the linear output. Since the network consists of various layers, the output of each layer becomes the input to the next. A neural network with two layers, where h has dimensions equal to p, can be written as:

$$\begin{aligned} h &= \sigma(W_1 x) \\ f(h) &= W_2^T h \end{aligned} \tag{2}$$

Activation functions play a crucial role in these models; without their presence, everything would collapse into a single linear function. The world of neural networks offers a wide range of activation functions to choose from, and this is a continuously evolving research field. One of the most common and often effective choices is the Rectified Linear Unit (ReLU), which replaces negative values with zero and keeps positive values unchanged. Its simplicity and ability to successfully address many problems make it an excellent default choice.

3.2.2 Train a Neural Network

For a neural network to be used for a task, it must first be trained, which, delving into more detail, means obtaining values for the parameters in Equation 1 that minimize the loss function between the model predictions and the true values. Indeed, these parameters are initialized using techniques (e.g., Xavier Initialization), and then, over iterations, they are optimized through the computation of the gradient concerning the loss function. Computationally, these calculations are performed by the backpropagation algorithm, which represents the most efficient method for these operations. As previously mentioned, a neural network consists of multiple layers, and when calculating the gradient of the network, it is necessary to compute all the sub-gradients in the various layers.

Equation 3 illustrates an example of gradient calculation for a neural network with 3 layers.

$$\Delta_{W_1} Y_{(a)} = \delta_{W_1}^T h1_{(a\times b)} \cdot \delta_{h_1}^T h2_{(b\times c)} \cdot \delta_{h_2}^T h3_{(c\times d)} \cdot 1_{(d)} \tag{3}$$

The backpropagation algorithm leverages the chain rule, starting from the right side of the equation, it performs calculations avoiding matrix multiplications, resulting in faster computation.

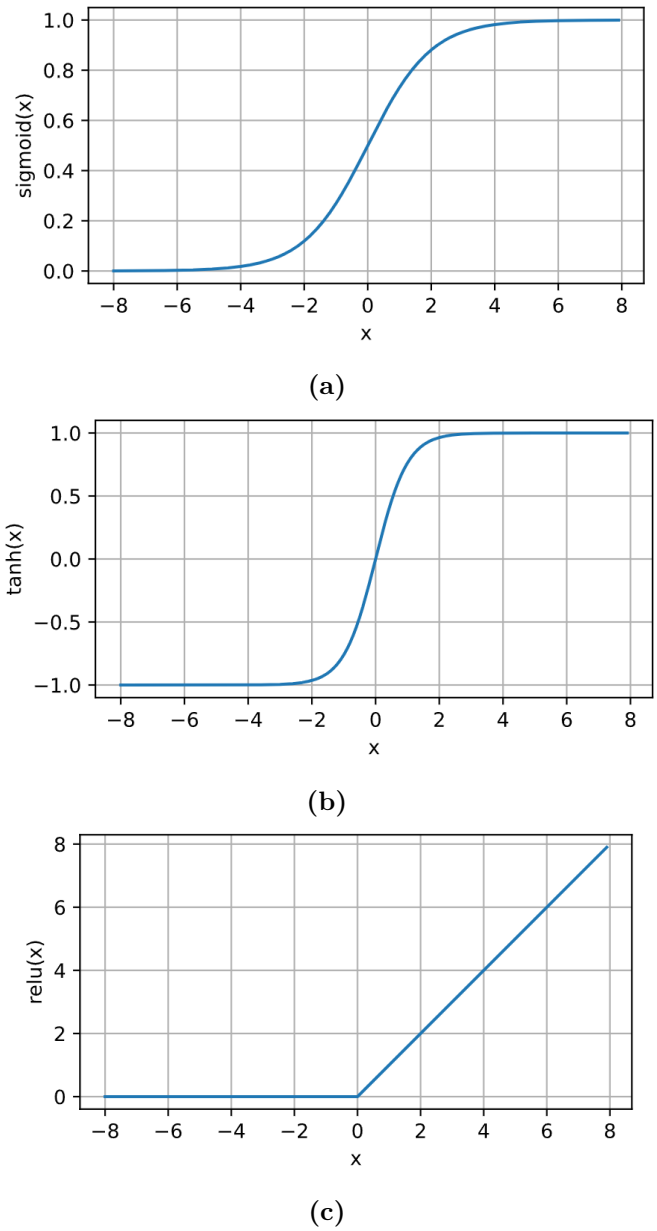


Figure 3.6. *Example of three of the most popular activation functions: (a) Sigmoid function; (b) Tanh function; (c) Relu function.*

Another crucial aspect during training is the adjustment of the learning rate α (Equation 4), which controls the size of the steps during parameter optimization. A too large learning rate can cause oscillations or instability, while a too small learning rate can make training very slow. Therefore, finding an appropriate value for this hyperparameter is essential.

$$\theta_{n+1} = \theta_n - \alpha \cdot \Delta J(\theta_n) \tag{4}$$

In deep learning models, two phases are encountered: training and testing (or inference). The training of a neural network is structured in epochs, representing iterations through the entire training dataset. During each epoch, the model receives input data, commonly divided into batches, from the training dataset. For each input, the network generates a prediction, which is then compared with the actual value. The resulting predictions are used to calculate the loss function. The ultimate goal of training a neural network is to minimize this function, meaning to make the model's predictions as close as possible to the actual data. In the inference phase, on the other hand, weights are no longer updated iteration by iteration. Instead, the network's performance is evaluated on a new dataset (the test set)

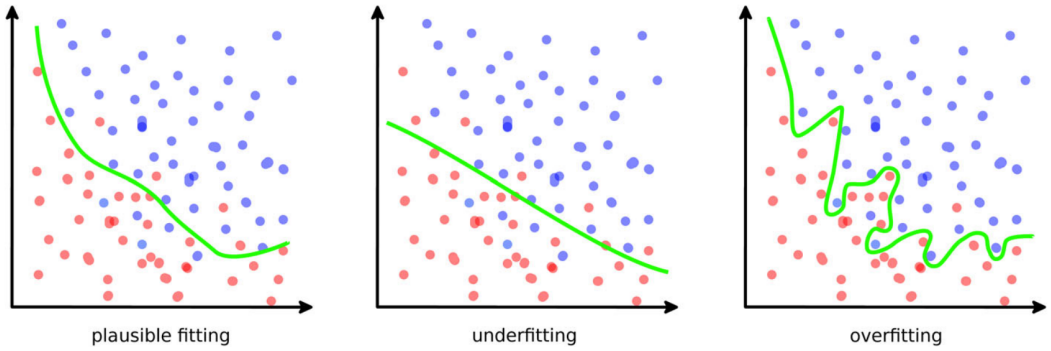


Figure 3.7. *Representation of the 3 possible generic cases in classification task (Reproduced from Fabio Galbusera et al., Artificial intelligence and machine learning in spine research, 2019).*

As observed in Figure 3.7, two phenomena can occur in the training cycle of a model: overfitting and underfitting. The former occurs when the network memorizes the training data but fails to generalize properly to unseen data, losing its ability to adapt to different situations. On the other hand, underfitting occurs when the model is too simple or undersized and fails to capture patterns in the training data. It is important to adjust the model parameters to avoid both phenomena. The goal is to have a well-adapted neural network capable of effectively generalizing to unseen data and making accurate predictions in real-world situations.

3.2.3 Improving a neural network

Several techniques and strategies can be applied to enhance the performance of a neural network. In this section, the choice has been made to highlight some elements that are utilized in the main model of this thesis.

Data Augumentation

The goal of data augmentation is to create new variations of the original data, but realistic ones, to enhance the network's performance, especially in cases where the amount of available training data is limited. Data augmentation is beneficial as it helps reduce overfitting by providing the model with a more diverse set of training examples. Additionally, it facilitates better generalization to new, unseen data. In this work, the chosen data augmentation technique involves applying random rotation to the fragments.

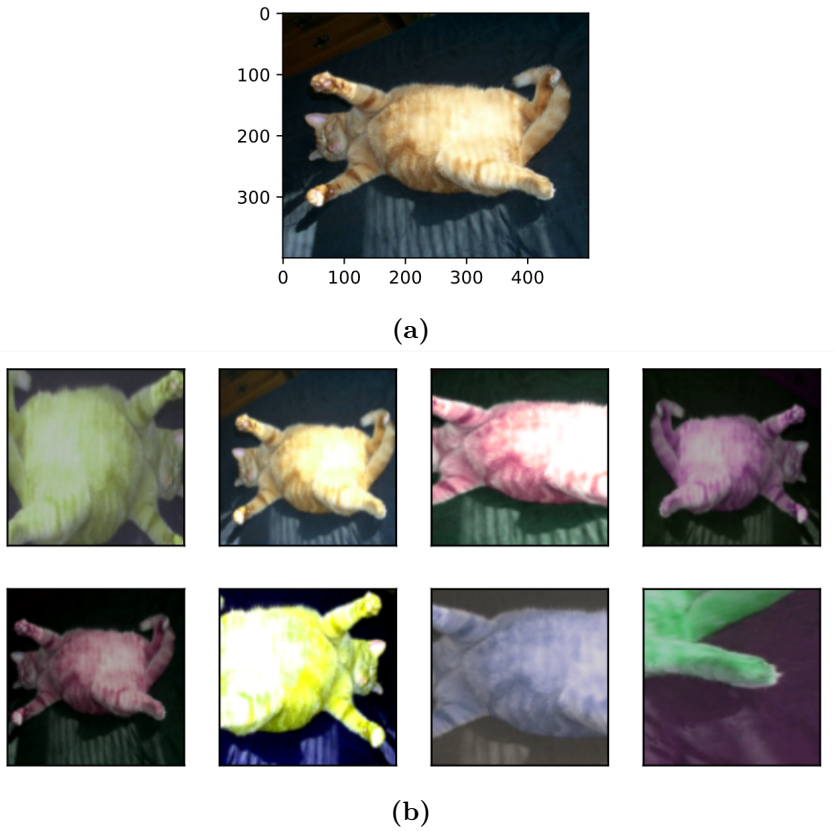


Figure 3.8. *(a) Original image; (b) Example of applying some data augmentation techniques to the original image. In particular, combinations of random flip, color augmentation, and shape augmentation were used in this case (Reproduced from Alexander J. Smola & Zachary Lipton, Dive into Deep Learning, 2023).*

Weight Decay

Weight Decay, also known as L2 regularization, is a technique employed in training neural networks to prevent overfitting. It is based on the principle of penalizing the model's weights, encouraging the network to use smaller weights, thereby reducing the model's complexity and improving its performance. Larger weights can lead to

overly complex networks that fit the training data too well but generalize poorly to unseen data. Mathematically, having a matrix of high weights means that a small change in input corresponds to a large change in output.

Dropout

The main idea behind dropout is to randomly "disable" a subset of neurons during each training iteration, preventing the network from becoming overly dependent on specific neurons and promoting generalization.

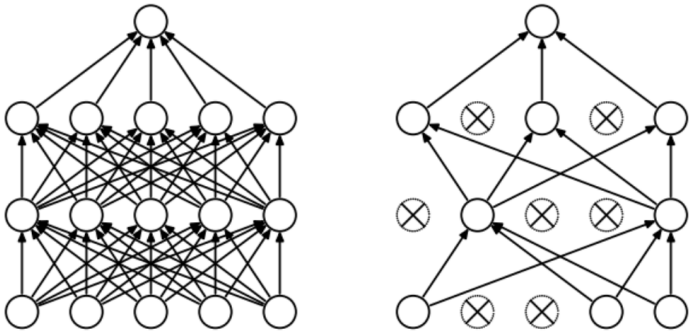


Figure 3.9. *A neural network before and after applying Dropout (Reproduced from Nitish Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014).*

If H is defined as the generic output of a layer in a fully connected neural network with f units divided into b mini-batches, the dropout operation can be represented by Equation 5:

$$\tilde{H} = H \odot M \tag{5}$$

Here, M represents a binary matrix, where the values are drawn from a Bernoulli distribution with probability p (i.e., $M_{i,j} = 0$ with probability p). Therefore, each neuron has a probability p of being deactivated.

This technique allows the network structure to vary for each iteration during training, dictated by the randomness of the matrix M. This should enable the model to become more robust.

During inference, in pursuit of a deterministic result, dropout behaves differently. The output of the layer H is replaced with its expected value, as shown in Equation 6:

$$E[H] = 0 \cdot p + H \cdot (1 - p) = H \cdot (1 - p) \tag{6}$$

Batch Normalization

The main objective of Batch Normalization is to make neural models more robust and accelerate the convergence of training. This technique is performed within a batch and consists of two main steps:

1. For each batch and each dimension, the mean ($\tilde{\mu}$) and empirical variance ($\tilde{\sigma^2}$) are calculated from the output of the linear layer (H). These values are used to standardize H:

$$H' = \frac{H - \tilde{\mu}}{\sqrt{\tilde{\sigma^2} + \epsilon}}, \quad \epsilon > 0 \tag{7}$$

2. Subsequently, using two parameters α_j and β_j, which are trained like other model parameters, a new mean and variance are added for each column:

$$[BN(H)]_{i,j} = \alpha_j \cdot H'_{i,j} + \beta_j \tag{8}$$

During inference, similarly to dropout, Batch Normalization behaves differently. To obtain deterministic results, mean and variance for each column are calculated over the entire dataset and kept fixed.

