UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LEONARDO HÖLTZ DE OLIVEIRA

Nubilum: A Captum's Extension Library for Semantic Segmentation Models Focused on Point Cloud Data

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. João Luiz Dihl Comba
Coadvisor: B.Sc Pedro Sidra de Freitas

Porto Alegre
September 2023

“I’m fine. Two plus two is...ten...
in base four. I’m fine!”
— GLaDOS

ACKNOWLEDGEMENTS

I would like to thank my family, specially my mother, Cristina, for the support and love I received during my life and journey at UFRGS. Only we know the opportunities, knowledge, comfort and happiness that entering a public university have brought to us.

I would also like to thank the friends I made since the beginning of graduation, especially the course veterans who helped me (and that I also helped) since before my first day at UFRGS, and the colleagues who started in the same semester as me. You were the people who made my experience at the university more fun and joyful.

My gratitude to UFRGS and Instituto de Informática (INF) for the excellent environment and education, and to all the professors and teachers that I had in the last decade. You have played a significant role in shaping my education and personal growth.

Finally I would like to thank to my advisors João Comba and Pedro Freitas for their help in this work. Their inspirations, insights, knowledge, suggestions and help in the last months have contributed immensely to the successful completion of this work.

ABSTRACT

With the increasing development of artificial intelligence (AI) applications in the last decade, ensuring trust in AI systems decisions created a demand for interpretability and explanations for black-box model predictions. Semantic segmentation over point cloud data is one of the areas with a significant demand for interpretability, thanks to visionary systems being developed, such as autonomous vehicles and robot navigation. To interpret semantic segmentation models for point cloud data, we propose a new Python's library called Nubilum, an extension of Captum, a model interpretability library built on PyTorch. We extended Captum's generic implementations of post-hoc attribution methods, developed model wrappers to be used as forward functions by these methods, and techniques to visualize the point cloud, its segmentation, and explanations for any semantic segmentation model. We tested our library by executing an analysis over the SoftGroup model and S3DIS dataset to visualize attributes and interpret the decisions made by SoftGroup.

Keywords: Artificial Intelligence (AI). Explainable Artificial Intelligence (XAI). Point Cloud. Semantic Segmentation.

Nubilum: Uma biblioteca de extensão do Captum para modelos de segmentação semântica focados em dados de nuvem de pontos

RESUMO

Com o aumento do desenvolvimento de aplicações de inteligência artificial (IA) na última década, garantir confiança sobre as decisões de sistemas de IA criou uma demanda para interpretabilidade e explicações para predições de modelos caixa preta. A segmentação semântica sobre dados em nuvem de pontos é uma das áreas com demanda significante para interpretabilidade, graças a sistemas visionários sendo desenvolvidos, como veículos autônomos e navegação de robôs. Parar interpretarmos modelos de segmentação semântica para nuvem de pontos, nós propomos uma nova biblioteca em Python chamada Nubilum, uma extensão do Captum, uma biblioteca de interpretabilidade de modelos construída sobre o PyTorch. Nós extendemos as implementações genéricas do Captum de métodos de atribuição post-hoc, desenvolvemos wrappers de modelos para serem usados como funções forward por estes métodos, e técnicas de visualização para visualizar nuvem de pontos, suas segmentações e explicações para qualquer modelo de segmentação semântica. Nós testamos nossa biblioteca com uma análise sobre o modelo SoftGroup e o conjunto de dados S3DIS, para visualizar atributos e interpretar as decisões feitas pelo SoftGroup.

Palavras-chave: Inteligência Artificial (IA). Inteligência Artificial Explicável. Nuvem de Pontos. Segmentação Semântica.

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AV	Autonomous Vehicle
CNN	Convolutional Neural Network
DNN	Deep Neural Network
GT	Ground Truth
IG	Integrated Gradients
LiDAR	Light Detection And Ranging
ML	Machine Learning
S3DIS	Stanford 3D Indoor Scene Dataset
XAI	Explainable Artificial Intelligence

CONTENTS

1 INTRODUCTION

1.1 Machine Learning models and its interpretability

In the last decade, our society has witnessed an unprecedented rise in the usage of artificial intelligence (AI). From recommendation systems to autonomous vehicles (AVs), AI is redefining our ways to learn, interact, and make decisions. With the capability to extract knowledge and patterns in data, machine learning (ML) models have great performance in many tasks. Their development is innovating in areas like image and speech recognition, natural language processing, healthcare and medicine, and many more.

