Deep Learning Algorithms for Fragmented Solid Objects Classification

Pasquale Santaniello[1], Valerio Ponzi[1] and Roberta Avanzato[2]

[1]Institute for Systems Analysis and Computer Science, Italian National Research Council, Rome, Italy
[2]Department of Electrical, Electronics and Computer Engineering, University of Catania, Catania, Italy

Abstract

This paper presents a series of experiments conducted on an artificially generated dataset of 3D shapes, specifically focusing on fragments of larger objects. Developed in collaboration with the Ente Parco Archeologico dei Fori Imperiali, this research aims to create a system capable of recognizing and classifying objects from their fragments, with a particular emphasis on ancient artifacts. To achieve this objective, we explore various methods for classifying fragmented objects, identifying the most effective approach for this specific task. Although there are multiple techniques for 3D shape classification, this study centers on the PointNet network, which directly processes 3D point cloud data. This method is not only computationally efficient but also well-suited for handling irregular and unordered data structures, making it particularly advantageous over traditional techniques. Furthermore, we investigate the impact of data augmentation and noise injection strategies to enhance the model's robustness. A comparative analysis with state-of-the-art architectures is also provided. Finally, we present the trained models developed on our artificial dataset, demonstrating classification performance on par with the best existing solutions, and highlighting the potential of our approach in the domain of fragmented object recognition.

1. Introduction

Fragmented solid objects are prevalent across various domains in real-world scenarios, ranging from archaeological remains and geological samples to industrial debris analysis. The task of classifying these fragments, identifying whether they originate from external surfaces or internal structures, or associating them with their original object classes, is important for applications such as artifact restoration, digital reconstruction, and quality control.

Unlike complete 3D models, fragments present partial, often noisy representations of the original objects. Surface degradation, random break patterns, and loss of significant geometric features introduce severe challenges in analyzing fragmented solids. Furthermore, fragments may exhibit highly irregular shapes, missing structural continuity, and limited discriminative features, making conventional 3D classification techniques inadequate.

In practical applications, especially in cultural heritage preservation, the ability to automatically analyze and classify fragments could significantly accelerate the processes of cataloging, reassembly, and reconstruction of historical artifacts. However, building robust models for fragment classification requires addressing several inherent difficulties: the unordered and sparse nature of 3D data, variations in fragment size and orientation, and the need for generalization across different fragmentation patterns.

This study aims to tackle these challenges by developing a deep learning-based framework capable of classifying fragments of solid objects based on their geometric properties. By simulating fragmentation processes on synthetic objects and introducing variability in fragment shapes and surfaces, we create a controlled yet challenging environment to evaluate the performance of learning algorithms. The ultimate goal is to bridge the gap between synthetic experiments and real-world applications, providing tools that can assist domain experts in reconstructing fragmented objects from incomplete information.

2. Related Works

The classification of 3D shapes has been extensively studied, leading to the development of several deep learning architectures designed to process different representations of three-dimensional data. The most widely explored approaches include volumetric convolutional neural networks (volumetric CNNs) [1], multi-view CNNs [2], spectral CNNs [3], and point-based methods such as PointNet [4], as well as many other CNN based approaches [5, 6, 7, 8, 9, 10, 11].

Early attempts at 3D shape classification relied on volumetric CNNs, which operate on voxelized representations of objects [12]. These methods encode a 3D shape as a binary or real-valued 3D tensor and apply 3D convolutions to extract features. Although effective, volumetric approaches suffer from high memory consumption and computational complexity due to the sparsity of voxel grids, making them impractical for high-resolution models. More recent refinements, such as Vote3D [13], at-

SYSTEM 2025: 11th Sapienza Yearly Symposium of Technology, Engineering and Mathematics. Rome, June 4-6, 2025
ponzi@iasi.cnr.it (V. Ponzi); roberta.avanzato@unict.it (R. Avanzato)

tempt to mitigate these issues by improving the handling of sparse volumetric data, but overall efficiency remains a challenge.