3.3 Transformers

Transformer is an advanced type of neural network, initially developed in 2017 for natural language processing (NLP) tasks. It features both an encoder and a decoder and is based on the powerful Attention mechanism (Figure 3.10). This capability allows the model to assign different weights to various parts of the input sequence during the encoding phase, automatically focusing on relevant portions of the text. This mechanism addresses the long-term dependency problem found in previous architectures like RNNs, enabling the network to capture more complex correlations within sequential data.

The encoder is tasked with processing and representing the input, extracting relevant information from the input sequence. On the other hand, the decoder is responsible for generating the output based on the contextual representation provided by the encoder.

Another favorable aspect of this architecture is the ability to perform computations in parallel, increasing overall efficiency. Additionally, differentiability is another advantageous property of Transformers: during the training process, there is no need to manually supervise the weights for each data point, as the network autonomously handles this task.

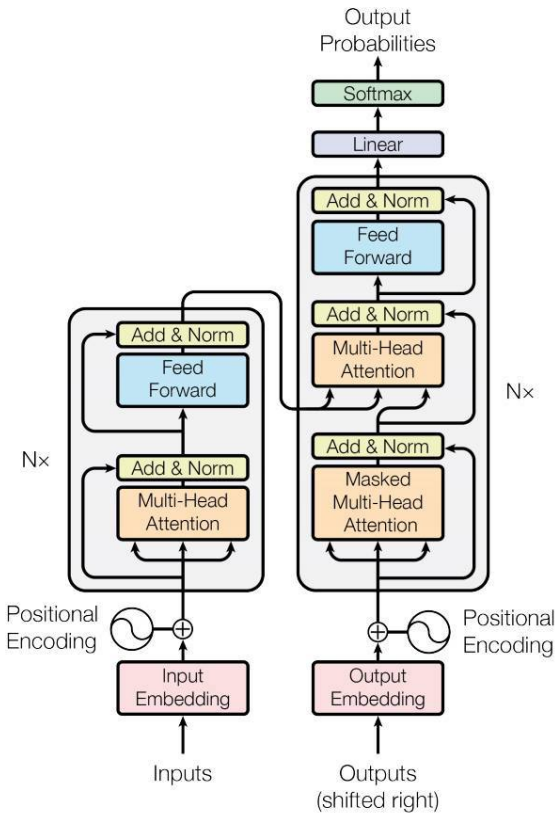


Figure 3.10. *Original Transformer Architecture (Reproduced from Ashish Vaswan et al., Attention is all you need, 2017).*

The central component of this neural network, as previously mentioned, is the attention function, specifically the one used by the researchers in the cited paper, which is the scaled dot product attention. The first step to perform its computation involves calculating, using three weight matrices and the input data to the model, as shown in Equation 9, the *Query* (Q), *Key* (K), and *Value* (V).

$$\begin{aligned} Q &= W_q \cdot X \\ K &= W_k \cdot X \\ V &= W_V \cdot X \end{aligned} \tag{9}$$

These three matrices play distinct roles, especially in the context of NLP:

- Query: represents the word for which the context is being determined. It will be used for comparison with other words in the sentence.
- Key: represents the other words in the sentence.
- Value: values are generated for each word in the sentence, preserving contextual

information.

The attention implemented in the paper is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{10}$$

The multiplication operation between *Query* and *Key* generates a similarity score, called attention weights, between the word represented in the query and all other words in the sentence. To make these weights more uniform, they are normalized by dividing by the square root of the size of the query input (d_k). Subsequently, the softmax function is applied to map the values to the range [0,1]. Through the final multiplication, the corresponding weight is assigned to each element of *Value* concerning the word present in the initial *Query*.

All of these concepts can be likened, for example, to web searches. The search engine maps the user's query (the text entered in the search bar) by comparing it with a predefined set of keys (article titles, descriptions, web pages) in the database. Finally, it outputs the pages that best fit the request (values). To be precise, self-attention is referred to when the three matrices (Q, K, V) are generated from the same input X, as in the case of the Transformer's encoder. Otherwise, if K and V are calculated from the same input X_a, while Q is from X_b, the term used is attention.

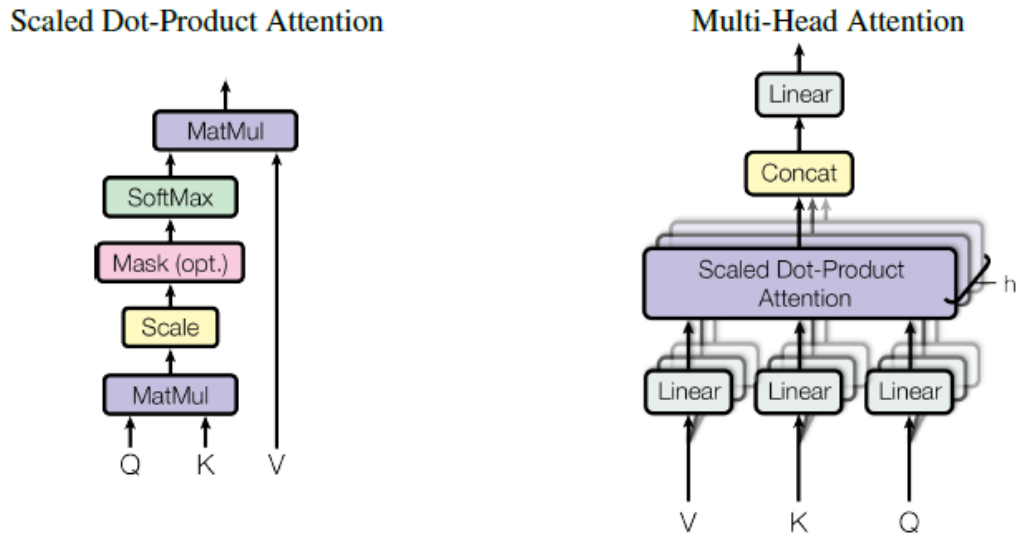


Figure 3.11. *Multi-head attention (Reproduced from Ashish Vaswan et al., Attention is all you need, 2017).*

In the original model, instead of using a single attention mechanism, Multi-head attention is employed (Figure 3.11). This approach involves incorporating several

layers of attention that operate in parallel. The primary objective behind this design is to empower the network to simultaneously focus on various aspects or relationships among words within the input sequence. Additionally, there exists a variation known as Masked Multi-head attention, where some tokens are intentionally concealed. This masking is often employed during the training phase to ensure that the model does not have access to future information when predicting a specific token.

Finally, another crucial component in this architecture is positional encoding, which is added to the inputs before being passed to the network. Since the attention mechanism is invariant to permutations, meaning that changing the order of words does not alter the model's result, the addition of these values overcomes this limitation by assigning a unique value p_i to each input value x_i, representing the position of the word within the sentence.

3.4 Point Cloud Transformers

Transformers originated in the field of Natural Language Processing (NLP) and have achieved considerable success, a trend that is being replicated for image-related tasks as well. Through the Point Cloud Transformer (PCT), it has become possible to adapt this type of model to 3D data.

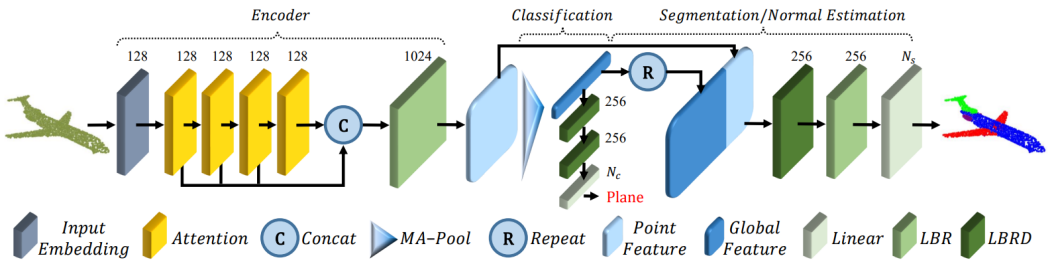


Figure 3.12. *Simple Point Cloud Transformer Architecture (Reproduced from Guo Meng-Hao et al., PCT: Point Cloud Transformer, 2021).*

The overall architecture of the PCT is presented in Figure 3.12. This network aims to encode input points into a new higher-dimensional feature space, allowing for the representation of semantic affinities between points as the foundation for various point cloud processing tasks. The PCT encoder largely shares the same philosophy as the original Transformer, with the exception of positional embedding, which is discarded since the coordinates of the points already contain this information.

The encoder begins by embedding the input coordinates into a new feature space.

Subsequently, these embedded features are fed into four attention modules, and their outputs are concatenated together. This enables the model to learn a rich and discriminative semantic representation for each point. Following this, there is a block, referred to by the authors as LBR, consisting of a linear layer, batch normalization, and ReLU activation function. To extract a global feature vector F_g representing the input point cloud, max-pooling and average-pooling are applied in concatenation. This vector F_g is utilized by the decoder for the classification and segmentation tasks.

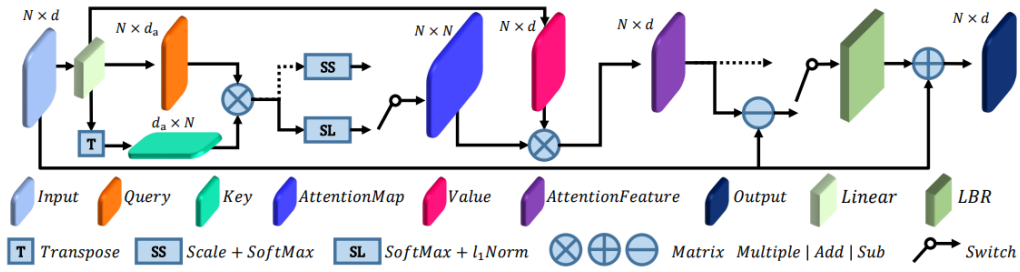


Figure 3.13. *Offset Attention Architecture (Reproduced from Guo Meng-Hao et al., PCT: Point Cloud Transformer, 2021).*

Subsequently, in the same paper, the previous architecture is slightly modified to enhance performance. The first upgrade involves changing the self-attention module to the offset-attention (Figure 3.13). The idea is inspired by Graph Convolution Networks, where using the Laplacian matrix (L) instead of the adjacency matrix (A) shows benefits. Recall that L is calculated as follows: $L = A - D$ where D is the diagonal degree matrix. In the case of the offset-attention (OA) layer in the model, it calculates the difference between the self-attention (SA) features and the input features through element-wise subtraction.

In fact, if the output of the self-attention of Transformers, defined in Equation 10, is denoted as F_{sa}, and the previously mentioned input data as F_{in}, the new attention weights are calculated as follows:

$$F_{oa} = F_{sa} - F_{in} \tag{11}$$

Even the normalization of the attention weights changes slightly. Specifically, the *Softmax()* function is used to normalize the first dimension, while the *l1*-norm is used for the second one, as shown in Equation 12:

$$\begin{aligned} \hat{a}_{i,j} &= Softmax(\tilde{a}_{i,j}) = \frac{exp(\tilde{a}_{i,j})}{\sum_k exp(\tilde{a}_{k,j})} \\ a_{i,j} &= \frac{\hat{a}_{i,j}}{\sum_k \hat{a}_{i,k}} \end{aligned} \tag{12}$$

The other modification present in the improved version of PCT is the use of local features obtained through Neighbor Embedding. As shown in Figure 3.14, the Neighbor Embedding module consists of two LBR blocks (Linear, BatchNorm, and ReLU) and two SG blocks (sampling and grouping). The LBR blocks function in the same way as in the previous version. The two SG blocks are used in cascade to gradually enlarge the receptive field during feature aggregation. The SG layer aggregates features from local neighbors for each point, grouping them through k-NN search and using Euclidean distance during point cloud sampling.

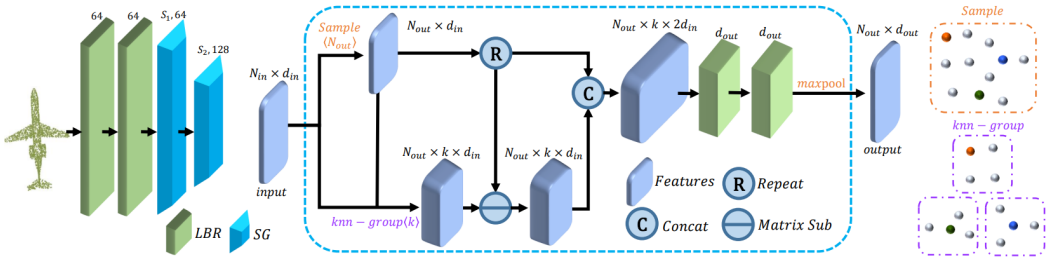


Figure 3.14. *Left: Neighbor Embedding architecture; Middle: SG Module; Top-right: example of sampling; Bottom-right: example of grouping with k-NN neighbors (Reproduced from Guo Meng-Hao et al., PCT: Point Cloud Transformer, 2021).*

Going into more detail, assuming that the SG layer receives an input point cloud P with N points and F features, the first step is to apply the Farthest Point Sampling (FPS) algorithm to downsample P into P_s. For each selected point $p \in P_s$, a group is formed containing its first k neighbors, denoted as $Knn(p, P)$. Subsequently, the output features F_s for the point p are calculated in the following way:

$$\begin{aligned} \Delta F(p) &= concat_{q \in Knn(p,P)}(F(q) - F(p)) \\ \tilde{F}(p) &= concat(\Delta F(p), RP(F(P), k) \\ F_s(p) &= MP(LBR(LBR(\tilde{F}(p)))) \end{aligned} \tag{13}$$

Therefore, the first step is to calculate $\Delta F(p)$, representing the differences between the features of point p and those of all other points q belonging to its neighborhood, and the results are concatenated. Next, the previous output is further concatenated with $RP()$, which simply repeats the F_p k times to form a matrix. Finally, this information is passed through two dense layers and a final Max-pooling to obtain F_s as output. This process thus shares a similar idea with the Edge Convolution in DGCNN.