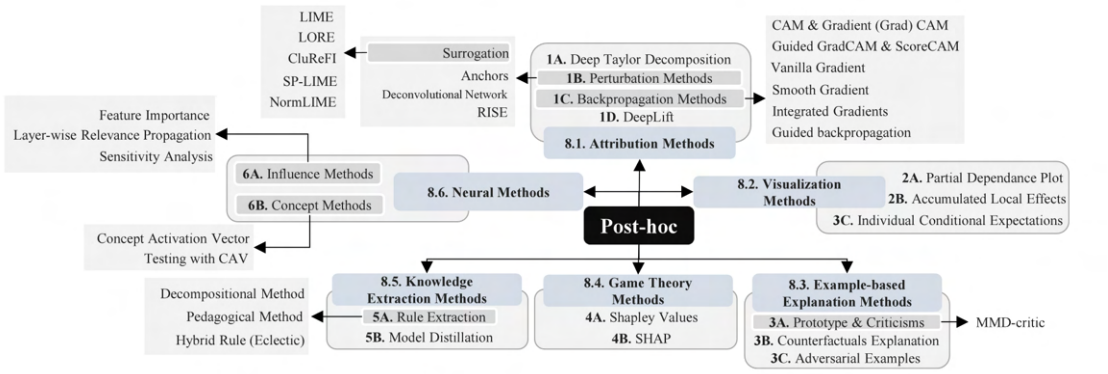
However, the black-box nature of some models, especially the ones developed in deep neural networks (DNNs) techniques, has raised concerns about these model's decisions. The lack of interpretability and transparency impacts the model's trust, and the AI community aims to find ways to create trustworthy AI. To achieve this, XAI comes with techniques to create ML interpretable models without interfering with the model's performance and to create explanations of the model's decisions for end-users.

> Interpretability enables developers to delve into the model's decision-making process, boosting their confidence in understanding where the model gets its results [...]. Explainability provides insight into the DNN's decision to the end-user to build trust that the AI is making correct and non-biased decisions based on facts. (ALI et al., 2023, p.2).

XAI is an extensive topic, and with more methods being developed, an organization of these methods and types of explainability is necessary. Ali et al. (2023) divides the purposes behind an explanation into four axes: (1) data explainability, (2) model explainability, (3) post-hoc explainability, and (4) assessment of explanations. We will focus on post-hoc explainability, which explains a model's decision after its prediction. Only in the post-hoc explainability axis we have a range of methods. Figure 1.1 contains a division of the post-hoc methods (The numbers in the divisions come from the original work of Ali et al. (2023).).

In this work, we analyze a pre-trained DNN model's decisions using an interpretability library containing most of the post-hoc methods listed in Figure 1.1. We initially chose two attribution methods based on backpropagation, the vanilla gradient, and the integrated gradients, described in Section 2.1.

Figure 1.1: An overview of post-hoc explainability methods.



Source: (ALI et al., 2023)

Although XAI has become a popular research subject within the AI field in recent years, its usage in non-common tasks, other than simple regression and classification, and data types, other than images and texts, is still an ongoing challenge (section 1.3 provides very recent works). In the next section, we will address a specific task that we will explore in this work: the semantic segmentation task for 3D point cloud data, the dataset used for the task, and the model designed for the task that we used to retrieve interpretable information.

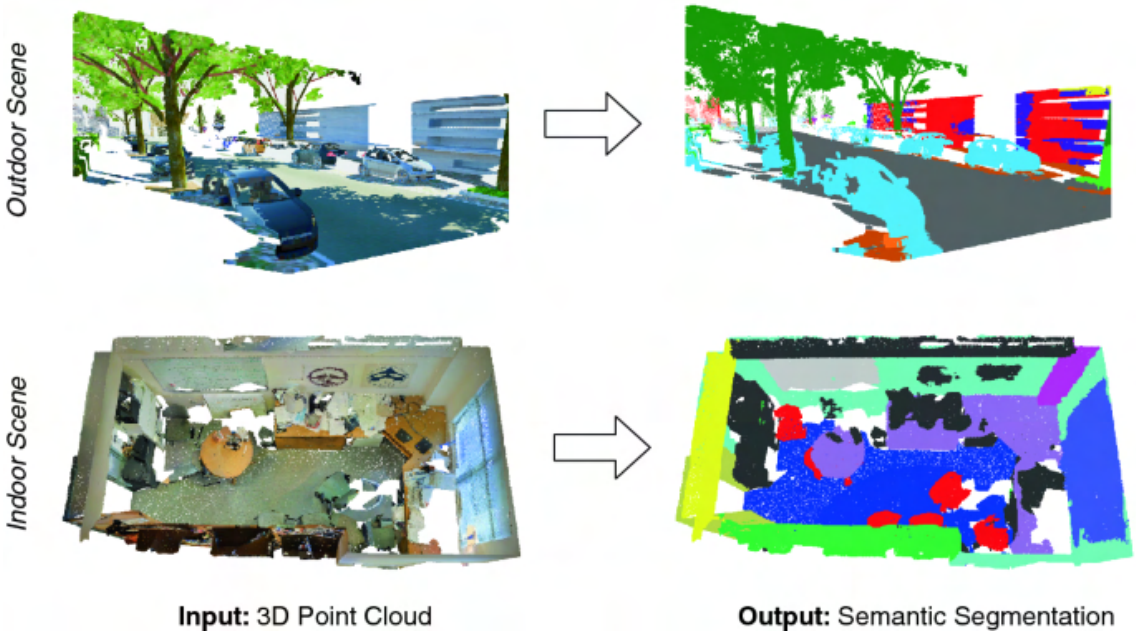
1.2 3D Point Cloud Semantic Segmentation task