Multi-view CNNs offer a different solution by rendering 3D objects into multiple 2D images from various viewpoints. These images are then processed using standard 2D convolutional networks, using well-established techniques from image classification. This approach has achieved state-of-the-art results in certain benchmarks, benefiting from the high performance of 2D CNNs. However, multi view methods introduce a trade-off: while they capture shape details effectively, they rely on predefined viewpoints, leading to potential information loss and requiring an extensive number of views to achieve robust performance.

An alternative approach is provided by spectral CNNs, which operate directly on 3D meshes by applying the Fourier transform [14]. This enables classification in the frequency domain, offering advantages such as spectral pooling, which efficiently reduces dimensionality while preserving important structural information. Nevertheless, spectral CNNs are primarily designed for manifold-based representations, making their application to non isometric shapes more challenging.

A more direct and flexible solution is to learn from raw 3D point clouds [15], which represent objects as unordered sets of points in space:

$$P = \{p_i \mid i = 1, \ldots, n\}, \quad p_i \in \mathbb{R}^3.$$

Each point p_i encodes spatial coordinates and, optionally, additional attributes such as color or surface normals.

PointNet represents a major breakthrough in this field, as it directly processes point cloud data without the need for intermediate projections or transformations. It is designed to be invariant to the ordering of points and effectively captures both local and global features through a symmetric function and a max-pooling aggregation step. Thanks to these properties, PointNet offers a highly efficient and accurate solution for 3D shape classification and segmentation, outperforming traditional approaches in both speed and performance.

Another advantage of PointNet lies in its computational efficiency. The following table (1) compares the number of parameters and floating-point operations (FLOPs) per sample for different architectures:

Architecture	Parameters (M)	FLOPs/Sample (M)
PointNet	3.5	148
SubVolume	16.6	3633
Multi-View CNN	60.0	62057

Table 1
Comparison of computational efficiency for deep architectures in 3D shape classification.

From this analysis, it is evident that PointNet offers a favorable balance between accuracy and efficiency, making it a strong candidate for real-time applications and large-scale 3D data processing. While Multi-View CNNs remain highly accurate, their computational complexity and memory requirements limit their practicality. Meanwhile, volumetric and spectral methods struggle with scalability, particularly for high-resolution models.

PointNet++ [16] extends the original architecture by incorporating hierarchical feature learning. By applying PointNet recursively on progressively refined subsets of points, PointNet++ captures local dependencies while preserving the efficiency and flexibility of the original model. This enhancement significantly improves performance on tasks such as object segmentation and classification.

Recent works have also explored the flexibility of graph-based architectures for 3D data, such as Graph Neural Networks (GNNs) [17, 18].

The next section provides a detailed overview of the PointNet architecture, discussing its main components and design principles in the context of fragmented object classification.

3. PointNet Architecture

Before explaining why PointNet was chosen as the framework for our classification task, it is essential to examine the properties of point clouds, which differ significantly from other 3D representations such as voxel grids or meshes.

A point cloud is defined as an unordered set of 3D points in $\mathbb{R}^n$, where each point is defined by its spatial coordinates and, in some cases, additional attributes such as color or surface normals. Unlike structured data formats such as images or voxel grids, point clouds lack an inherent spatial arrangement, which introduces unique challenges for deep learning models. One of the most significant properties of point clouds is their unordered nature. Since there is no predefined structure, a model designed to process point clouds must be permutation-invariant. This means that different orderings of the same set of points should yield the same result, ensuring consistency in processing. Another main aspect is the interplay between local and global dependencies. The geometric structure of a point cloud is dictated by the spatial relationships among its points. A well-designed model must be capable of capturing both fine-grained local details and broader global structures to accurately interpret the data. In addition, an effective classification model must exhibit invariance to rigid transformations such as rotations and translations. Regardless of how an object is placed in space, its fundamental characteristics should remain recognizable, ensuring reliable classification across different orientations. These properties

highlight the complexity of working with point clouds and the importance of designing specialized deep learning architectures that can effectively handle their unique characteristics.