To differentiate between the two versions, the authors refer to the first model, the one with self-attention and without Neighbor Embedding, as Simple Point Cloud

Transformer (SPCT). The second variant is named Point Cloud Transformer (PCT).

Chapter 4

3D fragments matching

This chapter begins by delving into the architecture of the pair model employed within this work, offering a comprehensive understanding of its design and functionalities. Next, the experimental setup used to run the model is explained, providing an overview of the methodologies and procedures adopted during the training phase.

4.1 Pair Model

As previously mentioned, the 3D fragments matching task is a novel challenge, and there are limited models capable of addressing it. The approach presented below signifies innovation. The PCT outlined in the preceding chapter served as the foundation, with subsequent modifications.

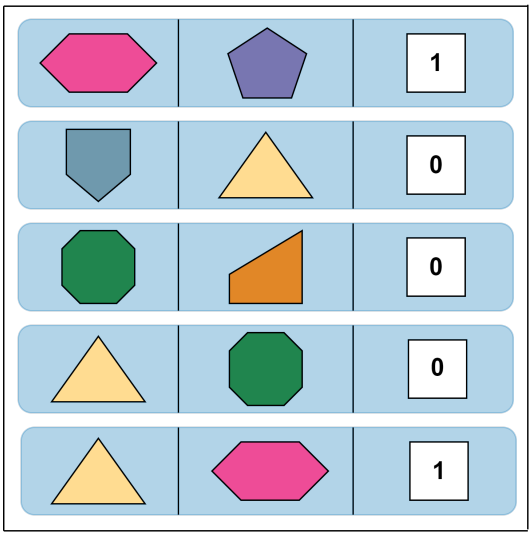


Figure 4.1. *Representation of generic input data of the model developed in this work.*

The data described in the dataset section, before being placed within this network undergoes a modification. Originally organized in clusters, to be processed by the neural network created in this work all possible pairs are unrolled and saved in a list in triplets [frag_a, frag_b, label]. In Figure 4.1, it is possible to see the conformation of the generic data that welcomes the model as input. Each geometric shape of a different color represents a different fragment, and each row therefore represents a possible pair of fragments with its label.

The neural network developed for this thesis, as shown in Figure 4.2, presents an architecture having two branches. In each of the two branches, there is the point cloud transformer encoder having shared weights. Compared to the original PCT encoder, modifications were made to allow compatibility with the data sizes used in this work. Each fragment has 7 features instead of the traditional three required by the first layer of the PCT encoder.

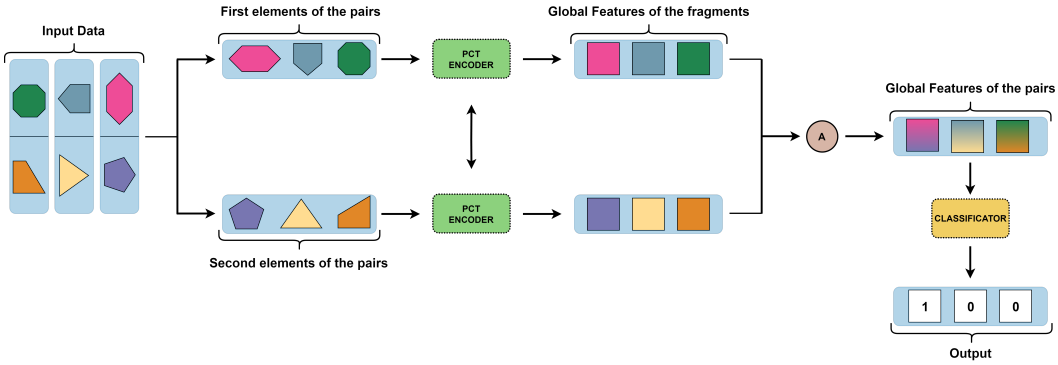


Figure 4.2. *Pair model.*

The input pairs are divided into two groups, one containing the first elements of each pair and the other the second. These tensors of fragments are processed in parallel in the two branches of the network through the pct encoder layers. The output of each branch represents the global features of the individual fragments input. The next step is to go and aggregate the two tensors produced to arrive at the global features of the pairs. Named G_1 and G_2, respectively, the global features of the first and second elements of the pair, the aggregation process is described in Equation 14:

$$\begin{aligned} G\_Sum &= G_1 + G_2 \\ G\_Mul &= G_1 * G_2 \\ G\_Tot &= cat(G\_Sum, G\_Mul) \end{aligned} \tag{14}$$

So, two symmetric functions such as sum and multiplication applied to G_1 and G_2 are used to produce the global features of the pairs (G_Tot). The symmetric

functions ensure that for a generic pair of fragments (a, b), the order of insertion of the two elements does not affect the creation of the global features of the pair.

Then, *G_Tot* is input to the PCT's original classifier, which consists of three linear layers, where both relu and batch normalization are applied on the first two, interspersed with two dropout layers. In the output, the model generates predictions regarding the adjacency of the two elements forming the pair.

4.2 Train Setup

Given the large number of pairs in the dataset (approximately 2 million) and the specific weight of each data point and the utilized neural network, the initial step involves randomly selecting 10,000 balanced pairs at each epoch of the model. In other words, 5000 pairs of adjacent fragments and an equal number of non-adjacent pairs are extracted through random sampling of the dataset, ensuring a balanced representation of various possible pairs.

Subsequently, the data is organized into batches, each composed of 64 pairs. At this point, the two elements constituting each pair are separated into two distinct branches. One branch contains the first elements, while the other contains the second elements. This division allows the insertion of data into the two branches of the network, where one branch processes all the first elements of the pair, while the other processes the second elements. To enhance the model's generalization capability, each individual fragment undergoes a random rotation, serving as a form of data augmentation. This practice, as explained in the previous chapter, enables the model to classify fragments independently of their orientation. Rotations are applied using Euler angles [α, β, γ], where the three values are randomly sampled for each fragment from the range [0, 360°]. This allows the fragments used during training to not have a privileged orientation, as is the case for real data obtained from fragment scanning. Furthermore, all fragments are translated to the origin. This adjustment helps ensure that the network cannot base its classification on the spatial position of the fragments but instead focuses on the intrinsic features of the data.

The model is trained using the ADAM optimizer, and the cross-entropy is employed as the loss function. This measures the dissimilarity between probability distributions and aids in training models by penalizing inaccurate predictions. In other words, it compels the network, over the epochs, to become increasingly precise in its probability-based choices. Cross-entropy is widely used in tasks such as binary

classification, where the goal is to classify data into two classes. The formula for this type of loss, calculated on examples from a dataset, is as follows:

$$Loss = \frac{1}{N}\sum_{i=1}^{n} y_i * log(p(y_i)) + (1 - y_i) * log(1 - p(y_i)) \tag{15}$$

Where:

- y_i: represents the label, which can be either 1 or 0.
- N: is the number of observations in the dataset.
- $p(y_i)$: represents the probability predicted by the network for the event y_i.

It should be noted that the use of the term 'epochs' takes on a peculiar meaning here. In traditional machine learning practice, the term is associated with completing a cycle during which the model has examined the entire training dataset. However, in the context of this work, at each epoch, the network only analyzes a subsample of the total data, representing a variation from classical terminology. In other words, the term 'epochs' assumes a more specific role, indicating the moment when the model examines a partial and balanced set of extracted pairs, thereby contributing to the efficient and weighted management of training.

Concerning the validation set and the Test set, a subsample of the original data is also used here. Specifically, three thousand positive pairs and an equal number of negative pairs are randomly included, which, unlike the training set, are kept constant throughout the training cycle.

Chapter 5

Results

In this section, the metrics used to evaluate the predictions outputted by the model are first described. Subsequently, the experiments conducted regarding the coupling of fragments using the data and model presented in the previous chapters are reported. All presented experiments have been made reproducible by fixing all of the random seeds and can be executed through the code available in the official GitHub repository [1].

5.1 Metrics for evaluation

In a binary classification model, such as the one in this study, the first step typically taken to evaluate the quality of classification is to print the so-called confusion matrix. In detail, the matrix highlights where the network makes errors, instances in which it responds less proficiently, and those in which it performs better.

The matrix is composed of four values, namely True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN), where:

- TP: represents the number of cases in which the model correctly predicted the positive class.
- TN: constitutes the number of cases in which the model accurately predicted a negative class.
- FP: represents the number of cases in which the model mistakenly predicted a positive class when it was negative.
- FN: indicates the number of cases in which the model mistakenly predicted a negative class when it was positive.

[1] https://github.com/Sottix99/Point-cloud-transformers-for-3D-fragment-matching/tree/main

The four values outputted by this matrix are extremely useful, as they provide an immediate idea of where the network is making the most mistakes, and they are also used to compute the most commonly used evaluation metrics.

In particular, to assess the performance of the model, in addition to the previously mentioned confusion matrix, three main metrics are considered: Accuracy, F1-Score, and AUC score.

Accuracy represents the fraction of correct predictions returned by the network over the total predictions made. It thus represents the percentage of times the model correctly guessed the class membership of the elements. The formula is as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{16}$$

Regarding the F1-Score, the formula is represented in Equation 17 and relies on the measures of Precision and Recall:

$$F1score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{17}$$

Since, in turn, Precision and Recall are calculated as follows (assuming the positive class is the target) through the use of measures derived from the confusion matrix:

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned} \tag{18}$$

It is possible to transform Equation 17 into:

$$F1score = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{19}$$

As stated earlier, Accuracy represents how often the network has predicted correctly, but it can be considered reliable only in a balanced dataset. In cases where there is a strong predominance of one of the two classes, the model could achieve good accuracy by simply predicting the majority class every time.

Precision represents how many of the model's "positive" predictions are actually correct. In the context of this work, it measures the fraction of fragments that the neural network correctly identified as adjacent to the total number of fragments predicted as adjacent. Recall measures how many elements belonging to the positive class were correctly identified by the model. Unlike the previous metric, the de-

nominator changes. Previously, the comparison involved correct positive predictions to the total positive predictions made by the network; now, the comparison is made between correct positive predictions and the total actual positive instances. These last two metrics represent a trade-off, meaning an increase in one leads to a decrease in the other. Increasing precision entails having a very strict classifier that even doubts truly positive elements, thus reducing recall. Conversely, having higher recall leads the network to accept more elements as positive, resulting in a reduction of precision. Ideally, the goal is to maximize both to achieve a more accurate classifier. The F1 score combines precision and recall using their harmonic mean. In contrast to the typical arithmetic mean, the harmonic mean assigns greater weight to smaller values. This ensures that a classifier achieves a high F1 score only when both precision and recall are high. Therefore, maximizing the F1 score implies simultaneously maximizing both precision and recall. Hence, this metric has become a widely used choice for model evaluation.

The last metric under consideration is called AUC (Area Under Curve). To calculate this value, it is necessary to initially plot the ROC curve (Receiver Operating Characteristic). This curve represents the relationship between the true positive rate and the false positive rate as the model's decision threshold varies. In other words, the ROC curve highlights the network's performance considering all possible classification thresholds. In the curve, a higher value on the X-axis indicates a greater number of false positives compared to true negatives, while a higher value on the Y-axis indicates a greater number of true positives compared to false negatives. The AUC metric precisely represents the area under the ROC curve. An AUC value of 1 suggests that the neural network can perfectly distinguish between positive and negative classes. Conversely, a value of 0.5 indicates that the model cannot make significant distinctions between classes, equivalent to random chance. In practice, AUC evaluates the model's ability to discriminate between classes, regardless of the choice of a specific decision threshold. A ROC curve that approaches the upper-left corner with a significant area underneath suggests a more performing model.

5.2 Runs

Several experiments were conducted to evaluate the model in the research task of this thesis. Initially, a baseline run was carried out to serve as a reference point for subsequent comparisons. In the subsequent experiments, parameters from the baseline run were adjusted to observe how the network responds to those changes.

5.2.1 Base Run

Hyperparameter	Value
Batch Size	64
Learning rate	0.00005
Number of Epochs	200
Optimizer	Adam
Weight decay	0.0001
Number of features	7
Number of couples for epoch	10.000
Type of couples	Balanced
Fixed Couples	no

Table 5.1. *Training details of the Base Run.*

The comprehensive and concise details of this attempt are reported in Table 5.1. To evaluate the performance obtained in this initial baseline run, let's start by analyzing the training loss, as shown in Figure 5.1.

Figure 5.1. *Train Loss Curve.*

It is evident that after around the 50th epoch, the values stabilize around 0.63%, fluctuating around this value in subsequent epochs. It can be observed that a similar phenomenon occurs more or less for the other two considered metrics, namely, Train Accuracy (Figure 5.2) and Train F1 (Figure 5.3). All metrics show the highest improvements in the first 20 epochs, although both accuracy and F1 score reach their peaks in the 163rd epoch, with values of 65.93% and 65.78%, respectively. The fact that accuracy and F1 score have very similar values suggests that the model is performing well on both fronts: it has a high rate of correct predictions

(accuracy) and effectively manages the trade-off between precision and recall (F1 score), indicating a balanced presence of False Positives and False Negatives.