The 3D acquisition technologies have rapidly improved their performance, and 3D sensor products (e.g., light detection and ranging (LiDAR) laser scanners and RGB-D cameras) have become more available and affordable (GUO et al., 2021). These sensors allow us to collect data from an environment, such as small rooms or open areas, and organize it in a point cloud structure. As Guo et al. (2021) lays down, the point cloud representation best represents a 3D space for autonomous driving and robot navigation applications. It preserves the original geometric information in 3D space without discretization.

Nonetheless, the sensor cannot acquire semantic information about the 3D scene. For example, when an autonomous car is moving in the street, identifying each object in the scene, particularly stop signs, pedestrians, and other cars, is essential and can be life-saving. Therefore, DNN models have been developed to achieve this objective. As

Hafiz and Bhat (2020) define — adapting from image's pixels to point cloud's points — the semantic segmentation task objective is to predict labels for each point. These labels indicate the class object or area that surrounds each point. There is also an extension for this task called instance segmentation, where we also have labels that indicate the instance of an object. For example, a point can be labeled as part of a car, but it is necessary to specify which car it belongs to. Figure 1.2 shows a visualization of two point clouds and an example of their semantic predictions.

Figure 1.2: Example of point clouds and their semantic prediction.



Source: (ENGELMANN et al., 2019)

The input of the 3D point cloud semantic segmentation task consists of scene, containing N points. Each point have a 3D coordinate, divided in x, y and z, and a color, divided in the three channels red, green and blue. The output of this task consists of a $N \times M$ matrix, where we have M scores, one for each possible class. for N points. Usually, an $argmax$ function can be executed over the output to obtain a vector with size N that contains the indices of the predicted class for each point. Instance segmentation task outputs also contain an extra vector with size N that contains the indices of the predicted object instances.

In this work, we use the SoftGroup model, proposed by Vu et al. (2022), and the Stanford 3D Indoor Scene Dataset (S3DIS), created by Armeni et al. (2017), to develop and test our explainability library. The dataset and model are described in more detail in

Sections 3.1 and 3.2.

1.3 Point Cloud Semantic Segmentation Explicability

The main objective of this work is to assist semantic segmentation and instance segmentation models of point cloud data developers with a library that uses Captum's library (KOKHLIKYAN et al., 2020) post-hoc interpretability methods and uses plot tools to explain these methods results. Previous works have already discussed the idea of explaining semantic segmentations of point cloud data. Heide et al. (2021) proposed example-based explanations for the semantic segmentation of 3D data with prototypes and criticism. Given a 3D point set to be explained, the prototypes are the most similar 3D point sets to the original set, and the criticisms are the most dissimilar point sets. If we use the categorization proposed by Ali et al. (2023), this method would be categorized in the data explainability axis. Schwegler, Müller and Reiterer (2023) proposed an integrated gradients method over a dataset with outdoor scenes, but it does not use any existing framework or library to compute the attributions. Lee, Yang and Han (2023) propose an epistemic uncertainty reduction by graphical information gain-based attention network. It measures the relative entropy between a target point and its neighborhood to obtain information. Lavado et al. (2023) propose a white-box model with intrinsic geometric interpretability using group equivariant non-expansive operators. The most similar work to ours is the one proposed by Matrone et al. (2022). It is a framework to understand 3D point cloud features, with an extension of the saliency maps method, dedicated to point cloud data and with visual methods for explanation. However, It is executed only in the point cloud classification tasks (unique objects).

Unlike the works cited, we made an extension of a well-known model-agnostic interpretability library with post-hoc interpretability methods dedicated to semantic segmentation tasks, with three different model wrappers and visualization tools already configured with methods to analyze point clouds, their segmentation results and explanations.

1.4 Contributions

The contributions of our work are:

- An extension of Captum's post-hoc attribution methods dedicated to point cloud

data.

- Model wrappers that reduce semantic segmentation and instance segmentation outputs to similar classification task outputs that can be used in the attribution methods to obtain attribute values of scene features with different magnitudes.
- Visualization methods to visualize 3D point cloud scenes, 3D segmentation of scenes, and explanations of attributes computed from scene features.
- A baseline approach for point cloud scenes.

