Segmentation of Heritage Building Point Clouds

Michele Pulvirenti
ML Student
ESILV
Paris, France
michele.pulvirenti@edu.devinci.fr

Guillaume Quere
Research director
A-BIME
Paris, France
guillaume.quere@a-bime.com

Ruiwen He
ML Teacher
ESILV
Paris, France
ruiwen.he@devinci.fr

***Abstract*—Point cloud segmentation can be a challenging task, especially when the point clouds are large and diverse. In this paper, two main Machine Learning techniques are presented to perform the segmentation. The first is a classic ML approach in which a classifier is trained to complete the segmentation in a point cloud. For the second approach, two Neural Network models are trained on a dataset and used to perform inference on new point clouds.**

***Index Terms*—segmentation, machine learning, neural network, deep learning, point cloud, heritage, building**

I. Introduction

The objective of this paper is to investigate on 3D point cloud segmentation techniques and build a working program that can segment point clouds of old heritage buildings. Two main segmentation approaches are presented and analyzed in detail. The dataset used is the ArCH Dataset [1], a collection of several point clouds of old heritage buildings. The crucial part in this work was to find the optimal data processing technique for training data that will then be applied to new and unknown point clouds.

II. State of the art

Semantic segmentation of 3D data can generally be approached through three principal methodologies:

- Projection-based methods: These techniques transform 3D point clouds into 2D representations (e.g., RGB images), enabling the use of 2D CNNs. SAM3D [2] extends this idea by projecting 2D masks from SAM onto 3D point clouds for zero-shot 3D segmentation.
- Voxel-based methods: In this approach, the spatial domain is discretized into volumetric grids that can be processed by three-dimensional convolutional neural networks (3D CNNs). Despite their effectiveness in certain contexts, voxel-based methods are less frequently used in cultural heritage applications because of their limited ability to preserve fine geometric details.
- Point-based methods: These methods operate directly on raw point cloud coordinates, avoiding the need for handcrafted feature extraction. Point-based techniques have emerged as the prevailing paradigm in contemporary point cloud segmentation research.

A. Academic Research Approaches

Early heritage segmentation used classical ML techniques on hand-crafted geometric features such as covariance-based features on spherical neighborhoods [3].

These supervised ML approaches (feature engineering + classifier) struggled to generalize due to the variety of architectural styles and details.

The advent of deep learning revolutionized point-cloud segmentation. Modern methods include:

- PointNets: PointNet [4] and PointNet++ [5] were among the first networks to feed raw point coordinates through neural networks. They capture global and local shape but have limited ability to model a fine-grained context by themselves.
- Graph-Based CNNs: DGCNN [6] constructs a dynamic k-NN graph and applies EdgeConv to learn local patterns. These networks can diffuse information across the point set and have become very popular in heritage research.
- Heritage-specific adaptations: Researchers often augment networks with domain cues. Pierdicca et al. (2020) [7] applied DGCNN to ArCH by adding HSV color and normal vectors as input features. Matrone et al. (2020) [1] further added geometric features such as planarity and verticality and compared various classifiers on ArCH, improving accuracy. Other strategies include data augmentation and transfer learning. These efforts underscore that standard network architectures often need heritage-specific tailoring.

B. Challenges and Open Problems

Despite rapid progress, key gaps remain.

A major issue is data scarcity: ArCH (17 labeled scans) is one of the very few datasets explicitly for heritage. Most segmentation models are trained on small, domain-specific datasets and may not generalize across eras, styles, or sensor modalities. Complex architectural details (carvings, ornaments) and damage (erosion) can cause over-segmentation or misclassification.

A recent work has shown that synthetic point clouds generated through diffusion-based methods can further improve model performance by enriching training datasets with realistic, diverse samples [8].

This approach is particularly beneficial in architectural and construction applications, where obtaining large volumes of

labeled point cloud data is often impractical. By leveraging synthetic data, models can achieve better generalization and robustness in segmenting complex indoor environments.

Deep models need large annotated sets, but expert annotations are costly. Handling multi-source data is also a problem: fusing UAV photogrammetry with terrestrial LiDAR (varying density, color vs intensity data) can confuse models.

In summary, while deep learning has enabled impressive results on structured parts of heritage scans, broad automation of full-building segmentation still requires more research, from better algorithms to richer annotated datasets.

III. Methods and approach

The research part on the subject was done by studying the basics of point clouds handling and treatment, and then by exploring the research papers done until now.

After that, two segmentation approaches were developed and implemented in the final program.