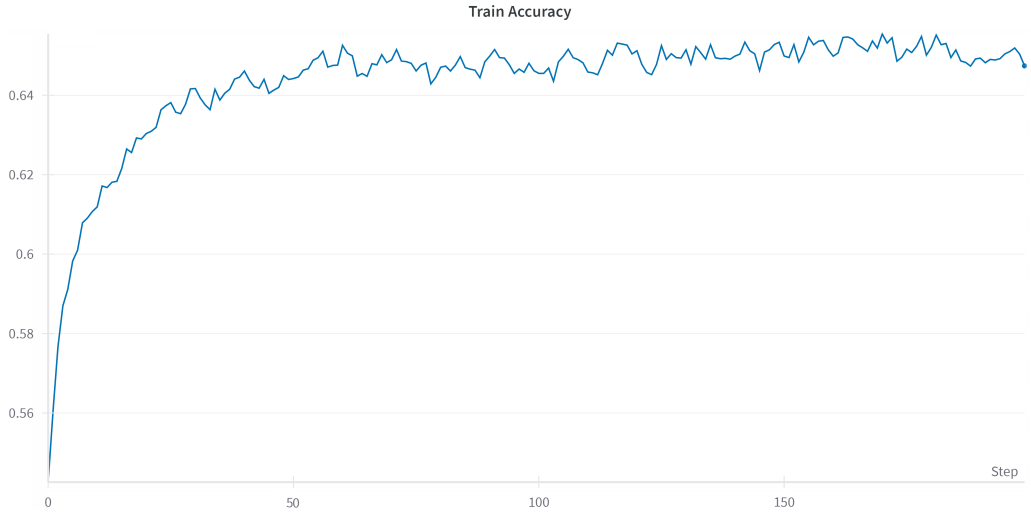


Figure 5.2. *Train Accuracy Curve*

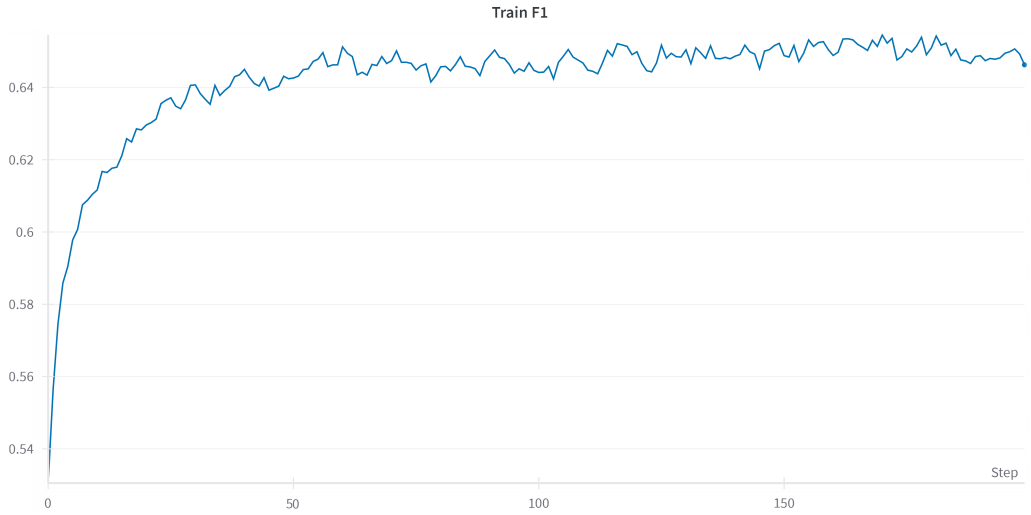


Figure 5.3. *Train F1 Curve*

Regarding the metrics calculated on the validation set, as illustrated in Figures 5.4, 5.5 and 5.6, more pronounced fluctuations between epochs can be observed, but overall, the trend is similar to that in Figures 5.1, 5.2 and 5.3.

Notable advancements are evident, particularly within the initial 20 epochs, showcasing the best results at epoch 116. During the iteration, the network attains a noteworthy accuracy of 65.47%, achieves a peak F1 score of 65.38%, and sees a substantial reduction in loss to 0.6235.

Figure 5.4. *Validation Loss Curve.*

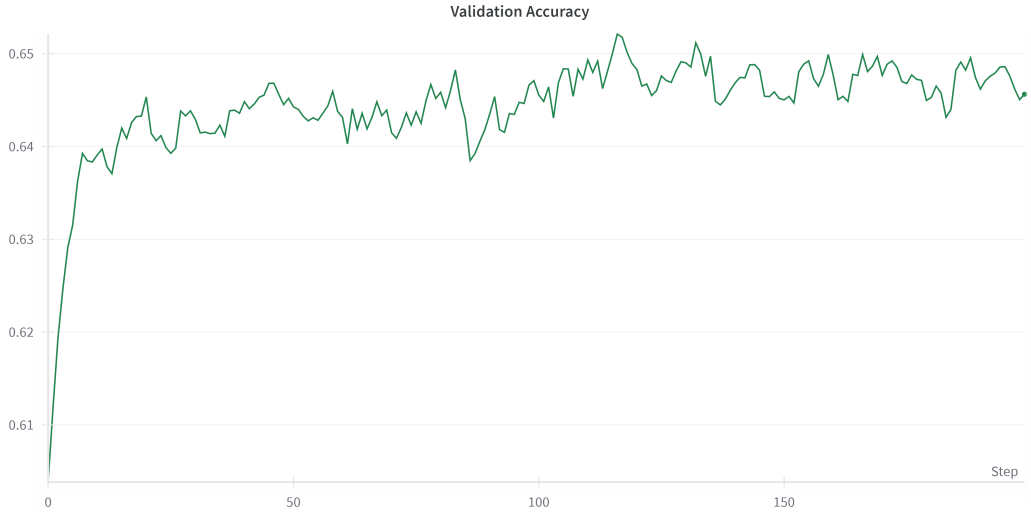


Figure 5.5. *Validation Accuracy Curve.*

Furthermore, utilizing the learned weights from epoch 116, the Area Under the Curve (AUC) score was computed, yielding a value of 0.705.

The confusion matrix, showcased in Figure 5.7, illustrates the model's tendency to classify a marginally higher number of pairs as negative (nonadjacent) rather than positive (adjacent).

However, it is notable that there exists a commendable balance between recall and precision, substantiated by the F1 metric value.

Figure 5.6. *Validation F1 Curve.*

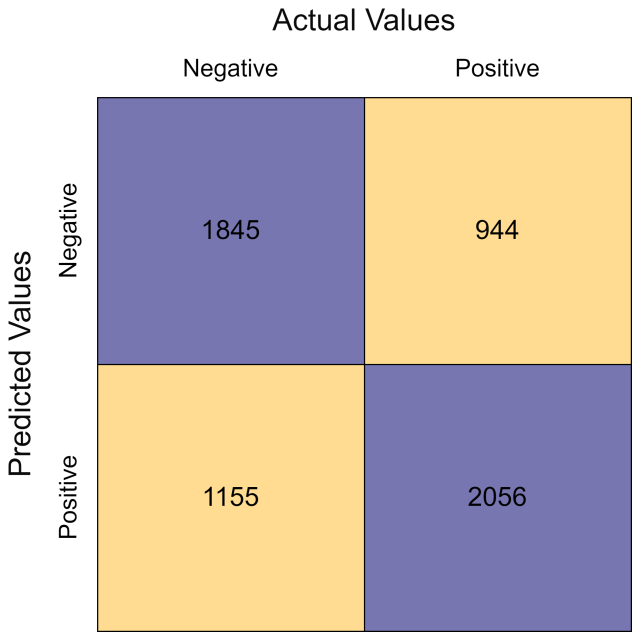


Figure 5.7. *Confusion matrix of the validation set computed for the epoch 116.*

The final step in evaluating this run involves performing inference on the test set. Using the weights from epoch 116, metrics such as loss, accuracy, F1, and AUC score were computed for this data set, and the results are presented in Table 5.2.

It can be observed that the metrics are very similar to those measured in the training and validation sets. In Figure 5.8, the associated confusion matrix has been presented, and it is observed that it is also similar to the one in the previous figure.

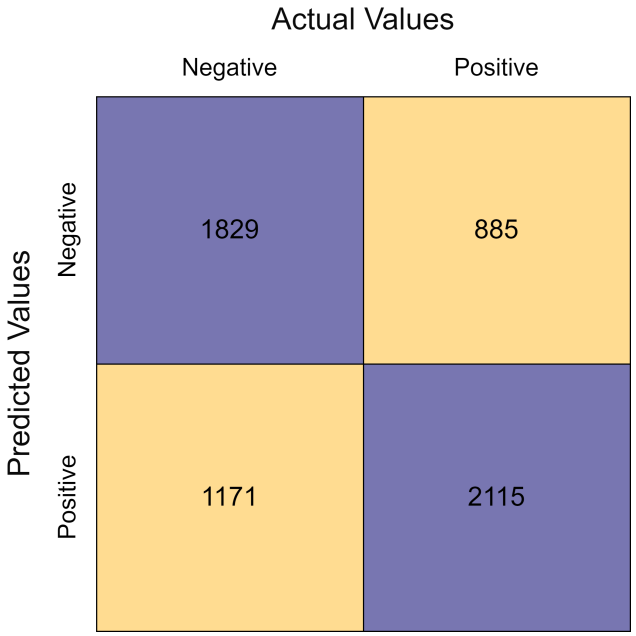


Figure 5.8. *Confusion matrix of the test set.*

Loss	Accuracy	F1	AUC
0.618	0.657	0.657	0.715

Table 5.2. *Metrics on the test set.*

5.2.2 Changing the numbers of features

As previously mentioned, the *Broken3D* dataset consists of seven distinctive features [*x, y, z, nx, ny, nz, A*]. Two training runs were conducted: one considering only the three spatial features [*x, y, z*], and the other exclusively removing information regarding the triangle area while retaining six features. This analysis aims to provide a more in-depth understanding of the contribution of features within the context of model performance. It is expected that the removal of feature A will not lead to significant changes, whereas the utilization of only spatial features may result in a noticeable decline in performance. This is because the information provided by the three normals, indicating the orientation of the solid surface, is considered valuable in the scientific literature for correctly assembling fragments. For these two runs, the hyperparameters were kept unchanged from the baseline run, except for the batch size, which was set equal to 16, and the number of epochs, which was decreased to 100. As it was seen that after the 50th epoch, the values became quite stable, and to save computational power and time, the choice was made to reduce the epochs.

In graphs 5.9, 5.10, and 5.11, the metrics of the two runs performed with a different number of features are presented.

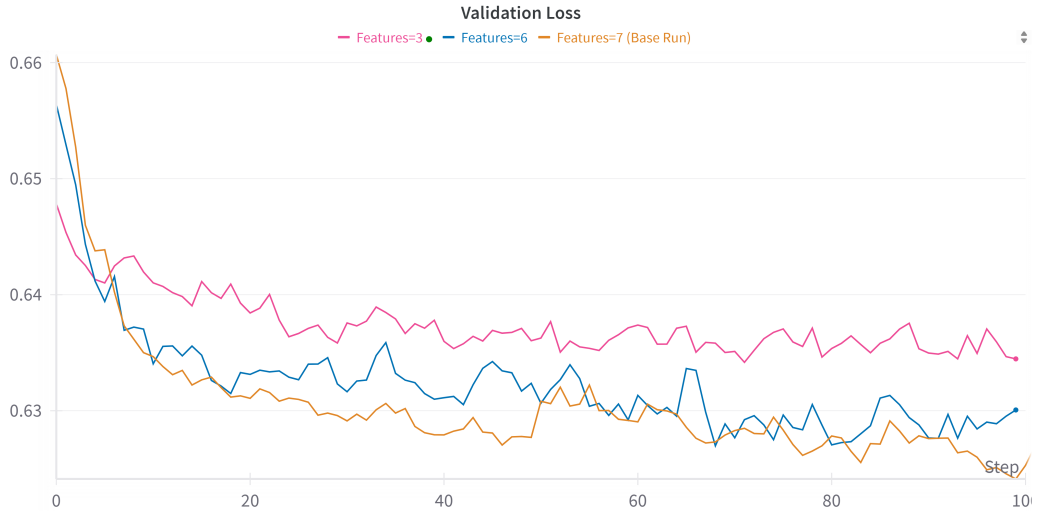


Figure 5.9. *Comparison of loss metric values on the validation set based on the number of features in the data.*

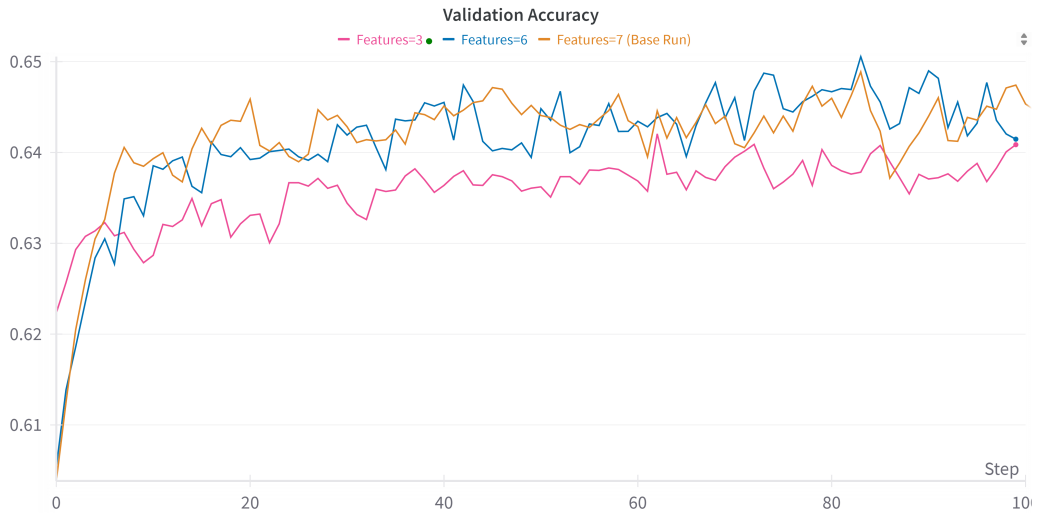


Figure 5.10. *Comparison of accuracy metric values on the validation set based on the number of features in the data.*

The results have been compared among themselves and with the first hundred epochs of the baseline run described in the previous section. It's evident that the initial suspicions were accurate: across all three plots, the line representing the scenario with three features consistently produces inferior results compared to the other two. Only at epochs 63 and 100 did the values of this run align with the other two, achieving an accuracy of 64.72% and an F1 score of 64.69%. The cases with 6 and 7 features, on the other hand, appear to be more or less equivalent.