1.5 Document Structure

This work is structured as follows. Chapter 2 introduces our library and its functionalities. It describes our attribute methods, model wrappers developed, and plot tools used to explain the attributes. Chapter 3 describes an explainability experiment conducted using the methods developed, the model and dataset used. Chapter 4 contains this work's concluding remarks and future directions. The appendix A contains the library's documentation.

2 NUBILUM LIBRARY

To retrieve explainable visualizations from point cloud instance segmentation models, we propose *Nubilum* ("cloudy" in Latin), a Python open-source library extension for Captum. It uses PyTorch (PASZKE et al., 2019) as its main basis for the model and interpretability methods since it uses the torch tensor as its primary data structure. Nubilum utilizes Captum's post-hoc interpretability methods to retrieve attribute values for each feature passed as input to a model. Usually, as in this work's model analysis covered in chapter 3, the main features to use as input to a model for an instance or semantic segmentation type of task with point cloud data are only the points coordinate and the points colors. Therefore, we need only these two features to use all of Nubilum's functionalities. Other features can be fabricated through data augmentation techniques, but they may vary from model to model. Therefore, our library does not offer visualization support for these other features. Captum and its functionalities are described in section 2.1.

Post-hoc interpretability methods are initially intended to interpret the model's decisions of classification tasks. Depending on the algorithm used, these methods require a target class to analyze which part of the features contributed to the model's decision and give attribute values for each part. However, we can adapt the nature of these methods to work for semantic and instance segmentation tasks. A model's output for this type of task consists of class scores for each point. If we modify a model's output to make it similar to a classification model's outputs, where we have only one group of scores, we can calculate attributes, for instance, segmentation models. Considering that, Nubilum offers ready-to-use model output modifications — model wrappers — to assist developers in achieving specific interpretations. Section 2.2 shows more details on each model wrapper.

Nubilum also utilizes Python's 3D plot libraries to assist in visualizing any attributes obtained by its methods. Currently, we use two libraries, the Plotly Graphing Library[1], and K3D-Jupyter visualization package[2]. Both library usage and its pros and cons for this work are described in section 2.3.

Since it is not the focus of this work, we will refrain from describing Nubilum's design pattern, code standards, or similar. Still, in the Appendix A, we present a complete documentation of Nubilum. The library implementation is also available on Github[3]. It's important to notice that Nubilum was tested only with architectures based on CNN, there-

[1] https://plotly.com/python/
[2] https://k3d-jupyter.org/
[3] https://github.com/LeonardoHoltz/Nubilum

fore we do not guarantee its works on networks based on Graph Convolutional Network.

2.1 Captum library

As said previously, Nubilum utilizes Captum[4], an open-source model interpretability library for PyTorch, as its primary basis. Captum was created from a demand for sufficient support of attribute algorithms frameworks for PyTorch models when state-of-the-art DNNs were being designed without considering interpretability. This library was chosen because Nubilum benefits from Captum's generic implementations of gradient and perturbation-based attribution algorithms, as well as its multiple modalities of inputs, extensibility, and ease of use (KOKHLIKYAN et al., 2020).

Kokhlikyan et al. (2020) divides Captum's attribution algorithms into three categories — primary attribution, layer attribution, and neuron attribution. The primary attribution category attributes values of the model's output to the input, evaluating the contribution of the input's features to the model's decision. The Layer attribution category attributes the model's output to a specific internal layer, evaluating the layer's contribution to the model's decision. Finally, the neural attribution category attributes internal neurons' output to the input, evaluating the contribution of the input's features to the neuron's output. Figure 2.1 exemplifies the three categories with code snippets.

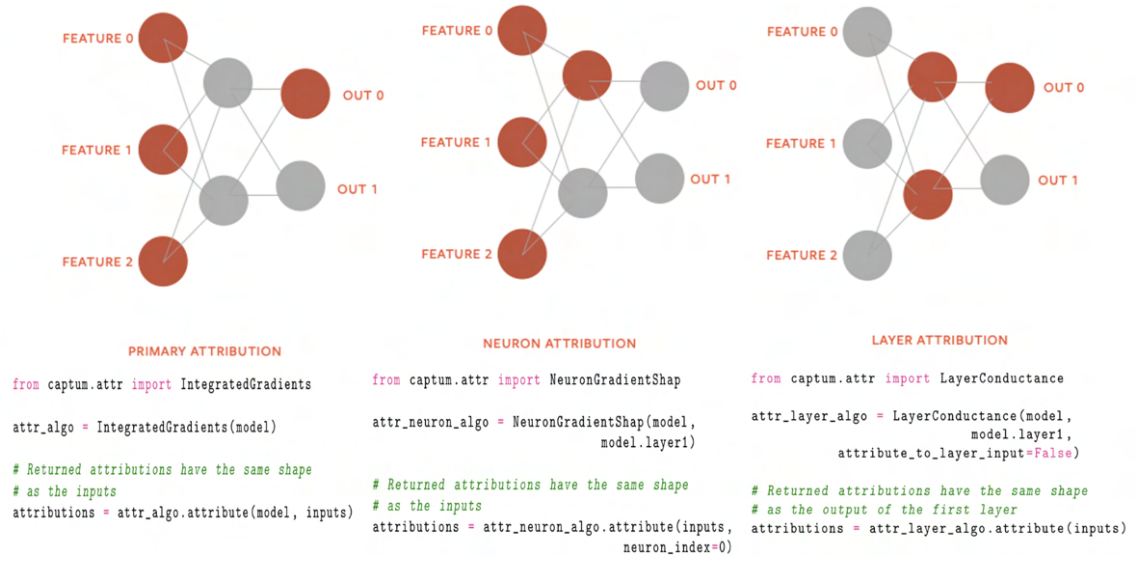
Nubilum's current version, described in this work, only adapts the attribution algorithms from the primary attribution group. Therefore, this work will not discuss any of the algorithms that belong to the other two categories. The following sections describe the attribution algorithms used and adapted for Nubilum.