In both approaches, the point cloud is voxelized in order to reduce the computational cost of the data processing and to reduce noise. To clean the possibly generated noise in the segmentation, points are taken in small neighboring groups and their label is set to the most frequent one inside the group through majority voting. After performing the segmentation, the voxelized point cloud is restored to its original number of points.

A. Classic Machine Learning approach

1) The approach: This solution aims to train a classifier on segmented data to complete the segmentation.

As mentioned earlier, the point cloud to segment must have an already segmented portion; this part of the data can be manually labelled. Then, the already segmented points are used to train several Random Forest and Extra Trees classifiers, varying their parameters. The Extra Trees classifier belongs, like the Random Forest one, to the ensemble learning family (specifically, bagging of decision trees), but they differ in how they build their trees and inject randomness. With noisy data, Extra Trees can outperform Random Forest.

The best performing model, the one that achieves the highest IoU (Intersection over Union) metric, is finally selected and used to segment the points that are not yet classified.

2) Data pre-processing: The entire point cloud is first voxelized to reduce the computational weight of the task. Data usually has the following basic features: coordinates, RGB colors, and normals. If normals are not present, they are computed. Since RGB is a color format where the channels are semantically tied, the color features are converted into HSV, where each channel is independent. To better describe the points, the following additional features are computed: curvature, planarity, omnivariance, and verticality. These features are first scaled to improve convergence and prevent dominance of the features, PCA is applied to take the most discriminative directions.

3) Segmentation: Several combinations of parameters are used to train multiple Random Forest models, and the one with the highest IoU score is selected. Then, the model is used to segment the points that are not already classified. The segmentation relies on the point itself, without considering the neighboring points. At the end, the point cloud is de-voxelized, restoring the original number of points.

4) Results and considerations: This approach requires that a portion of the point cloud is already pre-segmented. The most reliable way to achieve this is through manual labeling with external software, although this step can be time-consuming. However, the method remains flexible, as it does not impose constraints on the number or type of classes to be segmented.

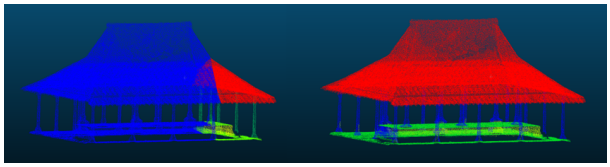

Fig. 1: Example with Pavillon

Also, if some elements are not segmented, the classifier will not be able to recognize them, and it will classify its points as the most similar ones present in the already segmented part.

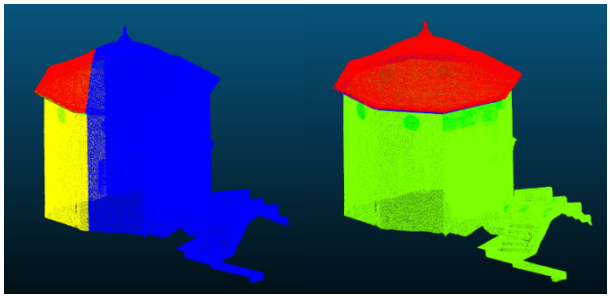

Fig. 2: Example with Chapel

As we can see in Figure 2, the stairs and the door cannot be segmented because, in the initial segmentation, there are no points of those elements.

B. Neural Network model approach

1) The approach: In this solution, the objective is to train a neural network model on the ArCH dataset and use it to segment new point clouds. Two model architectures were found suitable for this task: a modified PointNet++ and DGCNN.

Following [7], the `Other` class is excluded from model training, since its heterogeneous content could confuse the neural network.

2) Class imbalance: A significant problem of the dataset used is the class imbalance.

As shown in the Figure 3, the `wall` class is the most represented and `stairs`, `column`, `arch` are the least represented ones.

To address this issue, the weights of classes are computed to handle them effectively during data preprocessing and model training.

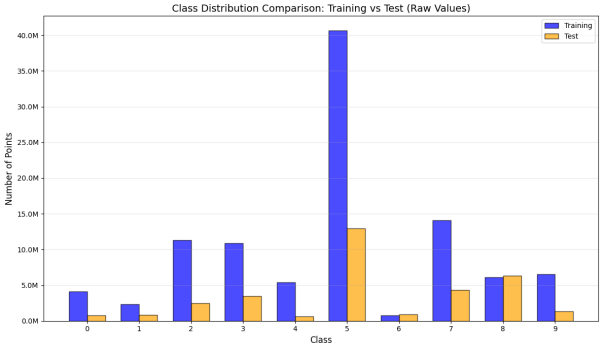

Fig. 3: Class distribution

Fig. 4: Class weighting techniques - Bar plot

Three weighting strategies were analyzed (Figure 4):

- `Inverse frequency`: Each class weight is inversely proportional to its frequency:

$$w_i = \frac{N}{K \cdot n_i},$$

where N is the total number of samples, K the number of classes, and n_i the count of class i. This ensures that the sum of weights across classes is normalized, but may overly penalize rare classes.