PointNet was chosen for our classification task because of its ability to directly process raw point-cloud data while preserving the main properties described above. It introduces an innovative approach that avoids the computational overhead associated with volumetric and multiview methods. The network consists of three key components: a symmetric function for permutation invariance, a hierarchical feature aggregation mechanism, and a transformation network for spatial alignment. PointNet is designed to efficiently process unordered point cloud data by leveraging a symmetric function that ensures permutation invariance. Since the order of points in a point cloud is arbitrary, the network aggregates information through a symmetric operation. Specifically, it approximates a general function f defined over a set of points as:

$$f(\{x_1, \ldots, x_n\}) \approx g(h(x_1), \ldots, h(x_n)) \quad (1)$$

where $h : \mathbb{R}^N \to \mathbb{R}^K$ represents a feature transformation that maps each point to a higher-dimensional space, while $g : \mathbb{R}^K \times \cdots \times \mathbb{R}^K \to \mathbb{R}$ is a symmetric function, typically implemented as a max-pooling operation. This approach allows the network to extract meaningful point-wise features before aggregating them into a global descriptor, preserving permutation invariance.

After extracting point-wise features, a global max-pooling operation is applied to condense the most relevant information into a fixed-length vector. This step enables the network to generalize across different point clouds, making it particularly effective for classification and segmentation tasks. By focusing on the most salient features, the model can recognize geometric structures regardless of the input point order.

To further enhance robustness, PointNet incorporates a joint alignment mechanism that addresses the challenge of arbitrary geometric transformations in raw point cloud data. The network includes a transformation module that learns an affine transformation matrix and applies it to align input points, ensuring invariance to translation and rotation. This alignment improves the consistency of point cloud representations. To stabilize training, a regularization term is introduced to enforce the transformation matrix to be close to an orthogonal matrix:

$$L_{\text{reg}} = \|I - AA^T\|_F^2 \quad (2)$$

where A is the learned transformation matrix. This regularization helps maintain transformation stability, preventing unwanted distortions in aligned point clouds.

Given these properties, PointNet is particularly well suited for our task of classifying 3D shape fragments. The next section details the dataset used in our experiments.

4. Dataset

One of the main challenges in this research was the lack of publicly available datasets for fragmented 3D objects. To address this, we generated an artificial dataset consisting of 3D objects created using the open-source software Blender [19]. This dataset was designed to simulate the problem of classifying object fragments, a key step in reconstructing partial shapes.

The dataset comprises three main object categories: spheres, cubes, and icospheres. Each object was systematically fragmented using a Python script within the Blender environment. After fragmentation, each piece was exported in STL format and labeled as either external or internal. External fragments contain portions of the object's outer surface, while internal fragments originate from the object's core. To better approximate real-world conditions, random surface deformations were applied to external fragments, simulating natural aging effects. Consequently, internal fragments retained smooth surfaces, while external fragments exhibited rough and irregular patterns. Fragmentation was carried out using two distinct cutting strategies. The first approach, regular cutting, involved dividing the object at fixed intervals along the x, y, and z axes, resulting in uniformly shaped fragments. The second approach, random cutting, introduced variability in the division intervals, producing fragments of irregular sizes.

The dataset was structured into three different productions. The first production combined both regular and random cutting methods, yielding a total of 5,461 fragments. Among these, 1,130 were classified as internal fragments, with 564 originating from regular cuts and 566 from random cuts. The remaining 4,331 were labeled as external fragments, consisting of 2,164 generated through regular cuts and 2,167 through random cuts. The second production exclusively employed the random cutting method, generating a total of 7,259 fragments. Of these, 2,093 were categorized as internal fragments, while 5,166 were classified as external. Lastly, the Auriga fragmentation was conducted using a real object, the Auriga, which was fragmented specifically for testing purposes. This process produced only nine fragments, all of which were labeled as external.