2.1.1 Saliency

Saliency is Captum's baseline approach for computing input attribution. The implementation is an adaptation of the work proposed by Simonyan, Vedaldi and Zisserman (2014). In the original paper, the authors describe that a Convolutional Neural Network (CNN) with a class score function $S_c(x)$, with an image input x, is a non-linear function that can be approximated by a first-order Taylor expansion to find derivatives, or gradients, for the network weights w. The saliency map is an aggregation of these gradients.

[4] https://captum.ai/

Figure 2.1: An overview of all the three attribution categories.



Source: (KOKHLIKYAN et al., 2020)

The derivative's magnitude of the weight w indicates how much a modification in the image's pixel associated with w can change the class score (SIMONYAN; VEDALDI; ZISSERMAN, 2014). The saliency map has the gradients for all possible classes. Thus, if we want to analyze a specific class c, It is essential to extract only the gradient for c. Figure 2.2 shows an example of a saliency map for a classification from the model proposed by the authors.

Fortunately, PyTorch models already save the gradients through its backpropagation process, if configured correctly. To calculate the saliency attributes, Captum computes the gradients of the model with respect to an input for a specific class of interest. As said before, Captum implementation is model-agnostic, which means that, although the original paper has described the process to an image classification CNN, this idea works with any type of model that can offer a class score function. Section 3.4.2 presents some of the results from this algorithm for our task. Since this algorithm is a baseline for computing input attribution, it is the most rudimentary algorithm, with the most basic results, and it is used as the basis for the other most complex algorithms.

Figure 2.2: Example of class saliency maps to classifications of a monkey and an ox.

Source: (SIMONYAN; VEDALDI; ZISSERMAN, 2014)

2.1.2 Integrated Gradients

Sundararajan, Taly and Yan (2017) proposed a method called *Integrated Gradients* (IG), where in addition to the input of a model, we use a baseline input. This method can be considered as an extension of the saliency method since it also uses the gradient values according to a certain input. The difference for the IG method is the constant change in the input to acquire different values of gradients and sum them. Let $\boldsymbol{x} \in R^n$ be the DNN input and $\boldsymbol{x'} \in R^n$ be the baseline input, and l be the straight line path, in R^n, between $\boldsymbol{x'}$ and $\boldsymbol{x}$. The IG is an accumulation of the gradients of all the points from $\boldsymbol{x'}$ to $\boldsymbol{x}$ in l. More specifically, the IG for a specific feature from the input $\boldsymbol{x}$ can be defined as

$$IntegratedGrads_i(\boldsymbol{x}) := (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha(\boldsymbol{x} - \boldsymbol{x'}))}{\partial x_i} d\alpha \qquad (2.1)$$

with α defining how close this point in l is to the baseline $\boldsymbol{x'}$ or to the input $\boldsymbol{x}$. An α close to 0 means an input close to the baseline, and α values close to 1 mean an input close to the original input.

Computationally, we can not calculate the gradients for all the points in l to achieve an exact value, but we can approximate it. As an implementation of this idea, the authors propose a Riemann approximation of the integral. A Riemann approximation, or Riemann sum, as Anton, Bivens and Davis (2012) defines, is an approximation of an integral by a finite number of sums where the partitions of the integrated integral are sufficiently small. For our case, these partitions can be defined by an extra parameter m to define the number of steps (points in the straight line l). The sum is done to the gradients at points occurring at sufficiently small intervals along the straight line l (SUNDARARAJAN; TALY; YAN, 2017). Despite Sundararajan, Taly and Yan (2017) have proposed the use of a Riemann sum, the integral approximation can be achieved by other methods as well.

$$IntegratedGrads_i^{approx}(\boldsymbol{x}) := (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m}(\boldsymbol{x} - \boldsymbol{x'}))}{\partial x_i} \times \frac{1}{m} \tag{2.2}$$

The quality of the results can change depending on (1) how good the baseline is, and (2) how many steps are used for the approximation. The original authors found that somewhere between 20 and 300 is sufficient, in image classification models, to achieve a relative error of 5%. Captum defines the number of steps as a default value of 50, that can be modified by the user. Schwegler, Müller and Reiterer (2023) already developed a baseline for point cloud data using a cuboid with the points randomly distributed through the volume with random colors. However, we decided to create our own baseline, adapting some of Schwegler, Müller and Reiterer (2023) ideas, but using a methodology based on how baselines are structured in images.