- `Power smoothed`: To reduce the sharp contrast between frequent and rare classes, the inverse-frequency weights are first normalized by their maximum, and then smoothed with an exponent $p \in (0, 1)$:

$$w_i = \left(\frac{w_i^{(\text{inv})}}{\max_j w_j^{(\text{inv})}} \right)^p.$$

Smaller values of p shrink the gap between rare and frequent classes, preventing instability while keeping relative differences.

- `Log smoothed`: Another approach (inspired from [9]) is to work with normalized frequencies $f_i = \frac{n_i}{N}$ and compute

$$w_i = \frac{1}{\log(\alpha + f_i)},$$

where $\alpha > 1$ is a smoothing constant. Weights are then normalized (e.g., dividing by $\max_j w_j$) for stability. This method mitigates the effect of very small frequencies and avoids excessively large weights for rare classes.

After this analysis, the log-smoothed technique was chosen and used as it is the most balanced.

3) Data pre-processing: In this case, the data pre-processing is more complex. First, all the point clouds in the training data are voxelized, and the centers of the voxels are taken as points. All the point clouds in the dataset present the same structure:

x y z r g b nx ny nz label

where:

- `x y z` are the point coordinates
- `r g b` are the color channels
- `nx ny nz` are the normals
- `label` is the class assigned to the point

Colors are converted into HSV format, as explained earlier, and a new column called `height` is added.

$$\text{height}_{\text{norm}} = \frac{z - \min(z)}{\max(z) - \min(z)}$$

This feature is a duplicate of the `z` column normalized in the whole point cloud; this column will remain unchanged in the next processing steps.

The final set of features is then:

x y z r g b nx ny nz height

Since neural network models require a fixed data size as input, all point clouds are split into 'cubes' of points (Figure 5). This is achieved by shifting a cube-shaped window, taking the points within it, and using them to create an entry of the modified dataset. Then, the window is shifted and the process is repeated until all points of the point cloud have been taken. In order not to penalize the points on the edges, the window is shifted with a small overlap relative to its previous position. So the offset is computed as

$$\text{offset} = \text{cube\_size} - \big(\text{cube\_size} \times \text{overlap}\big)$$

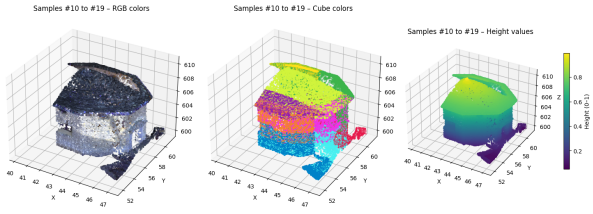

Fig. 5: Example of cubes splitted with height values (overlap = 0.10, cube_size = 5)

For consistency, each sample is constrained to 4096 points. Cubes containing more points are downsampled, whereas those with fewer points are augmented with padding until the fixed size is reached.

Padding points (with label -1) will be ignored during training.

In this step, when downsample is needed, points belonging to more frequent classes are reduced first to avoid penalizing

less frequent ones (the choice is made using the already computed class weights).

Two padding techniques were analyzed:

- `Interpolate padding`: This technique (Figure 6) adds new points between existing ones in a point cloud by interpolating their positions. It helps to create a denser point cloud, filling gaps and providing smoother representations of surfaces.

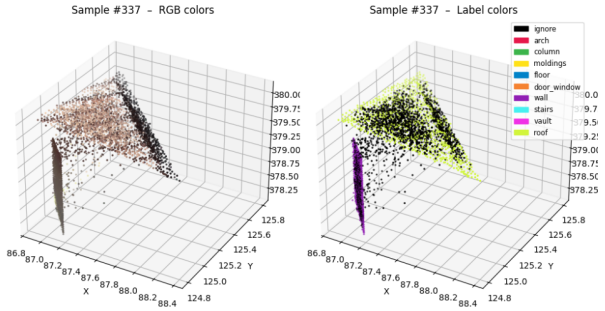

Fig. 6: Example of interpolate padding

- `Adaptive sampling`: Adaptive sampling (Figure 7) generates additional points by duplicating existing points, then applying small offsets to them to avoid exact overlap. This allows regions with insufficient points to reach a desired number of points while preserving the overall structure of the point cloud.