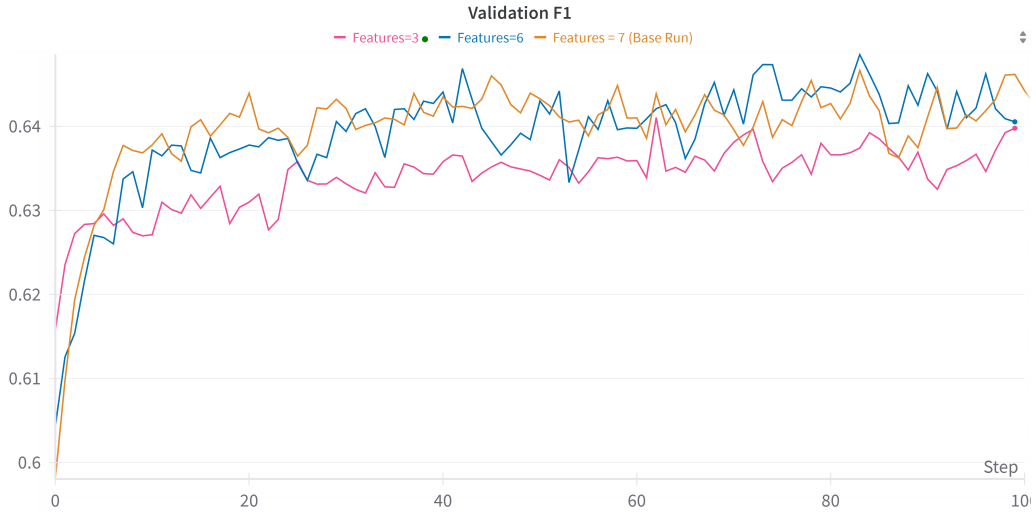


Figure 5.11. *Comparison of F1 metric values on the validation set based on the number of features in the data.*

Surprisingly, in the first hundred epochs of the validation set, the highest peaks in both accuracy and F1, even if only slightly, are achieved by the run with 6 features. In epoch 83, for instance, the validation accuracy of this run reaches 65.35%, and the F1 score also reaches 65.35% (both slightly lower than the highest value of the base run at epoch 116). Regarding loss, however, the case with all seven features consistently performs as the best option.

In Figures 5.12 and 5.13, the confusion matrices associated with the best epoch of each of the two runs presented in this section are shown. To determine the best epoch for each case, the selection was based on the one with the highest F1 score in the validation set. Specifically, epoch 63 was chosen for the case with three features and epoch 43 for the one with six features. It can be observed that the matrix with six features performs slightly better in identifying both true positives and true negatives, but their behavior is similar in any case.

When comparing these two matrices with the one from the baseline run, also on the test set in Figure 5.8, it can be stated that in the baseline run, the network is more adept at identifying true positives compared to the cases with three and six features. However, concerning true negatives, the latter two runs perform better. In Table 5.3, the results obtained in the test set from each run using the weights of its best epoch are reported in each row. The results indicate that having more information in the data, as one would logically expect, leads to slightly better metrics.

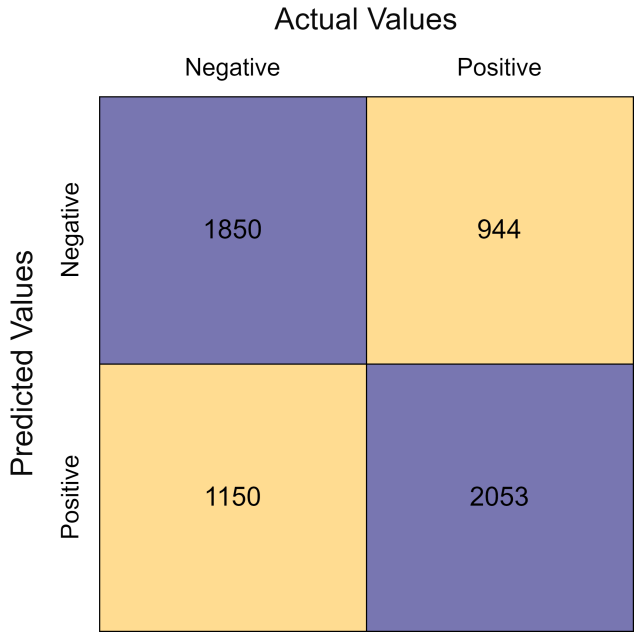


Figure 5.12. *Confusion matrix of the test set for the case of the data with 3 features.*

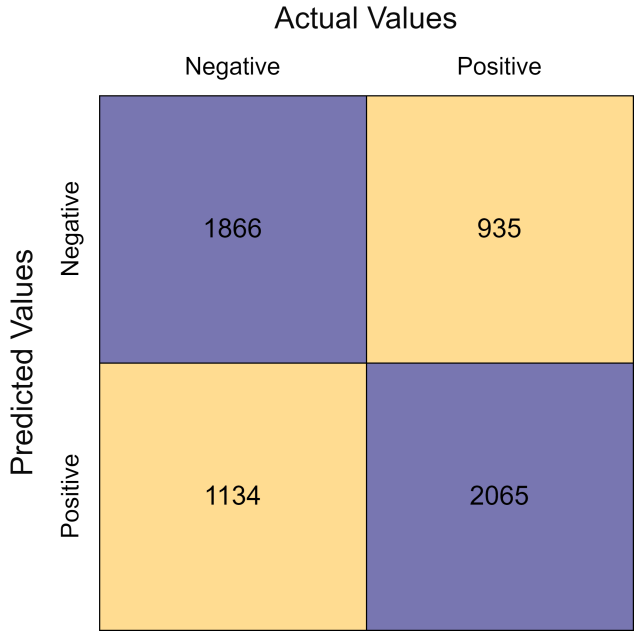


Figure 5.13. *Confusion matrix of the test set of the data with 6 features.*

It is important to consider that, although there is a minimal difference between having three and seven features when looking only at the best epochs, examining the previously presented graphs reveals a clear distinction in average performances, respecting what is indicated in the scientific literature about the importance of normals nx, ny and nz.

As mentioned earlier, the difference between having six and seven features is almost negligible, both in the metrics of the table and when observing the graphs. Therefore, the seventh feature, the triangle area, can be removed from the original data, as it does not contribute to increased performance, but rather burdens the data by increasing the memory required by the network.

Number of Features	Loss	Accuracy	F1	AUC
3	0.628	0.650	0.649	0.698
6	0.621	0.655	0.655	0.709
7 (Base Run)	0.618	0.657	0.657	0.715

Table 5.3. *Metrics on the test set.*

5.2.3 Removing Random Rotations

A desirable property for a model that operates on data in the form of a point cloud, as mentioned earlier, is invariance under rigid transformations. Since data from the scanning of real-world fragments lacks a privileged orientation, rotational invariance of the input is necessary for a neural network designed to utilize such data. In this work, as discussed in the previous chapter, to achieve rotational invariance, the noise was applied to the input data, with the noise being random rotations (Figure 5.14). This way, the model does not have access to information from any privileged directions present in the data and should learn to generalize better regarding the orientation of the fragments.

(a) Before **(b)** After

Figure 5.14. *Before and after applying a random rotation to the Fragment 1 from Couple 1.*

To verify this, an experiment was conducted in which data augmentation was not applied during the training phase. The values of the other hyperparameters are

the same as the first run, except for the number of epochs (100) and the batch size (16). Table 5.4 presents the results of the 4 possible cases that could occur with combinations of training and test data. When referring to the model trained with *Augmented Data*, it pertains to epoch 116 of the base run. Instead, for the model trained with *Original Data*, it refers to epoch 98 of the model trained for this section. From the results presented, it can be noted that there is an effect, albeit small, on performances: the model trained without data augmentation, when evaluated during inference on the test set where random rotations are applied, seems to have less effective predictions compared to when evaluated on the original data. In the case of the other model, however, the difference between the two inference scenarios appears to be negligible. It exhibits the same performance regardless of the orientation of the point clouds, thus achieving the goal of making it rotationally invariant.

Train Data	Test Data	Loss	Accuracy	F1	AUC
Original Data	Original Data	0.613	0.660	0.659	0.72
Original Data	Augmented Data	0.625	0.648	0.644	0.707
Augmented Data	Original Data	0.618	0.657	0.657	0.715
Augmented Data	Augmented Data	0.617	0.659	0.659	0.715

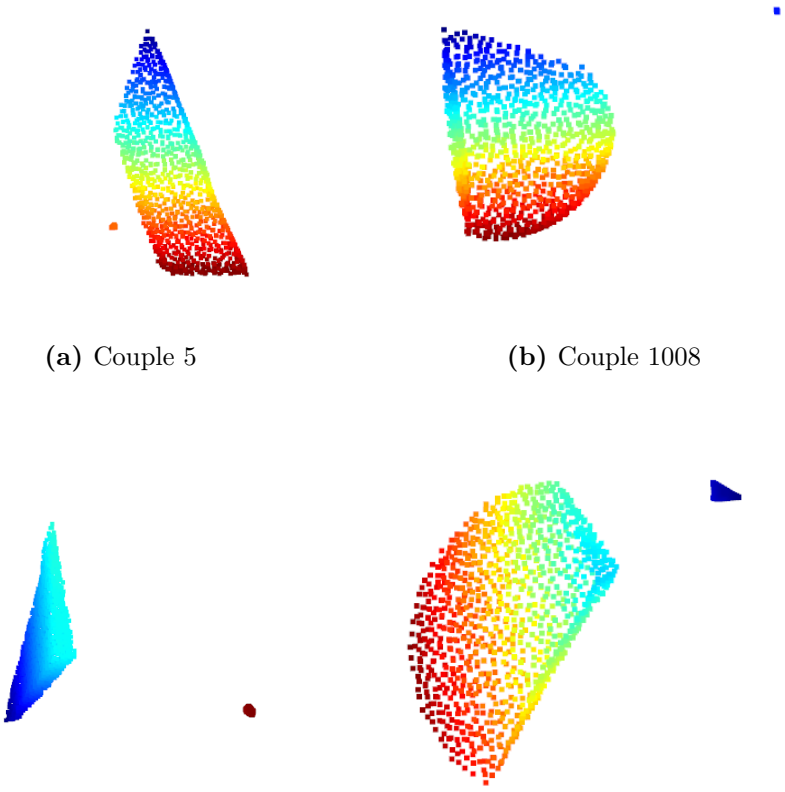
Table 5.4. *Comparison of performances in the case where data augmentation is applied or not applied to the data.*

5.3 Considerations on model predictions

It was decided to investigate more deeply into the predictions made by the network, using the weights from epoch 116 of the base run. The objective was to scrutinize the behavior and decisions of the model through a graphical analysis of the two elements that make up the various pairs, to understand if the shapes of the fragments influence predictions.

Examining various pairs, one of the initial observations is the presence of some "peculiar" cases. In 10% of the pairs, with rough inference, one of the fragments is significantly larger than the other and the associated label is often 0 (nonadjacent). Examples of these cases are shown in Figure 5.15. The neural network tends to predict these types of pairs quite accurately. However, these situations could have a negative impact on the model by "contaminating" the data and leading it to frequently predict the value zero when it encounters elements with such disproportionate sizes. Furthermore, it is conceivable that for an archaeologist, in scenarios where there are many large fragments and only a few smaller ones, accurately predicting whether an extremely small fragment is adjacent to a much larger one may have limited

significance for the overall reconstruction of the object.