2.1.2.1 Proposed point cloud baseline

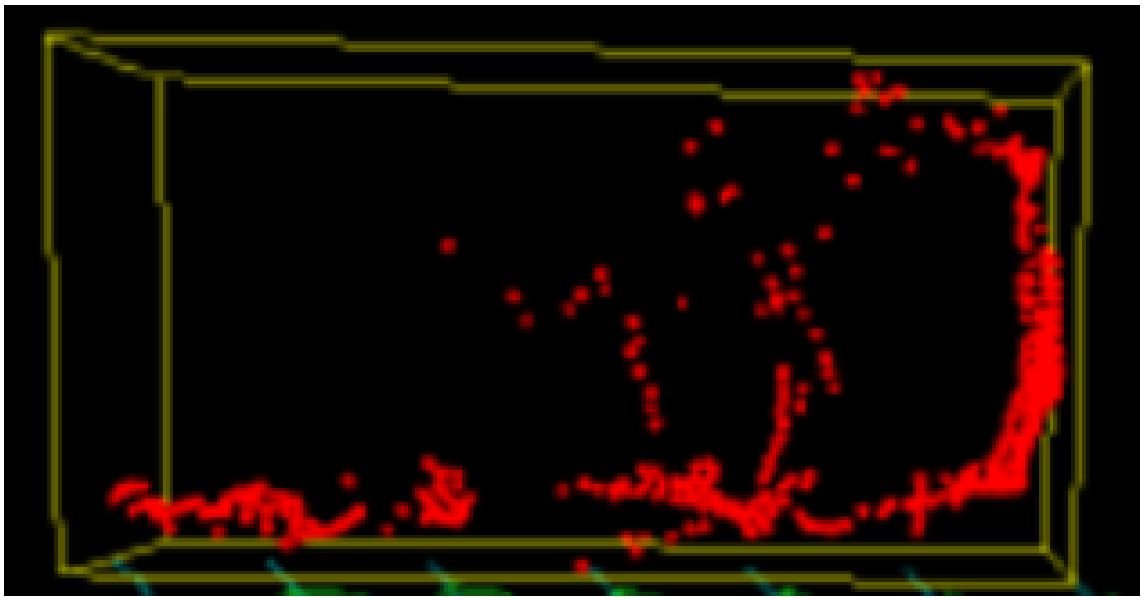
Because of the generic nature of Captum, the default baseline proposed by the library is a default zero scalar for each input tensor. This baseline, despite being simple, does not represent the structure of our domain, potentially leading to misleading or inaccurate attributions. One example is the coordinates of a point cloud. A scene with shifted coordinates to a certain direction in the 3D space would impact the factor $(x_i - x_i')$ if the default baseline is used. As a consequence, based on baselines used in image models, we propose that Nubilum has its own baseline. Sundararajan, Taly and Yan (2017) recommend the usage of black images for image models, and we propose a similar strategy.

Since we are dealing with two types of features — colors and coordinates — we will discuss them separately.

To begin with, inputs of a point cloud are not well-behaved and can be different in the number of points. Therefore we always need to define a new baseline if we want to interpret a specific input. The color feature of a point cloud is straightforward similar to a color image. Each point has a value for each RGB channel, as well as the pixels of an image. Considering that, we can utilize the same strategy of black images by using the absence of color in all points.

The main difference is when we analyze the coordinates features since images do not have this type of information. What we can use as an inspiration is the information on the height and width of an image. Image data geometrically corresponds to a uniform grid of pixels, with the dimensions of the image determining the size of this grid. Therefore, we can also create a grid of points as our baseline, but instead of a grid area, we create a grid volume. We do not have information on height, width, and length at first, but we can obtain it by checking the bounding box surrounding the point cloud, obtaining intervals in the x, y, and z coordinates and consequently the values of height, width, and length. Figure 2.3 shows an example of a point cloud and its bounding box.

Figure 2.3: A point cloud and its bounding box.