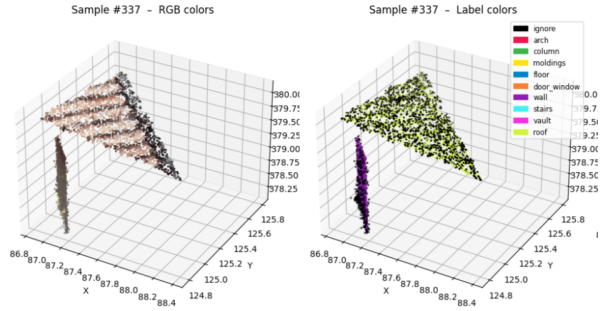

Fig. 7: Example of adaptive sampling

Interpolate padding is generally more effective for PointNet++ as it enhances local surface density, while adaptive sampling better suits DGCNN by preserving graph connectivity without introducing artificial structures.

4) Data augmentation: Applying extensive data augmentation to structured point clouds can produce unrealistic samples, potentially leading the model to incorrect predictions. Therefore, only minimal augmentation was applied.

Following [9], random rotations were applied only around the z-axis.

Additionally, a small random jitter was introduced to the point positions, creating slightly different samples each time without altering the overall geometry. A minor random jitter was also applied to the HSV color values.

5) Training: Model training, performance evaluation, and dataset analysis were conducted on Google Colab Pro and Kaggle platforms using Jupyter notebooks.

a) Loss Function: We implemented a version of the Jaccard loss (also known as Intersection-over-Union, IoU, loss) optimized for GPU efficiency using the `torch` library. Given model predictions of shape $[B, C, N]$ (batch, classes, points), the raw logits are first converted into probabilities by a softmax along the class dimension. The ground-truth labels are then transformed into one-hot encodings of the same shape. A variable called `ignore_index` allows the masking of an irrelevant label, such as the padding class.

The IoU score is obtained as

$$\text{IoU} = \frac{|A \cap B| + \varepsilon}{|A \cup B| + \varepsilon},$$

where ε is a small constant for numerical stability. The final loss is given by

$$\mathcal{L}_{\text{Jaccard}} = 1 - \frac{1}{C} \sum_{c=1}^{C} \text{IoU}_c.$$

The Jaccard loss is particularly powerful under class imbalance because it directly optimizes the IoU, which focuses on the overlap between predicted and true regions rather than raw pixel/point counts. This reduces the bias toward majority classes and ensures minority classes contribute meaningfully to the loss.

b) Optimizer: AdamW is a variant of the Adam optimizer that decouples weight decay from the gradient-based update. While standard Adam incorporates weight decay by adding an ℓ_2 penalty term to the gradients, AdamW applies weight decay directly to the parameter update. This adjustment decouples weight decay from the adaptive learning rates, resulting in more stable training and improved generalization. In summary, the key difference is that Adam applies weight decay through gradient regularization, whereas AdamW decouples it as a separate step.

AdamW, by decoupling weight decay from gradient updates, helps prevent the majority classes from dominating the optimization process and encourages better generalization to minority classes.

c) Scheduler: The ReduceLROnPlateau learning rate scheduler lowers the learning rate when a monitored validation metric (e.g., loss) stops improving for a specified number of epochs. Unlike step-based schedulers, which decrease the learning rate after fixed intervals, ReduceLROnPlateau adapts dynamically to the training process, ensuring that the learning rate is reduced only when the optimization appears to have plateaued. This allows the model to converge more effectively while avoiding unnecessarily small learning rates during periods of steady improvement.

ReduceLROnPlateau adaptively lowers the learning rate when progress stalls, allowing the optimizer to fine-tune decision boundaries for underrepresented classes instead of overshooting due to a high learning rate.

d) Metrics: During training, performance is monitored using IoU, F1-score and accuracy.

IoU is a more reliable metric than accuracy in the context of class imbalance. While accuracy can be dominated by majority classes (e.g., background points), IoU explicitly measures the overlap between predicted and ground-truth regions, penalizing both false positives and false negatives. This makes IoU more sensitive to minority classes, providing a fairer evaluation of segmentation performance in point clouds.