A challenge in this dataset is the imbalance between external and internal fragments, with external fragments being significantly more numerous. This imbalance could negatively impact model training, leading to biased predictions. To mitigate this, we applied data augmentation techniques [20, 21], as detailed in the following section.

Production ID	Oversampling Factor	Internal Fragments	External Fragments	Total Fragments
3	2	4331	2260	6591
4	3	4331	3390	7721
5	4	4331	4520	8851
6	1	4331	(1130 + 2093)	7554

Table 2
Overview of final dataset compositions after applying oversampling and noise augmentation strategies.

5. Model

The classification network takes as input a set of n points, applies input and feature transformations, and aggregates point features through a max-pooling operation. The final output consists of classification scores over k classes.

The architecture includes two transformation networks. The first transformation network is a mini-PointNet that processes the raw point cloud and predicts a 3×3 transformation matrix. It consists of a shared multilayer perceptron (MLP) applied independently to each point, with output sizes 64, 128, and 1024. This is followed by a global max-pooling operation across all points and two fully connected layers with 512 and 256 units, respectively. The resulting matrix is initialized as the identity matrix. Except for the final layer, all layers use ReLU activations and batch normalization.

The second transformation network follows a similar structure but outputs a 64×64 transformation matrix, also initialized as the identity. A regularization term, weighted by 0.001, is added to the softmax classification loss to encourage this matrix to remain close to orthogonal.

Following the transformation steps, a global max-pooling layer aggregates the point-wise features into a compact global descriptor. This descriptor is then passed through an MLP consisting of three fully connected layers with 512, 256, and 2 units, respectively, for binary classification. In the case of multi-class classification, the final dense layer is modified to output 3 units instead of 2.

Dropout with a keep ratio of 0.3 is applied after the second-to-last fully connected layer (dimension 256). An additional dropout layer was added as it empirically improved model performance.

The network is trained using the Adam optimizer with an initial learning rate of 0.001, momentum 0.9, and a batch size of 32.

Several models were developed during the experimental phase. The main differences between them lie in the data augmentation techniques applied prior to training. Regarding binary classification, the best-performing model is referred to as Model 2 in the results. Model 3, Model 4, and Model 5 share the same architecture but were trained on different sample compositions. For the multi-class classification task, a slightly different model was used, where only the final MLP layer differs, adapting the output dimension to the three target classes.

6. Data Augmentation Techniques

To address dataset imbalance and improve model generalization, we applied various data augmentation techniques, including oversampling, noise injection, and dataset mixing. To enhance the performance of the model and address the imbalance of the dataset, we implemented several data preprocessing and augmentation strategies. First, we applied oversampling, replicating samples from the minority class (internal fragments) to balance the dataset. We experimented with different replication factors ($k = 2, 3, 4$) to assess their impact on model performance. Another strategy was random data shuffling, in which we shuffled the dataset after each training epoch. This approach minimized the variance of the batch composition and reduced the risk of overfitting, improving the generalization of the model.

To further enhance the robustness of the model, we introduced noise injection, adding small random perturbations to the coordinates of the points. This technique prevented the model from memorizing exact spatial patterns, thereby improving its ability to generalize to unseen data. Lastly, we used dataset mixing, combining fragments from different dataset productions. This increased data variability and allowed us to test the model's ability to adapt to different fragmentation strategies. The final dataset configurations used in the training are summarized in 2.

Similar strategies to enhance robustness and generalization through data transformation and noise-based techniques have been successfully applied in other domains, such as EEG signal processing and affective computing, including the use of GAN-based denoising [22], CycleGAN for cross-domain adaptation [23], and Transformer architectures for sentiment classification [24].