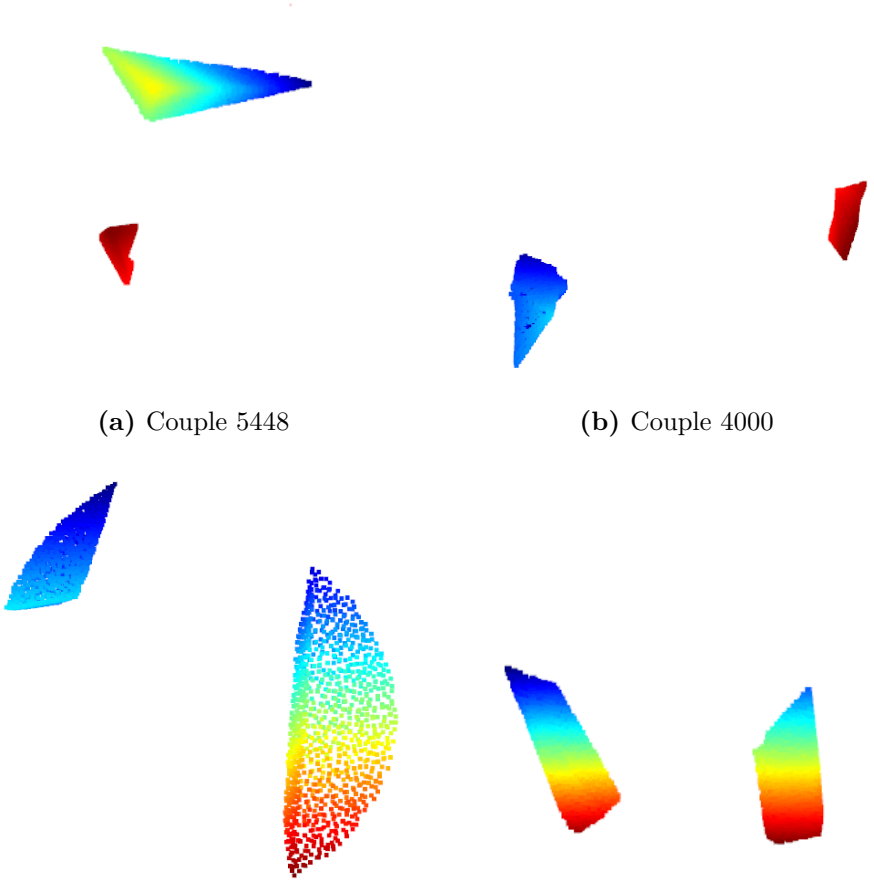
(a) Couple 5 **(b)** Couple 1008

(c) Couple 5411 **(d)** Couple 3020

Figure 5.15. *Representation of some bizarre pairs, in which one of the two elements is significantly larger than the other. In all the depicted figures, both the true label and the one predicted by the model are equal to zero.*

Another interesting observation is that the network seems to rely on the similarity of the shapes and sizes of the two fragments to make its predictions. After observing several pairs, their labels, and the model's predictions, it is possible to get an idea of what the network will predict by observing the shape and proportion between the two point clouds. Of course, there are exceptions: situations where very different fragments are correctly identified as adjacent (Figure 5.16 Panel A), as well as pairs of similar point clouds that are correctly classified as nonadjacent (Figure 5.16 Panel B). However, the predictions seem to be based on the similarity of the fragments, as can be seen in Figure 5.16 Panel C and D.

Were this assumption to prove correct, the model's strategy of assessing the similarity between the two point clouds would make sense. This approach could be analogous to a puzzle-solving strategy, where the attempt to join the pieces begins by looking for pairings between the most similar pieces. Such logic is consistent, since in reality, during the reconstruction of a fragmented object, it is likely that the most similar pieces are those that are close together. However, it is important to note that this consideration is closely related to the archaeological context of reference.



(a) Couple 5448 **(b)** Couple 4000

(c) Couple 1071 **(d)** Couple 5426

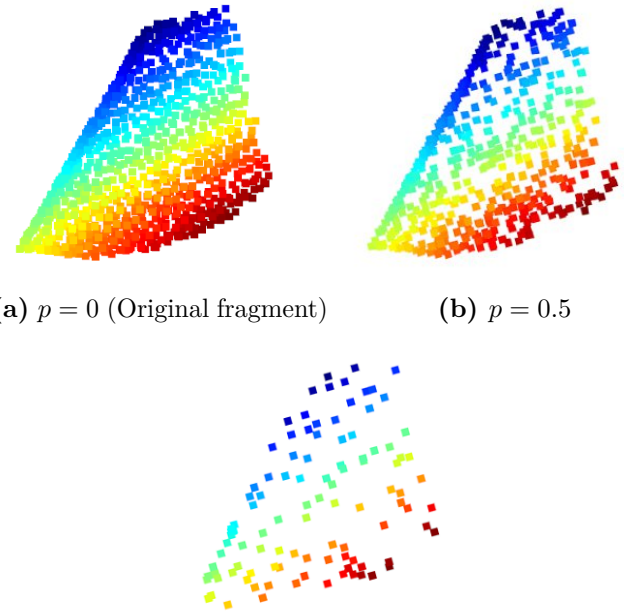
Figure 5.16. *(a) Couple 5448, where the elements have different shapes and sizes, but both the true and predicted labels are equal to one; (b) Couple 4000, where the elements are similar, but both the true and predicted labels are equal to zero; (c and d) Couples 1071 and 5426 are both composed of similar elements, the model correctly predicted that they are both adjacent.*

5.4 Robustness Analysis

This section reports on experiments where the focus is on applying changes to the input data to assess the effect there is on the model's performance. This can be interpreted as an investigation of both the robustness of the data in being able to retain information and the ability of the neural network to perform well in cases where it has access to fewer distinct points on the fragment surfaces. The model used is the base run, and the weights are from epoch 116.

5.4.1 Substituting random points

The first analysis performed is to evaluate the effect of replacing the seven features belonging to randomly chosen points from the point clouds with the averages of their respective columns to assess the impact on model performance and predictions. In this section, p represents the percentage of changed points. Random modification should help to avoid the concentration of substitutions in specific areas of the object, maintaining global information.



(a) $p = 0$ (Original fragment) **(b)** $p = 0.5$

(c) $p = 0.9$

Figure 5.17. *Before and after applying substitutions with the mean for randomly sampled points in the Fragment 1 of Couple 0.*

In Figure 5.17, an example of this procedure has been illustrated in which the original fragment (Panel A), the same after it has been modified with p equal to 0.5 (Panel B), and one in which the parameter is equal to 0.9 (Panel C) are represented. With half of the points modified, it is evident that the point cloud still retains many details about the shape of the object represented, while in the last case, it is hard to fully recognize the peculiarities of the fragment. Inference was then made on the test set modified by p, varying this parameter equally in steps of 0.1 in the range of 0.1 to 0.9. During the process, different model metrics and the confusion matrix were monitored to observe how the distribution of predictions between the two classes changed.

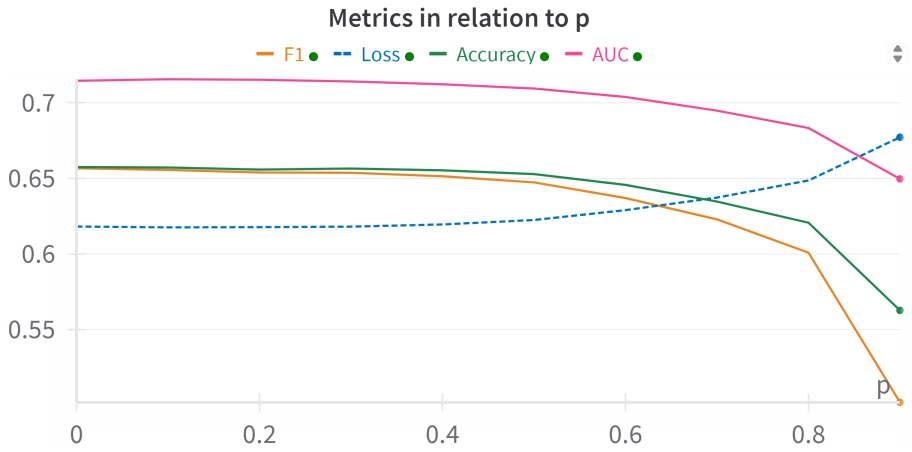


Figure 5.18. *Comparison of metric values based on p.*

Figure 5.18 shows a graph representing the performance of the neural network as the percentage of points changed. Some interesting things can be gleaned: between the range 0.1 and 0.4 of p, the 4 metrics used, maintain almost the same value as when no change is made to the data. Subsequently, as 50% of the points are changed, there is a drop in model performance, with a drop of about 1% on each metric for every additional 10% of points changed. The metrics experience a significant fall when p reaches a value of 0.9, with an accuracy of 0.56, F1 of 0.50, AUC of 0.65 and an increase in loss to 0.68.

It was also intended to track the distribution of the two classes predicted by the network as the number of points changed in the data. In Figure 5.19 it is possible to observe it. As was the case with the metrics, again, between the range 0.1 and 0.4 the changes are almost negligible, going from 55% of adjacent pairs predicted in the case with the original data to 60%. The increase continues by about 2% with each increase in modified points, attesting to 68% when p reaches the value 0.7. The model finally comes to predict 85% of data as adjacent and only 15% as nonadjacent

when the data are changed in 90% of their points.

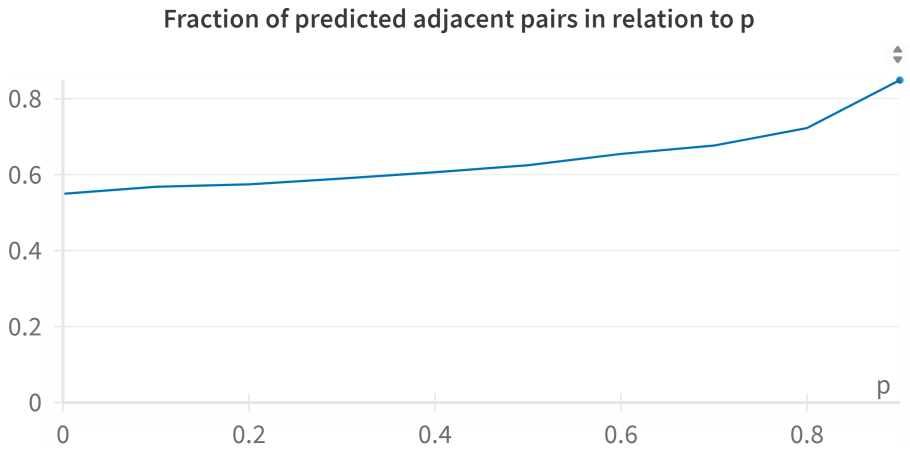
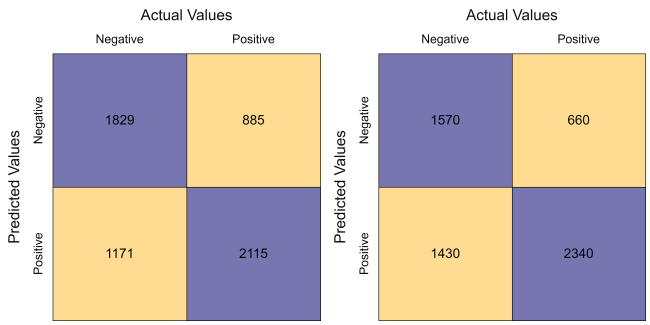


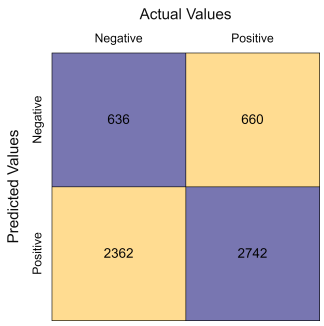
Figure 5.19. *Fraction of the predicted ones as p varies.*

Going into more detail, the three confusion matrices associated with the cases where p is equal to 0, 0.5 and 0.9, respectively, have been shown in Figure 5.20.



(a) $p = 0$

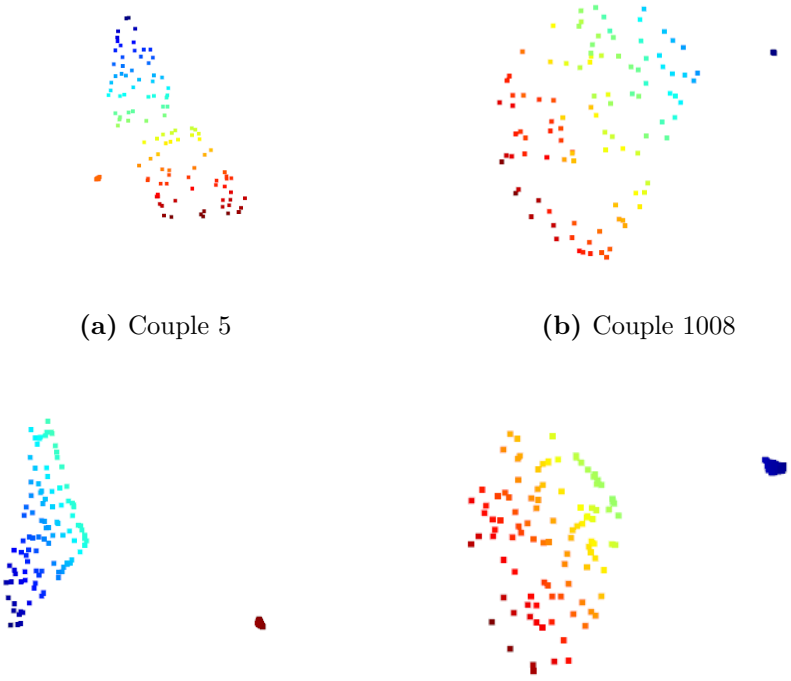
(b) $p = 0.5$



(c) $p = 0.9$

Figure 5.20. *Three confusion matrices are shown, each associated with a different p.*

Therefore, based on the analysis conducted, it can be said that the network maintains substantially unchanged performance as long as the variation in the data exceeds 40 percent of the points. After that, as p increases, there is a progressive loss of information that affects the performance of the model. The inclination to predict as adjacent two fragments gradually increase, the general shapes lose representativeness, turning into increasingly "anonymous" fragments.