The F1-score is also useful in the context of class imbalance because it balances precision and recall, capturing how well the model identifies minority classes without being biased by the majority class. While IoU measures spatial overlap, F1 provides complementary insight into the trade-off between false positives and false negatives.

e) Models: Modified versions of PointNet++ and DGCNN waere implemented and trained.

Both introduce targeted enhancements to improve point cloud segmentation, particularly for complex and irregular structures such as heritage buildings.

Modified PointNet++: modifies the original architecture by explicitly including XYZ coordinates in each Set Abstraction (SA) layer while counting only non-coordinate features. Multi-Scale Grouping (MSG) is employed in the first two SA layers, while a single-scale approach is adopted in the third to avoid issues with coordinate handling. A Transformer layer is added after the last SA layer to capture richer contextual information. Feature Propagation layers incorporate residual connections to propagate features better. The segmentation head is enhanced with a residual block structure consisting of two convolutional layers separated by Batch Normalization, Leaky ReLU activation, and dropout. This design improves feature learning and stability.

Modified DGCNN: builds on standard DGCNN by replacing EdgeConv layers with residual EdgeConv blocks, preserving features across layers. Local features from each EdgeConv block are concatenated with a global feature vector to provide context for every point. The segmentation head is deeper, using sequential Conv-GroupNorm-LeakyReLU blocks with dropout, which enables more robust learning for fine-grained segmentation.

Overall, these modifications in both networks enhance their ability to capture detailed local structures and global shape information, resulting in improved performance.

f) Training with Curriculum Learning: To enhance model performance, the training procedure is divided into two distinct stages. The first stage employs a Curriculum Learning strategy, where the model is trained on a reduced subset of the data that contains mostly samples with underrepresented or more challenging classes.

In this case, the chosen classes are *arch*, *column*, *door_window*, and *stairs*. Emphasizing these classes helps the model learn their features more effectively before training on the full dataset.

This stage uses a slightly higher learning rate and runs for a few epochs, allowing the model to capture essential patterns for these difficult classes quickly. In the second stage, training continues on the full dataset with a lower learning rate, enabling the model to refine its representations and achieve more stable convergence across all classes.

6) Inference: To perform inference on a new point cloud, the data must be pre-processed in the same way as done to the dataset during model training. It is essential to track the points transformations to reconstruct the original point cloud with the corresponding segmentation labels. Specifically, each point is assigned a unique index, and voxelization is performed while preserving the indices of the points contained within each voxel. After segmenting the voxelized sample, the original points are restored by assigning them the label of the corresponding voxel center.

This procedure is applied both to the full point cloud initially and during cube splitting when downsampling is required.

Additionally, HSV colors must be converted back to RGB format after the segmentation.

IV. Results

The following results belong to the effective training phase following curriculum learning.

ModPointNet++

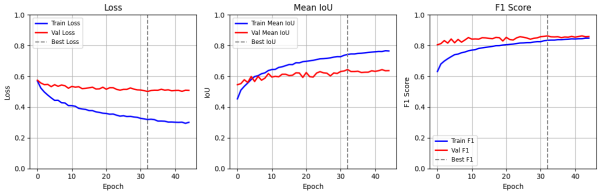

Fig. 8: Metrics across epochs

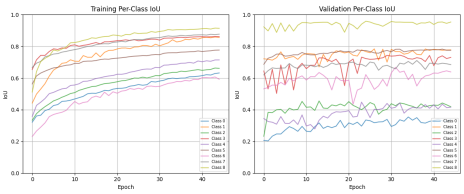

Fig. 9: Per-class IoU

The analysis of the training and validation per-class IoU curves (Figure 9) shows that the model performs unevenly across different classes. During training, all classes show a steady improvement, with some reaching values above 0.8, while others remain consistently lower, converging around 0.6. In validation, the same classes achieve high IoU values, while the weaker ones continue to be problematic. Furthermore, the validation curves reveal significant fluctuations for certain classes, indicating instability likely caused by class imbalance, limited training samples, or high intra-class variability. There is no evidence of severe overfitting, as the training and validation behaviors remain broadly aligned.