7. Noise Addition and Tolerance

Once we have obtained a very good model for the classification of the dataset, thanks to data augmentation techniques,we investigated how much noise (points are jitted on all three axes) could be inserted without changing the

performance of the network, and also if the procedure of adding noise increased or decreased the generalization capability of the network. This procedure is also called noise injection. The noise injection method refers to adding 'noise' artificially to the ANN input data during the training process. Jitter is one particular method to implement noise injection. With this method,a noise vector is added to each training case in between training iterations. This causes the training data to 'jitter' in the feature space during training, making it difficult for the ANN to find a solution that fits precisely the original training dataset, and thereby reduces the overfitting of the ANN. We are actually able to inject noise inside the dataset owing to a random uniform distribution applied on the training sample,varying the lower and upper bounds of our distribution to check tolerance and generalization capabilities. We tried to change different levels of jitter to actually try to understand when the model would collapse in terms of parameters. We tried ranges of uniform distribution from (-0.005,+0.005) to (-0.100,+0.100). This last interval could be considered the breaking point of performance of our model, as we will show in the results section. At testing time instead, we did experiments to check which jitter injection would cause the net to be capable of better generalizing when classifying a different dataset. Different noise ranges were tested 3.

Noise Range	Change of performances
(-0.005,+0.005)	Default case
(-0.010,+0.010)	Slow change of performances
(-0.050,+0.050)	Slow change of performances
(-0.100,+0.100)	High drop of performances

Table 3
Comparison of model complexity and computational cost (FLOPs per sample) for selected 3D shape classification architectures.

Then to test what range of noise will increase the generalization capability of the net, we created a new Dataset, to obtain a new Dataset that is very different from the previous one especially considering the minor class (the Dataset was balanced by adding random fragments).

The noise in the range of (-0.010,+0.010) will make our model (trained on the first dataset, different from the new one) achieve the same performances as a classifier trained directly on the new dataset (see the Result section for a better explanation). This could probably mean that this range of noise will not throw down performances on the dataset while maintaining a good generalization power on different Datasets. All the results are shown in the result section with attached confusion matrices.

8. Extension to Multi-Class Classification

One thing that we noticed during the implementation of our model,was actually based on how the Dataset was composed. As we saw in section, the Dataset used for training and testing the model was obtained by fragmenting objects starting from three classes: Sphere, Icosphere and Cube. Considering the fact that this kind of algorithm could be used in order to actually try to 'reconstruct' objects starting from its fragments as final goal, i.e. a 3D puzzle,we thought that one further step in order to extend the algorithm to this task was to try to implement a multi-class classification instead of a binary one, trying to deduce the 'family' of objects where each fragment comes from. For doing this we had to slightly modify first the Dataset composition and related label's extraction and then we also modified a bit the model in order to make it able to predict over the three mentioned classes,Sphere, Icosphere and Cube.

Before saying how training and test samples are obtained for this classification task, we have to say that since during our experiments we noticed that the 'internal' fragments are very similar between them also across different starting classes, using also this kind of fragments for the new classification task was too challenging for our application. That is why we decided to create two productions of samples made entirely by the 'external' fragments and use them to train and test the new model. So what we obtained are two productions: Production 1: made of all the external fragments of the first production in Section, obtaining 4331 external fragments. Production 2: we tried to increase the number of samples for training, and to do this we added all the 'external' fragments of the second production in Section obtaining 9497 'external' fragments. Labels : Now that we created a dataset made only of 'external' fragments, since they are the most 'discriminative' ones,we had to label each of these fragments with the original class where it was fragmented, and this was easy since in each fragment's name exported from Blender, there was the initial object where it was composed. So we could actually retrieve labels from the fragment's name just by processing the string of the name. The model used for prediction is very similar to the one described in Section, the only thing that changes is the actual last dense layer of the MLP since it has 3 units instead of 2. One possible solution to the problem, to actually extend the task of classifying fragments of object in the object they were composed, could actually be to introduce new families of objects that differ a bit from the other ones, or maybe try to apply some Data Augmentation making the model able to distinguish between Sphere and Icosphere during classification.