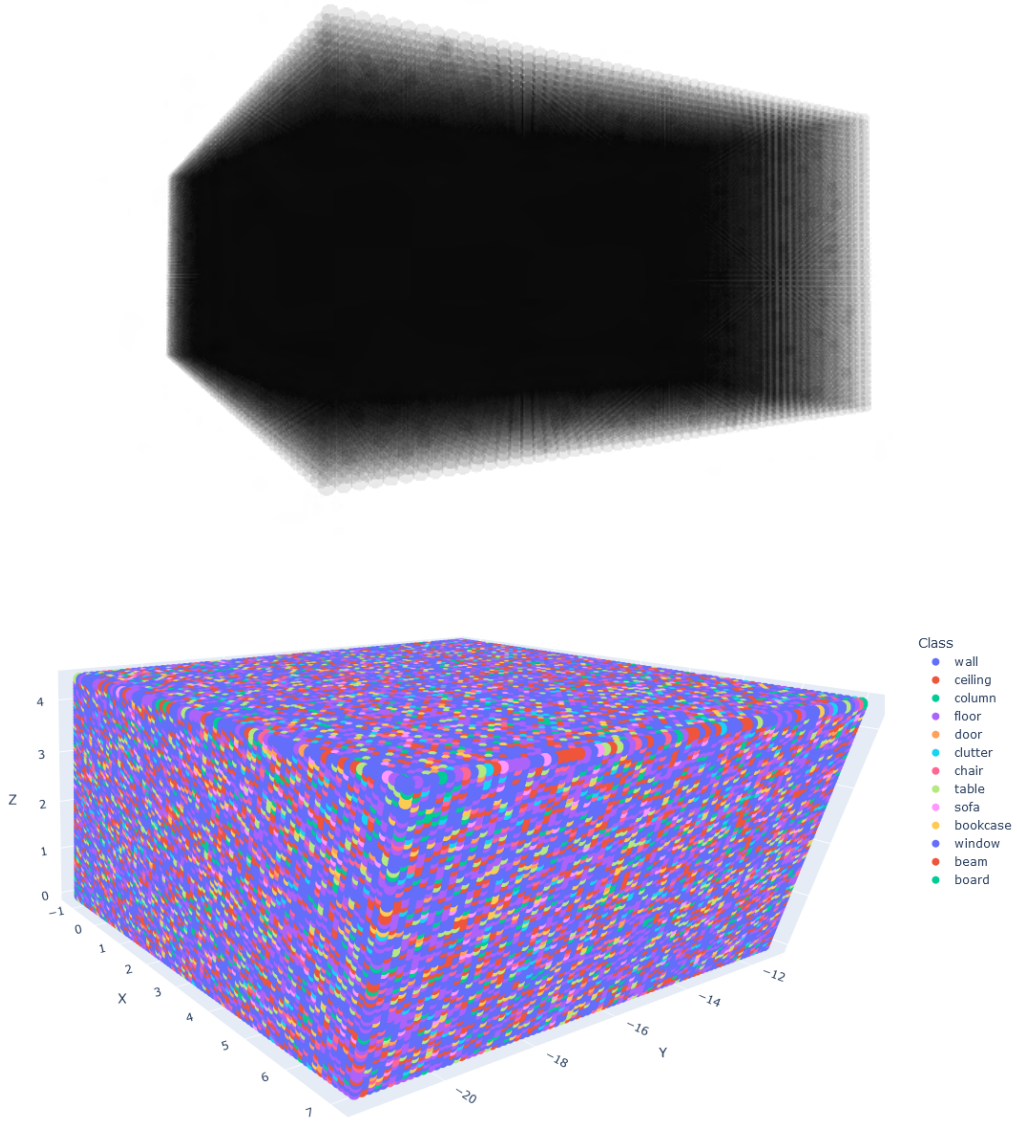


Source: (AUCLAIR; COHEN; VINCENT, 2007)

With the sizes of the volume, we create a uniform distribution of the points through the volume. The remaining points from the intervals division are distributed randomly.

Figure 2.4 shows an example of our proposed baseline, the small opacity from the visualization allows us to see the uniform distribution and the random points from the remainder. We can also see the segmentation of the baseline made by the SoftGroup model, discused in section 3.2. The noisy segmentation determines the lack of a bias in the results, suggesting that our proposed baseline is a good approach.

Figure 2.4: Plots of the proposed baseline and its segmentation.



Source: The Author

2.2 Model wrappers

The attribution algorithms need a target class to evaluate the contribution of the input features to the scalar output, in our case, a score that represents the target class. For classification tasks, each score defines the probability of the total input being classified as a certain class, and the attribution algorithms are useful for thse classification models where the outputs have one group of score values for the entire input. A group of scores defines the score values for each possible class. The size of the output for our task, however, is the same as the size of the input. More specifically, the output has a group of scores for each point, making it impossible for the algorithms to identify which group of scores to analyze and attribute.