(a) Couple 5 **(b)** Couple 1008

(c) Couple 5411 **(d)** Couple 3020

Figure 5.21. *Representation of the same pairs from Figure 5.15 after the modification with p equal to 0.9. The model correctly continues to predict pairs 5 and 1008 as non-adjacent, while it makes an error in predicting pairs 5411 and 3020 as adjacent.*

Most of the data that the network continues to correctly predict as nonadjacent, even when the value of p is 90%, are mainly of the type illustrated earlier in Figure 5.15. This phenomenon could be attributed to the fact that although changing most of the points leads to a loss of information, the substantial differences between the two elements forming the pair still allow the two fragments to be distinguishable, as demonstrated in the cases presented in Figure 5.21. It is evident that for the smaller point cloud of the pair, which is characterized by values in extremely narrow ranges, the change does not produce a significant change when compared with the effect on the larger element.

5.4.2 Substituting selected points

The present analysis, unlike the previous approach, in which points were randomly sampled, involves the targeted selection of certain points of the point cloud i based on the axis $\theta \in (x, y, z)$, modifying the surfaces according to a specified range of coordinates. To select the points to modify S_{θ_i} dynamically and adaptively for different point clouds, which are characterized by considerably heterogeneous shapes and sizes, the choice of the two extremes is handled as follows in Equation 20:

$$\begin{aligned} S_{\theta_i} =& [min(\theta_i) - 1 < \theta < min(\theta_i) + \epsilon_{\theta_i}] \\ \epsilon_i =& \frac{max(\theta_i) - min(\theta_i)}{f} \end{aligned} \tag{20}$$

The lower interval is represented by the minimum of the column of the reference axis from which one is subtracted, to be sure to select it, while the upper interval is the sum of the minimum and ϵ. The latter value is determined by the ratio of the interval width to the parameter f, which is inversely proportional to the amount of points to be changed. A higher value of ϵ results in a larger range of points selected for modification. The points were replaced with other ones that do not fall in the range. The option of replacing them with the mean was also explored, but the results were the same.

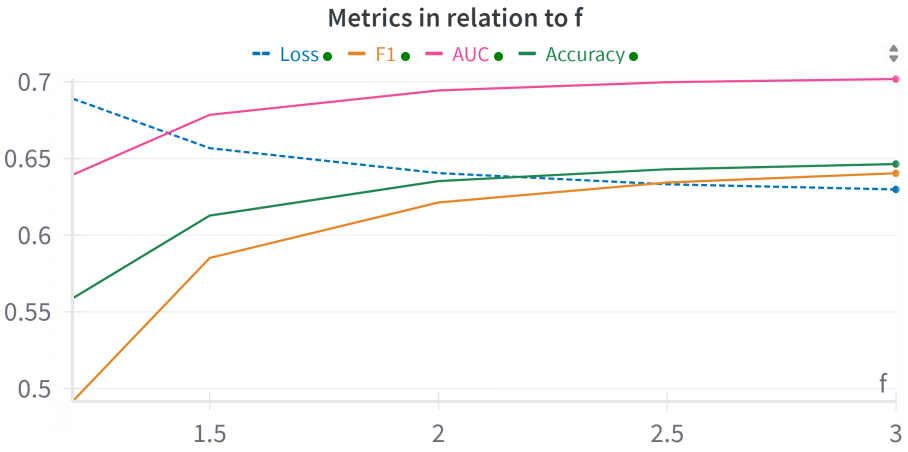


Figure 5.22. *Comparison of metric values based on f.*

Figures 5.22 and 5.23 show the metrics and the percentage of adjacent pairs predicted as the parameter f changes. Since regardless of the axis chosen the average performances of the model, holding the f value fixed, are more or less constant, an arithmetic mean of the axes has been plotted in the graph for each value. The cases

where $f \in \{3, 2.5, 2, 1.5, 1.2\}$ were evaluated.

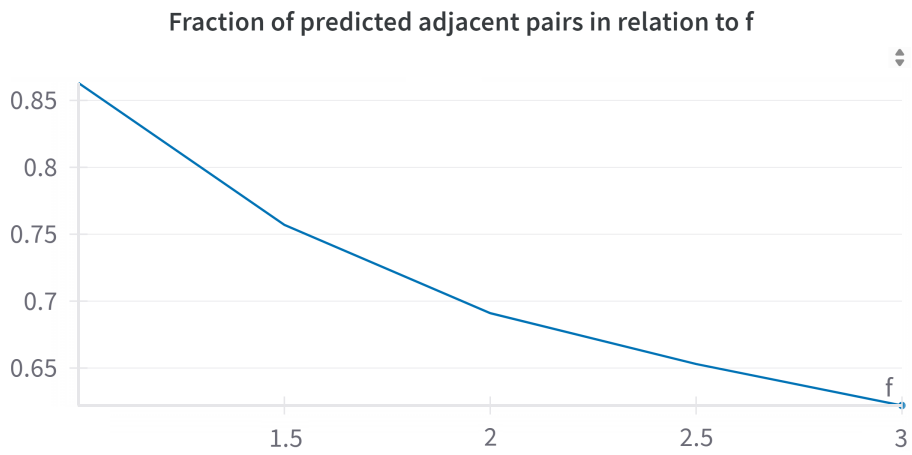


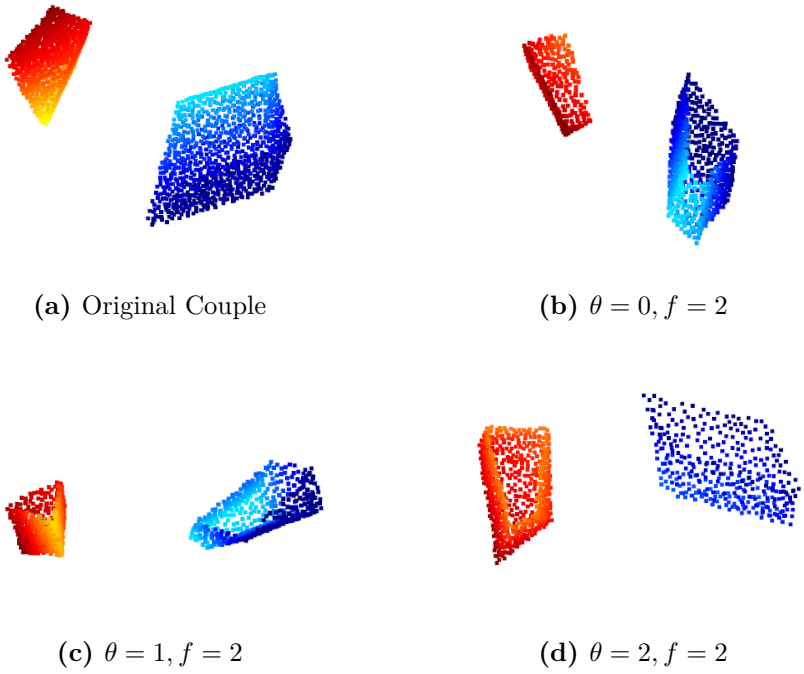
Figure 5.23. *Fraction of the predicted ones as f varies.*

As in the previous case, as the number of modified points increases, the network raises the errors by predicting more and more pairs adjacent. Although, in this way, the worsening seems to be faster than before. For example, when $f = 1.2$, that is, with about 83% of the points changed, the model performs slightly worse than when 90% of the randomly sampled points are changed. The same for the number of ones predicted, 0.86 compared to 0.8 out of the total in the case of the previous analysis. This effect can be attributed to the targeted modification of surfaces, which leads to the loss of the global information of the object earlier. Again, it can be seen that most of the data correctly predicted as nonadjacent, even with a significant amount of changed points, are the pairs that fall into the same category as those shown in Figure 5.15. This occurs when one of the two elements is markedly larger and different from the other. The interpretation of this phenomenon remains in line with what was observed in the previous analysis.

An interesting phenomenon is observed in some data when the value of f is held constant: by varying the axis chosen as the reference for the change, the model alters the prediction of the pair. For example, with f fixed at 2, it is observed that only in 80% of cases do the predictions remain constant on all three axes. This percentage decreases as the amount of points removed increases.

It is particularly noteworthy that, through the modification of certain surfaces, the network can correct some erroneous predictions made initially when it had access to all distinct points in the original data, during epoch 116 of the base run of this thesis. For example, considering the $f = 2$ example, there are 507 pairs in which the neural network was able to correct the predictions in at least one of the

three cases where the axis selected for modification changes. Figure 5.24 shows the pair 5595 either in the original version (Panel A) or as θ changes, holding constant $f = 2$ (Panels B, C and D). Recalling that this pair is adjacent, The model, however, succeeds in predicting it correctly only in Panel c, which is when the data experiences a change in the axis y ($\theta = 1$).



(a) Original Couple **(b)** $\theta = 0, f = 2$

(c) $\theta = 1, f = 2$ **(d)** $\theta = 2, f = 2$

Figure 5.24. *(a) Representation of the Original 5595 Pair and (b, c, d) after modification with the parameters shown. The network correctly predicts the pair as adjacent only in the case represented in (c). The images are oriented to show the "camera" the surface that has been removed.*

In the above example, it could be that halving the width of the y-axis interval led the two fragments to lose points which caused the model to identify these point clouds as too dissimilar to be adjacent. Regarding the data experiencing this phenomenon, it could be that through the modification of specific surfaces, the two fragments lose the points that initially drove the network to make errors. In other words, their shape transforms such that the architecture can predict rightly.

5.4.3 Substituting selected points with generated ones

The last analysis to evaluate robustness is to change the coordinates of some selected points, but instead of replacement by the mean or coordinates of existing points, this time new values are randomly generated that alter the point cloud. The procedure for selecting points remains the same as in the previous section, with the parameter

earlier named f now renamed g to distinguish it from the former analysis while retaining the same function. Regarding the random generation part, two approaches were developed: the first where the three features of the normals and the one related to the area of the mesh triangle were kept intact, changing only the spatial features. In the other case, all features of the selected points underwent modification. Changes to the column related to the chosen axis θ, will involve only the addition or subtraction of the column mean multiplied by a random weight between zero and one.

The operation is carried out as follows: for each row, i of the tensor D to be modified, a row containing noise generated by random values (R) is added or subtracted, depending on the result of a binary random value p. Each array of R is subsequently multiplied by the product of two weights a and b. The choice of the values of the weights is based on several trials performed, as an attempt was made to avoid excessively drastic point transformations. This process is summarized by Equation 21:

$$\begin{aligned} D_{\text{new}_i} &= D_i + p \cdot R_i \cdot a \cdot b \\ p &\in \{-1, 1\} \\ a &\in [0, 1] \\ b &\in [0.1, 0.4] \end{aligned} \tag{21}$$

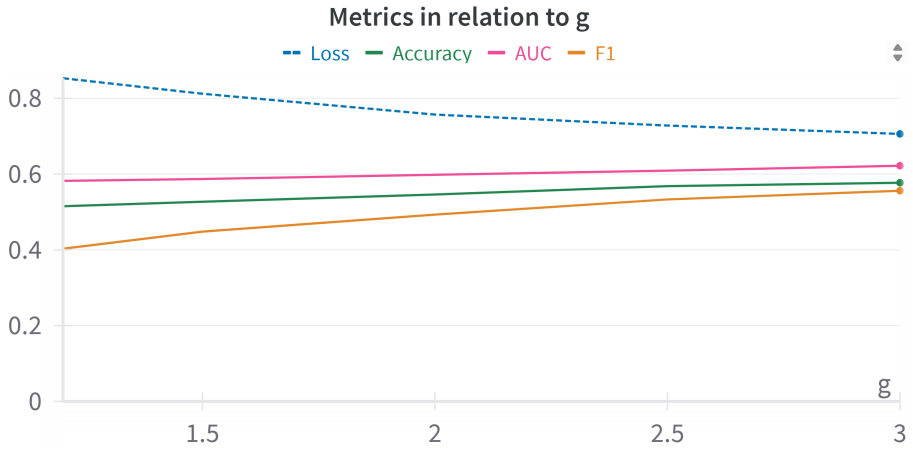
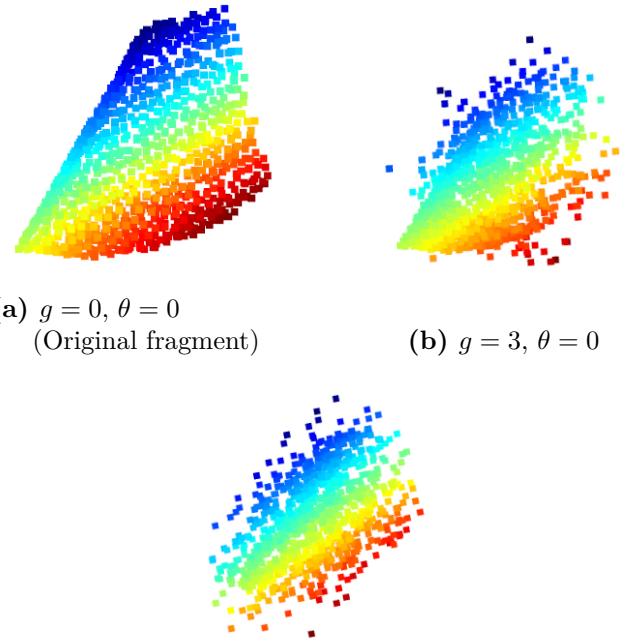


Figure 5.25. *Comparison of metric values based on g.*

In Figure 5.25 the metrics are monitored as the amount of points altered changes, in the case when only spatial features experience the modification. In fact, in the other situation, namely when the modification affects every coordinate, already at $g = 3$ the performance of all metrics plummets precipitously: they are all worse than

when 90% of the points are removed randomly and when $f = 1.2$ in the previous section. The results immediately reach 55% accuracy, 42% F1 and 57% AUC. They have not been reported graphically, as they remain fairly constant as the parameter g changes. Whereas, with only the modification of the three spatial features, the values are slightly higher than the previous ones but still low. Indeed, in turn, it is found, again for $g = 3$, 58% Accuracy, 55% F1 and 62% AUC. As in the previous experiments conducted in this section, a deterioration in the performance of the model as the modified points increase is evident, although in this case, the decline of metrics is the fastest.