9. Results

In this section, we present all the results obtained during our experiments, providing comparisons and highlighting the most effective techniques for classifying 3D object fragments. We begin by reporting the accuracy achieved by our model without any data augmentation, using standard noise injection. The dataset used for training is Production 1, while the test set consists of a random subset of the training data. As a starting point for our experiments, we compare the performance of models trained on samples converted using Open3D and Trimesh.

Conversion Library	Sampling Points	Accuracy
Open3D	1024	86.18%
Trimesh	1024	85.00%

Table 4
Baseline PointNet model accuracy using different 3D conversion libraries (Open3D and Trimesh).

As the next step, we compared models trained with different conversion libraries, varying sampling densities, and applying oversampling as a data augmentation technique. The table below summarizes the configurations and corresponding accuracies achieved on the Production 1 dataset. Our goal was to train a competitive model capable of reaching accuracy levels comparable to more recent architectures like PointNet++, solely by balancing the dataset through oversampling.

Conversion Library	Sampling Points	Oversampling Factor	Accuracy
Trimesh	1024	3	89.26%
Trimesh	2048	3	88.74%
Open3D	1024	3	92.22%
Open3D	2048	3	89.90%
Trimesh	1024	2	87.57%
Trimesh	1024	4	83.12%

Table 5
Impact of conversion library, sampling density, and oversampling factor on model performance (Production 1 dataset).

Thanks to these techniques, we obtained a model that significantly improved its ability to classify fragments, including internal ones, compared to the initial production. Moving forward, we focus on identifying best practices for preventing overfitting. Before doing so, it is important to highlight an interesting observation made during testing. After training a model with samples converted using either Trimesh or Open3D, once the model has converged, the choice of conversion library for the test set no longer affects performance: both conversion methods yield identical results at test time. In practice, while the choice of conversion library can impact the model's performance during training, it does not influence the outcomes during testing.

Now we will consider experiments regarding the noise injection,in particular we are interested in obtaining the 'break point' of the net in which the noise is too much and performances of the net will go down steeply,and also in retrieving the range of noise that will grow up generalization performances of our network. We have to say that all these experiments are performed on the best performing model discovered until now , the one that reached 0.9222 of accuracy. In this table we show the comparison between various range of noise applied to the best performing model and the 'break point'. In this case the test set is obtained by a percentage of the Training set.

Jitter range	Accuracy
(-0.005,+0.005)	0.9222
(-0.010,+0.010)	0.8906
(-0.050,+0.050)	0.8964
(-0.100,+0.100)	0.8052

Table 6
Accuracy reached from the model with different jitter levels, to evidence how much rumor our model can sustain before breaking down performances

As observed, introducing noise within the range of (-0.100, +0.100) leads to a significant drop in performance, which we identify as the "breaking point" for noise injection in our application. Beyond this threshold, the model's classification accuracy deteriorates considerably. The next objective is to determine the optimal level of noise injection to improve generalization while avoiding overfitting. To this end, we exploit a different dataset, referred to as Production Final. Production Final differs from Production 1 mainly in the composition of internal fragments, as it aggregates internal samples from both the first and second productions, resulting in a more balanced dataset. To assess the impact of noise injection, we first train a new PointNet model from scratch on Production Final, treating it as a reference model. We then evaluate the best-performing model previously obtained (trained on Production 1) by applying various levels of noise injection, testing it on a subset of Production Final. By carefully analyzing the confusion matrices, we aim to identify the level of noise that enables the previously trained model to behave similarly to the new baseline model trained directly on Production Final. If a model trained with noise augmentation achieves similar classification patterns to the baseline, we can reasonably conclude that noise injection at that level enhances the model's generalization ability. In the following, we present the accuracy of the model trained from scratch on Production Final.

Now we test the best performing model (model2), trained with different levels of jitter, on the same test set made by Production Final, since we want to see which model acts more similarly to the Baseline model trained

Conversion	Dataset	Accuracy
Open3D	Production Final	0.8703
Trimesh	Production Final	0.8451

Table 7
Baseline model trained from scratch on a different Dataset, used to test generalization power

directly on the new Dataset.