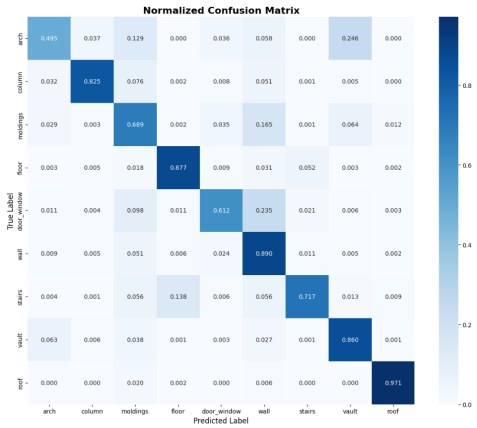

Fig. 10: Confusion Matrix Normalized

The confusion matrix shown in Figure 10 is normalized to make it clearer where the model is making mispredictions. As we can see, the model is predicting 25% of `arch` points as `vault` and 13% as `moldings`, 24% of `door_window` points as `wall` and 14% of `stairs` points as `floor`.

As explained in [7], the low performance values are mainly due to geometric similarities between certain elements, the heterogeneous nature of some classes, and limitations in the data acquisition process. In particular, some categories share very close geometric features, leading to confusion between them, while others group highly diverse objects, making them harder to classify consistently.

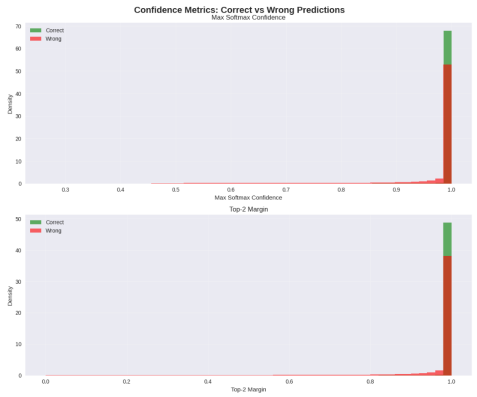

Fig. 11: Confidence and Top-2 Margin

In Figure 11 further analysis on model predictions are presented. The *confidence* (max softmax confidence) corresponds to the probability assigned to the most likely class. A higher confidence indicates that the model is more certain about its prediction. The *top-2 margin* is defined as the difference between the highest and the second-highest predicted probabilities. A large margin reflects that the model strongly favors one class over the others, whereas a small margin indicates that the prediction is more uncertain or ambiguous.

In this case, the model demonstrated high confidence for the correct predictions, while the misclassifications were generally associated with greater uncertainty.

Fig. 12: Confused Pairs

In Figure 12 the 10 most confused pairs of classes are analysed. The uncertain samples are computed by selecting those where the probability margin between the top-1 and top-2 predictions falls below a predefined threshold (0.5). For these uncertain predictions, we count the occurrences of each class pair, sorting the pairs by frequency to determine the top 10 most confused pairs.

This analysis highlights which class pairs the model struggles to distinguish, revealing specific weaknesses.

ModDGCNN

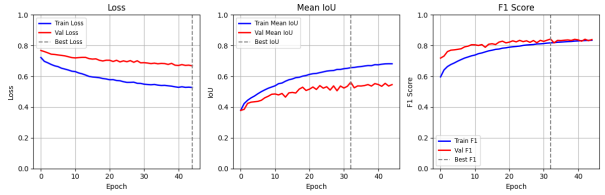

Fig. 13: Metrics across epochs

The DGCNN model training metrics (Figure 13) show progress similar to the PointNet++ ones.

Following the initial training phase, as no clear signs of overfitting were observed and the performance continued to show slight improvements, the process was extended for additional epochs to refine the model further.

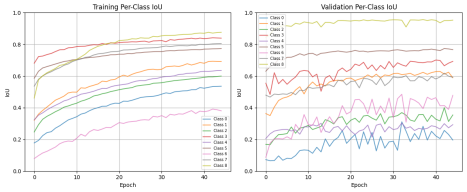

Fig. 14: Per-class IoU

The per-class IoU performance (Figure 14) in this case turned out to be slightly lower compared to that of the PointNet++ model.

By examining both plots, we can see that the learning curves on the training set remain relatively smooth and stable, while those on the validation set exhibit significant fluctuations. This instability is particularly evident in classes that are underrepresented in the dataset, since even small variations can heavily impact the IoU metric for those categories.

Moreover, part of this instability can also be explained by the limited size of the validation set itself, which reduces the statistical robustness of the results and amplifies noise in the evaluation (Figure 3).