(a) $g = 0$, $\theta = 0$ (Original fragment)

(b) $g = 3$, $\theta = 0$

(c) $g = 1.2$, $\theta = 0$

Figure 5.26. *Before and after applying the changes to the coordinates of the selected points in Fragment 1 of Couple 0.*

All this indicates that the architecture built is not suitable for handling data that undergoes this significant change. It would appear that the network no longer recognizes the modified point clouds as belonging to the same category as those used during training. The model was trained on fragments made in a certain way, which had a clear shape and with the points close together. These characteristics are compromised by the modifications made. Of the three tests conducted to evaluate robustness, this one was the most challenging. As shown in Figure 5.26, after modification, the fragments show significant differences from the original configuration, with the appearance of numerous isolated points. This situation likely limits the ability of the neural network to understand the shapes.

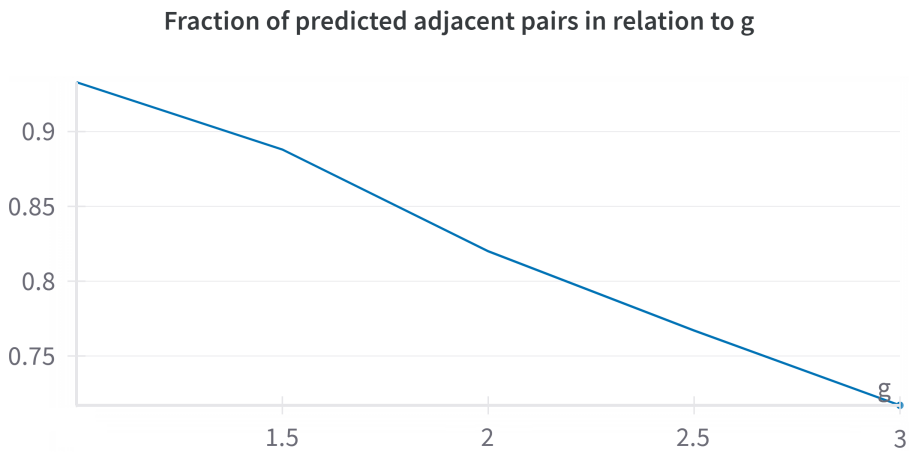


Figure 5.27. *Fraction of the predicted ones as g varies.*

Taking the shapes of the various fragments to extremes, some pairs were at any rate predicted to be nonadjacent, as can be seen in the graph shown in Figure 5.27. The point clouds, undergoing this important modification, tend to have similar shapes and characteristics, even with a small portion modified.

This analysis reveals a significant application for the current work. Studying the couples that the model keeps predicting as zeros, it turns out that they are mainly of the particular type, where one of the two elements is significantly larger and different than the other. In the few examples predicted as zeros with $g = 1.2$, these pairs constitute almost the whole. The concentration is significantly more pronounced than observed in the two previous analyses with $p = 0.9$ and $f = 1.2$.

Therefore, this experiment could be used as a tool to identify and remove such particular data from the dataset. In addition, it is possible to detect these pairs with different levels of sensitivity by varying the parameter g.

Conclusions

This work followed the growing trend in the integration of artificial intelligence and archaeology, highlighted in *Chapter 2* through the studies reported. From this emerged the ability of deep learning architectures to perform well in tasks such as detecting bone surface marks, classifying artistic styles present on fragments, and locating via aerial imagery hidden archaeological sites. Early attempts regarding the idea of fragment matching were also reproduced. The same chapter also reported on the scientific literature on models capable of handling 3D data, and on the work done by Alessandro Baiocchi, of which this thesis can be seen as the extension and continuation.

In *Chapter 3*, the methods used in this work were presented. Initially, it proceeded with the description of the data, starting with the definition of a point cloud and reaching the presentation of the structure of the *RCD* dataset used, which came from the *Broken3D* archive. This description revealed the marked unbalanced nature in the distribution of the two classes belonging to the dataset. Next, a concise but detailed explanation of how a neural network architecture works was reported and then concluded by describing the model used as the starting point of this thesis, the Point Cloud Transformer (PCT).

The *Chapter 4* focused on the experimental setup adopted in this study. Initially, the architecture of the dual-branch neural network employed to perform the task was explained. Next, the values assumed by the selected hyperparameters and parameters were given. In addition, the undersampling strategy adopted to handle data characterized by high unbalance and considerable computational burden both in terms of memory and required computing power was justified and presented.

The results of the conducted experiments are presented in *Chapter 5*. Initially, the different runs performed are outlined along with the related hyperparameters. Using all available features, good results were obtained, including an Accuracy and F1 of 66% and an Area Under the Curve (AUC) of 71%. Next, the influence of

variation in the number of features on network performance was examined. It shows the importance of normals in the task and the possibility of optimizing computational resources and memory by excluding triangle areas without compromising performance. Further insight is given into the positive impact of data augmentation on the rotation invariance property of the architecture. Next, model predictions are explored through a graphical analysis of the data. From the results that emerged, it can be hypothesized that the model tends to strongly exploit the similarity between the elements of the pairs to make the predictions. In addition, anomalous pairs are noted in which one of the fragments is significantly larger and different from the other, possibly contaminating the data and the network during the training phase, as they are almost always nonadjacent. In further experiments, the robustness of the model was evaluated by modifications to the point clouds. Changing randomly sampled points showed that the data retained information longer as the number of changes increased. Removing specific areas accelerates the descent of metrics, and some predictions vary with the modified surfaces. Finally, the application of random noise to point coordinates showed that the network begins to exhibit poor performance even with a very small number of altered points. The robustness studies, especially the last one, suggest that they can be used as a filter to detect anomalous data. in fact, with more than 90% of the points modified, the pairs that the model continues to predict as nonadjacent are predominantly composed of bizarre data.

In conclusion, recalling again the novelty of this task, the following work has moved an important step forward in the field of fragment matching by introducing an innovative method based solely on the use of Transformer-type neural networks. The performances of the model are discrete, also given the task. The positive impact on the metrics of the three normals and random rotations was ascertained, while the areas of the mesh triangles were negligible. By graphically analyzing the point clouds, also under the lens of the three changes made, it can be hypothesized that the network tends to rely on the similarity of shape and proportion between fragments to make its decisions. Moreover, the editing operations may prove to be a valuable tool for identifying and removing the various anomalous pairs present in the datasets. Such couples, in fact, likely present a low degree of usefulness in archaeological research. It is likely to become customary in the future for teams of archaeologists to be joined by intelligent tools to accelerate operations, including those of correctly combining artifact fragments, thus helping to unearth objects of immeasurable historical, heritage and cultural value.

Bibliography

[1] L. D. Norris & M. G. Inghram, *Half-Life Determination of Carbon (14) with a Mass Spectrometer and Low Absorption Counter*, Physical Review 70, 772, 1946.

[2] Athos Agapiou, *Archaeology from space: Using Earth Observation data to unearth our past*, Research Outreach, 2020.

[3] Janet Cady, Mark Wilson & Ellen Greytak Parabon NanoLabs, Inc., *DNA Phenotyping on Ancient DNA from Egyptian Mummies*, 2021.

[4] Antreas Kantaros, Theodore Ganetsos & Florian Ion Petrescu, *Three-Dimensional Printing and 3D Scanning: Emerging Technologies Exhibiting High Potential in the Field of Cultural Heritage*, Applied Sciences, 13(8), 2023.

[5] Manuel Domínguez-Rodrigo, Gabriel Cifuentes-Alcobendas, Blanca Jiménez-García, Natalia Abellán, Marcos Pizarro-Monzo, Elia Organista & Enrique Baquedano, *Artificial intelligence provides greater accuracy in the classification of modern and ancient bone surface modifications*, Sci Rep 10, 18862, 2020.

[6] Leszek M. Pawlowicz & Christian E. Downum, *Applications of deep learning to decorated ceramic typology and classification: A case study using Tusayan White Ware from Northeast Arizona*, Journal of Archaeological Science 130(2):105375, 2021.

[7] Siyamack Sharafi, Sajjad Fouladvand, Ian A. Simpson & Juan Barcelo, *Application of pattern recognition in detection of buried archaeological sites based on analysing environmental variables, Khorramabad Plain, West Iran*, Journal of Archaeological Science Reports 8:206-215, 2016.

[8] Hector A. Orengo, Francesc C. Conesa, Arnau Garcia-Molsosa, Agustín Lobo, Adam S. Green, Marco Madella & Cameron A. Petrie, *Automated detection of archaeological mounds using machine-learning classification of multisensor*

and multitemporal satellite data, Proc Natl Acad Sci USA 117(31):18240-18250, 2020.

[9] Mehrnoush Soroush, Alireza Mehrtash, Emad Khazraee & Jason A. Ur, *Deep Learning in Archaeological Remote Sensing: Automated Qanat Detection in the Kurdistan Region of Iraq*, Remote Sensing 12(3):500, 2020.

[10] Marek Bundzel, Miroslav Jaščur, Milan Kováč, Tibor Lieskovský, Peter Sinčák & Tomáš Tkáčik, *Semantic Segmentation of Airborne LiDAR Data in Maya Archaeology*, Remote Sensing 12(22):3685, 2020.

[11] Jonah C. McBride & Benjamin B. Kimia, *Archaeological Fragment Reconstruction Using Curve-Matching*, 2003 Conference on Computer Vision and Pattern Recognition Workshop, Madison, Wisconsin, USA, pp. 3-3, 2003.

[12] F. Jampy, A. Hostein, Fauvet Eric, Olivier Laligant & Frederic Truchetet, *3D puzzle reconstruction for archeological fragments*, Proc. SPIE 9393, Three-Dimensional Image Processing, Measurement (3DIPM), and Applications 2015, 939308, 2015.

[13] Yao Hu, Guohua Geng, Kang Li & Wei Zhou, *Unsupervised segmentation for terracotta warrior point cloud*, 2022.

[14] Hongjuan Gao & Guohua Geng, *Classification of 3D Terracotta Warrior Fragments Based on Deep Learning and Template Guidance*, IEEE Access, vol. 8, pp. 4086-4098, 2020.

[15] Daniel Maturana & Sebastian Scherer, *VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition* In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 922–928, 2015.

[16] Charles R. Qi, Hao Su, Kaichun Mo & Leonidas J. Guibas, *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*, CoRR, abs/1612.00593, 2016.

[17] Charles R. Qi, Hao Su, Kaichun Mo & Leonidas J. Guibas, *Pointnet++: Deep hierarchical feature learning on point sets in a metric space*, CoRR, abs/1706.02413, 2017.

[18] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, & Justin M. Solomon, *Dynamic Graph CNN for Learning on Point Clouds*, arXiv 1801.07829, 2019.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser & Illia Polosukhin, *Attention is all you need*, Advances in Neural Information Processing Systems , page 5998–6008, 2017.

[20] Guo Meng-Hao, Cai Jun-Xiong, Liu Zheng-Ning, Mu Tai-Jiang, Martin Ralph R. & Hu Shi-Min, *PCT: Point Cloud Transformer*, Computational Visual Media 7, 187–199, 2021.

[21] Karim Abou Zeid, Jonas Schult, Alexander Hermans & Bastian Leibe, *Point2Vec for Self-Supervised Representation Learning on Point Clouds*, arXiv, 2023.

[22] A. Baiocchi, S. Giagu, C. Napoli, M. Serra, P. Nardelli & M. Valleriani, *Artificial neural networks exploiting point cloud data for fragmented solid objects classification*, Machine Learning: Science and Technology, 2023.

Acknowledgments

I would like to express my heartfelt gratitude to Professor Scardapane, my thesis advisor, for the knowledge imparted during his lectures and for the immense support and kindness demonstrated throughout the writing of this thesis.

I am also thankful to Alessandro Baiocchi, my co-advisor, for his invaluable contributions and assistance during the development of this work.

I am deeply grateful to my parents and sister for their unwavering presence and support, especially in challenging times. They have guided me at every step of this journey. Without them, I could never have made it this far.

I thank my grandparents, even though two of them are no longer with us, for believing unconditionally in me from the very first day of school and for all the esteem and pride you have always felt toward me.

I am grateful to all my family members for their unwavering support over the years and for always being my advocates.

Last but certainly not least, I thank my friends, those who have been with me from day one and those I have met over the years. Together, we have celebrated moments of happiness and satisfaction, and we have consistently supported each other through times of struggle and adversity.

It seems to me right and dutiful to conclude this work, to which I have devoted the previous months, with the words of Coach Luciano Spalletti: "*Non è nelle stelle che è conservato il nostro destino, ma in noi stessi. Uomini forti, destini forti, uomini deboli, destini deboli. Non c'è altra strada.*".