Jitter range	TestSet	Accuracy
(-0.005,+0.005)	Production Final	0.8400
(-0.010,+0.010)	Production Final	0.8412
(-0.050,+0.050)	Production Final	0.8392
(-0.100,+0.100)	Production Final	0.7723

Table 8
Accuracy of the best performing model (model2) obtained on the new Dataset(the one tested with the Baseline model) with different jitter levels, trying to evidence which jitter level will make our model work as a Baseline model trained directly on that Dataset(that here we are using for testing),as to obtain some generalization performance parameter

We observe that the best-performing model, trained with a jitter range of (-0.010, +0.010), is able to classify samples from the new dataset almost as effectively as a model trained directly on it. Considering that the main difference between the two datasets lies in the internal fragments — which, according to the confusion matrices, are classified similarly by both models. We can conclude that injecting noise within this range slightly reduces performance on the original dataset, but enhances the model's generalization capabilities. Regarding the binary classification task (internal vs external fragments), the final evaluation concerns the predictions made by the best-performing model on the fragments of a real object (Auriga), as introduced in Section 2.

Auriga's fragment	Label	Prediction
Fragment n.1	External	External
Fragment n.2	External	External
Fragment n.3	External	External
Fragment n.4	External	External
Fragment n.5	External	External
Fragment n.6	External	External
Fragment n.7	External	External
Fragment n.8	External/Internal	External
Fragment n.9	External	External

Table 9
Predictions on real object's fragments, real objects are all labeled as external, except for the n.8 which seems to be 'hybrid', so predictions are correct

Last results to show are about the extension to Multi-class classification. As we said, we are using a new dataset and a slightly modified model . We first show results of the model trained on the external fragments contained in the First Production, without Data augmentation techniques, using the Trimesh library for conversion.

Conversion	Sampling number	Accuracy
Trimesh	2048	0.6701
Trimesh	1024	0.6756
Open3D	1024	0.6785

Table 10
Classification accuracy reached by models trained for multi-class classification

Problems with this kind of classification can be easily exploited by visualizing the confusion matrix:

	Cube	Sphere	IcoSphere
Cube	0.94	0.06	0.00
Sphere	0.00	0.99	0.01
Icosphere	0.00	0.98	0.02

Table 11
Confusion Matrix of the Model used for multi-class classification

We can see how the Cube class classification works well, while all the samples of Icosphere are classified as Sphere.

10. Conclusion

In this work, we presented a series of experiments focused on the classification of 3D object fragments using deep learning techniques. Starting from an artificially generated dataset of fragmented shapes, we evaluated the impact of different preprocessing methods, data augmentation strategies, and model configurations. Our experiments demonstrated that simple oversampling techniques, combined with the use of Open3D for point cloud generation and an appropriate sampling density, allowed us to improve the classification performance of the baseline PointNet architecture, reaching an accuracy comparable to more advanced models such as PointNet++. We also investigated the role of noise injection as a means to enhance model generalization. Our results suggest that controlled noise injection, particularly in the range of (-0.010, +0.010), enables the model to better adapt to new datasets without severely impacting performance on the original data.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT, Grammarly in order to: Grammar and spelling

check, Paraphrase and reword. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, L. J. Guibas, Volumetric and multi-view cnns for object classification on 3d data, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 5648–5656.

[2] H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller, Multi-view convolutional neural networks for 3d shape recognition, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 945–953.

[3] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, arXiv preprint arXiv:1312.6203 (2013).

[4] C. R. Qi, H. Su, K. Mo, L. J. Guibas, Pointnet: Deep learning on point sets for 3d classification and segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 652–660.

[5] R. Avanzato, F. Beritelli, M. Russo, S. Russo, M. Vaccaro, Yolov3-based mask and face recognition algorithm for individual protection applications, in: CEUR Workshop Proceedings, volume 2768, 2020, p. 41 – 45.