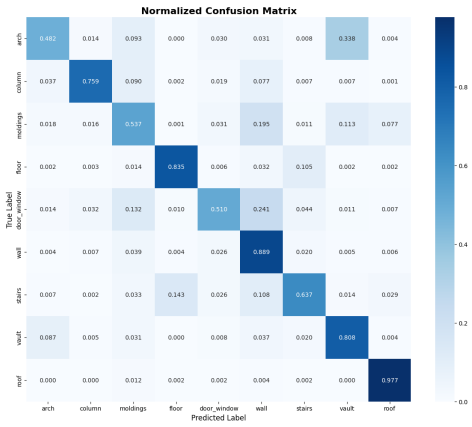

Fig. 15: Confusion Matrix Normalized

The confusion matrix (Figure 15) shows similar results to those of the PointNet++ one, with slightly worse results.

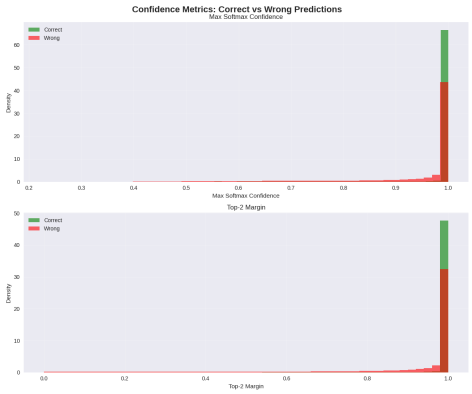

Fig. 16: Confidence and Top-2 Margin

The plot in Figure 16, when compared to the one in PointNet++, does not show major differences overall; however, the model appears less confident in its predictions, especially for those that are incorrect.

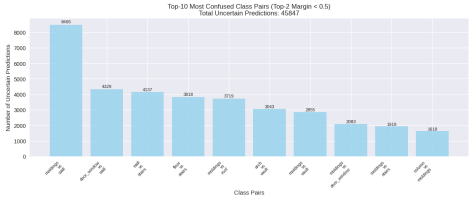

Fig. 17: Confused Pairs

Both models share similar top confusions (moldings-wall, door_window-wall, floor-stairs, moldings-roof), but the absolute numbers are consistently higher in DGCNN (Figure 17), indicating more uncertainty.

A. *Final models results*

In the following section, the final evaluation metrics of the models, along with some inference example,s are presented.

Model	Loss	IoU	F1 Score
PointNet++	52.84	60.35	85.69
ModPointNet++	50.14	65.05	88.03
DGCNN	53.84	56.92	83.74
ModDGCNN	62.21	58.80	85.35

TABLE I: Comparison of evaluation metrics (%)

Table I presents a comparison of the metrics for the final trained models. For reference, basic versions of the models are also included to illustrate the improvements achieved through the model modifications.

Model	arch	column	moldings	floor	door/window
PointNet++	36.61	68.95	37.51	74.79	25.27
ModPointNet++	33.72	76.83	44.64	75.54	45.33
DGCNN	31.13	67.78	34.96	62.54	37.54
ModDGCNN	33.20	67.70	37.41	72.36	35.54

(a) Classes arch–door_window

Model	wall	stairs	vault	roof
PointNet++	75.92	61.42	67.38	95.28
ModPointNet++	79.35	63.93	69.61	96.54
DGCNN	78.26	48.66	60.54	90.87
ModDGCNN	77.83	49.64	59.89	95.61

(b) Classes wall–roof

TABLE II: Per-class IoU comparison (%)

Table II presents the per-class IoU metrics for each model. Overall, the modified version of PointNet++ achieved the strongest performance among the tested models.

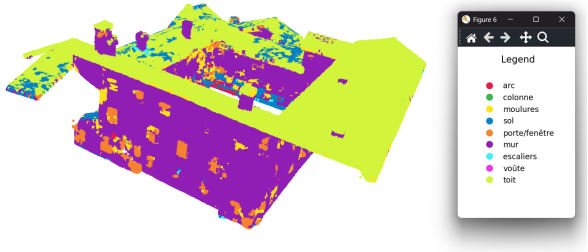

Fig. 18: Novalesa Cloister segmented with ModPointNet++

Predicted point clouds: Although PointNet++ outperforms DGCNN in overall quantitative metrics, visual assessment indicates that DGCNN (Figure 19) produces more accurate segmentation of columns and doors/windows than PointNet++ (Figure 18) in this point cloud example. However, DGCNN's results exhibit higher variability, whereas PointNet++ produces more consistent segmentations. However, this outcome is specific to this point cloud, and results may vary with different data.

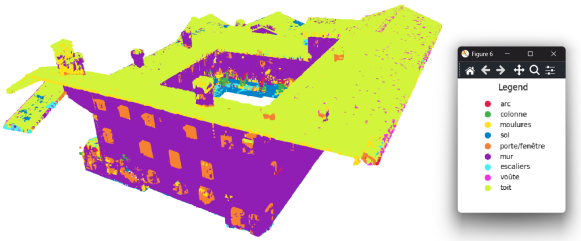

Fig. 19: Novalesa Cloister segmented with ModDGCNN

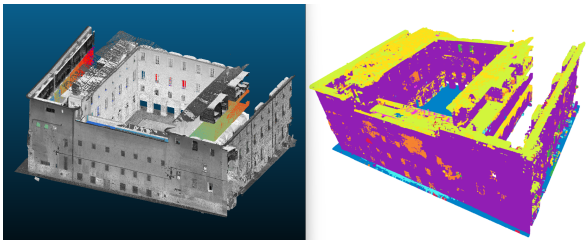

Fig. 20: Lycée Sainte Catherine segmented with ModDGCNN

As illustrated in Figure 20, since colors were incorporated during model training, the absence of color information in a point cloud can lead to worse results.

V. Discussion

The results indicate that the models learned to recognize certain patterns within the point cloud; however, segmentation lacks precision. This limitation is attributed in part to the variability in data quality and also to data scarcity for certain classes.

Fig. 21: Detail of Novalesa Cloister point cloud

As illustrated in Figure 21, the sampled point cloud exhibits uneven quality. Some areas are overexposed with very bright colors and the point sampling contains a significant amount of noise, which badly affects the segmentation performance.

VI. Conclusion

After applying pre-processing and analyzing different model architectures, the modified PointNet++ shows a slight performance improvement. However, the results remain insufficient for reliable predictions on new point clouds, as certain classes are still poorly segmented. However, this provides a solid foundation for further development.

Next Steps

To further strengthen the models, the most important requirement is a richer and more diverse annotated dataset. A larger dataset would allow the model to learn more robust representations of the data and generalize more effectively.

Another promising direction is the incorporation of geometric consistency rules to correct potential misclassifications. For example:

- The points of the doors and windows should always be surrounded by the points of the wall.
- The wall points should always be located above the floor points.
- Stair points should not appear among the roof points.

Enforcing such constraints after prediction can significantly improve output quality.

This approach can be extended by grouping segmented points into object-level entities and constructing a relational graph of their spatial configurations. Using these relationships, inconsistencies in segmentation can be detected and corrected, further improving the reliability of the model.

References

[1] F. Matrone, A. Lingua, R. Pierdicca, E. S. Malinverni, M. Paolanti, E. Grilli, F. Remondino, A. Murtiyoso, and T. Landes, "A benchmark for large-scale heritage point cloud semantic segmentation," *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, vol. XLIII-B2-2020, pp. 1419–1426, 2020, doi: 10.5194/isprs-archives-XLIII-B2-2020-1419-2020.

[2] Y. Yang, X. Wu, T. He, H. Zhao, and X. Liu, "SAM3D: Segment Anything in 3D Scenes," *arXiv preprint arXiv:2306.03908*, 2023.

[3] E. Grilli, "Automatic classification of architectural and archaeological 3D data," Ph.D. dissertation, Alma Mater Studiorum, Universitá di Bologna, Bologna, Italy, 2020. [Online]. Available: https://amsdottorato.unibo.it/id/eprint/9347/1/Grilli_Tesi_PhD_final.pdf

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," arXiv preprint arXiv:1612.00593, 2016.

[5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," arXiv preprint arXiv:1706.02413, 2017.

[6] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," *arXiv preprint arXiv:1801.07829*, 2018.

[7] R. Pierdicca, M. Paolanti, F. Matrone, M. Martini, C. Morbidoni, E. S. Malinverni, E. Frontoni, and A. Lingua, "Point cloud semantic segmentation using a deep learning framework for cultural heritage," *Remote Sensing*, vol. 12, no. 6, p. 1005, Mar. 2020, doi: 10.3390/rs12061005.

[8] H. Yue, Q. Wang, L. Huang, and M. Zhang, "Enhancing point cloud semantic segmentation of building interiors through diffusion-based scene-level synthesis," Automation in Construction, vol. 178, p. 106390, 2025, doi: 10.1016/j.autcon.2025.106390.

[9] D. Griffiths and J. Boehm, "Weighted point cloud augmentation for neural network training data class-imbalance," *arXiv preprint arXiv:1904.04094*, 2019. [Online]. Available: https://arxiv.org/pdf/1904.04094