[6] F. Fiani, V. Ponzi, S. Russo, Keeping eyes on the road: Understanding driver attention and its role in safe driving, in: CEUR Workshop Proceedings, volume 3695, 2023, p. 85 – 95.

[7] G. Capizzi, G. L. Sciuto, P. Monforte, C. Napoli, Cascade feed forward neural network-based model for air pollutants evaluation of single monitoring stations in urban areas, International Journal of Electronics and Telecommunications 61 (2015) 327 – 332. doi:10.1515/eletel-2015-0042.

[8] N. Brandizzi, V. Bianco, G. Castro, S. Russo, A. Wajda, Automatic rgb inference based on facial emotion recognition, in: CEUR Workshop Proceedings, volume 3092, 2021, p. 66 – 74.

[9] E. Iacobelli, S. Russo, C. Napoli, A machine learning based real-time application for engagement detection, in: CEUR Workshop Proceedings, volume 3695, 2023, p. 75 – 84.

[10] V. Ponzi, S. Russo, A. Wajda, C. Napoli, A comparative study of machine learning approaches for autism detection in children from imaging data, in: CEUR Workshop Proceedings, volume 3398, 2022, p. 9 – 15.

[11] G. Lo Sciuto, G. Capizzi, R. Shikler, C. Napoli, Organic solar cells defects classification by using a new feature extraction algorithm and an ebnn with an innovative pruning algorithm, International Journal of Intelligent Systems 36 (2021) 2443 – 2464. doi:10.1002/int.22386.

[12] A. S. Gezawa, Z. A. Bello, Q. Wang, L. Yunqi, A voxelized point clouds representation for object classification and segmentation on 3d data, The Journal of Supercomputing 78 (2022) 1479–1500.

[13] D. Z. Wang, I. Posner, Voting for voting in online point cloud object detection., in: Robotics: science and systems, volume 1, Rome, Italy, 2015, pp. 10–15.

[14] R. N. Bracewell, The fourier transform, Scientific American 260 (1989) 86–95.

[15] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, M. Bennamoun, Deep learning for 3d point clouds: A survey, IEEE transactions on pattern analysis and machine intelligence 43 (2020) 4338–4364.

[16] C. R. Qi, L. Yi, H. Su, L. J. Guibas, Pointnet++: Deep hierarchical feature learning on point sets in a metric space, Advances in neural information processing systems 30 (2017).

[17] V. Ponzi, C. Napoli, Graph neural networks: Architectures, applications, and future directions, IEEE Access (2025).

[18] V. Ponzi, L. Comito, C. Napoli, Pnmlr: Enhancing route recommendations with personalized preferences using graph attention networks, IEEE Access (2025).

[19] L. Flavell, Beginning blender: open source 3d modeling, animation, and game design, Apress, 2011.

[20] C. Shorten, T. M. Khoshgoftaar, A survey on image data augmentation for deep learning, Journal of big data 6 (2019) 1–48.

[21] D. A. Van Dyk, X.-L. Meng, The art of data augmentation, Journal of Computational and Graphical Statistics 10 (2001) 1–50.

[22] I. E. TIBERMACINE, S. Russo, F. Citeroni, G. Mancini, A. Rabehi, A. H. Alharbi, E.-S. M. Elkenawy, C. Napoli, Adversarial denoising of eeg signals: A comparative analysis of standard gan and wgan-gp approaches, Frontiers in Human Neuroscience 19 (2025) 1583342.

[23] S. Russo, S. Ahmed, I. E. Tibermacine, C. Napoli, Enhancing eeg signal reconstruction in cross-domain adaptation using cyclegan, in: 2024 International Conference on Telecommunications and Intelligent Systems (ICTIS), IEEE, 2024, pp. 1–8.

[24] I. E. Tibermacine, A. Tibermacine, W. Guettala, C. Napoli, S. Russo, Enhancing sentiment analysis on seed-iv dataset with vision transformers: A comparative study, in: Proceedings of the 2023 11th international conference on information technology: IoT and smart city, 2023, pp. 238–246.