Fortunately, we can use some strategies to reduce an output of a segmentation model to an output of a classification model, in order to obtain only one group of scores. Each strategy depends on which segment of the input we want to check the attributes, and they depend on a specific selection of these scores. For Nubilum, we defined three strategies. The first one allows us to attribute the points that contribute to all classifications of a specific target class. The second allows us to select an instance of an object of interest and analyze the contributions of each point to the classification of this object. The third allows us to check the attributions for the classification of a single point.

These strategies are done by creating a wrapper function called *model wrapper*. As the name suggests, it wraps the model by executing it with a specific input with an additional computation over the model's output. Nubilum's model wrappers, in general, after the model computes its forward computation, execute a selection of points, followed by a sum of the scores from the selected points.

2.2.1 Scene summarization for specific class

Captum tutorials[5] describe how to use its attribution methods in semantic segmentation tasks. The most simple and intuitive way to do this is to accumulate all the score values for each class. The attribution algorithms do not interpret the scores as probabilities, as long as the highest score represents the target class, accumulating the scores is not a problem. By doing this, we can summarize all points classifications in the scene and have an idea of the class distribution of the scene. However, this can mislead us into in-

[5]https://captum.ai/tutorials/Segmentation_Interpret

correct assumptions of how the segmentation occurred and, consequently, the attribution. After all, points can still have high score values for classes other than the ones predicted for them

Imagine a scene with two possible classes for each point. A model can predict that all points belong to class 1, with a score value of 0.6 for class 1, and 0.4 for class 2. If we execute this model wrapper, ignoring the results from the original output, the assumption that we would have is that 60% of all points are classified as class 1, and 40% are classified as class 2, which is incorrect.

A way to correct this problem is to consider, in the sum of the scores, only the score of the predicted class. This way, we have a summary of the scene, and the lower scores do not introduce a bias in this summary. In our last example, as all the points belong to class 1, only the scores with a value of 0.6 will be added, and class 2, in the summary, will have a total score value equal to zero. This can introduce a loss of information from the segmentation process, but it works for attribution methods since we are only interested in the target class, the points that are classified as this target, and the points that contribute to these classifications. With this, we have a model wrapper that can summarize the total score of an specific class of interest, we call this model wrapper as a summarized model wrapper. The summarized model wrapper can be useful in attribute methods to identify general mistakes that a model makes to a specific class and allow the developers to refine it in the training phase. Considering that $\boldsymbol{y}$ is the output of a segmentation model in the form of a $N \times M$ matrix, containing, for N points, the score values of M classes, and $\boldsymbol{y'}$ the output of the summarized model wrapper, containing only one vector of scores of M classes, equations 2.3 and 2.4 shows how the score accumulation occurs.

$$PredScores_{ij} = \begin{cases} y_{ij}, & \text{if } y_{ij} = \max(y_i), \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

$$y'_j = \sum_{i=1}^{N} PredScores_{ij} \tag{2.4}$$

2.2.2 Per scene object

A summary of the scene can be interesting at first glance. However, our domain demands a more meticulous interpretation of attributes, mainly for the identification of

particular objects and for an explanation of the model's output for these objects. To find specific problems and explanations, we want to have a more rigorous point selection. Returning to the AV problem, imagine a vehicle can have an outstanding performance, but sometimes it ignores the speed limits of a road. Assuming that the car sensors are not the problem, this indicates that the vehicle's model may potentially not be identifying road signs. This model's developers should be able to identify why specifically road sign objects are not being detected, revise the model, and amend it. A model wrapper that can attribute points based on their contribution to road sign objects could help solve this problem.

In general, instance segmentation datasets contain Ground Truth (GT) labels of the object instances for each feature, in our case, the points. We can use the GT labels, the predicted instance labels, or a custom point selection in a model wrapper to accumulate only the scores of points that belong to a specific instance and attribute the features that contribute to the prediction of these points. With this, we can reduce the scope of our analysis and focus our interpretation on a specific segment of the scene. This is helpful to identify the contribution of the scene context to the classification of an object. For example, in a conference room, a chair is classified as a chair because of the local features (e.g., points are in the format of chair legs and back) or because of the scene's global features, such as other close chairs and a table in the same environment. We define this model wrapper as object wrapper or instance wrapper. The mathematical definition contains only a difference, compared to equation 2.4. Now we introduce a binary mask $\boldsymbol{m}$ with size N where m_i is 1 if the point to be accumulated is part of the object of interest, and 0 otherwise. Equation 2.5 shows how the wrapper output is computed.

$$y'_j = \sum_{i=1}^{N} m_i \cdot PredScores_{ij} \tag{2.5}$$

Since models for our task usually do not use instance label tensors as a feature in the input, we can not directly pass these tensors as part of the input to our model wrapper. Thankfully, Captum allows us to use additional arguments in PyTorch's forward functions, in our case the model wrappers. This *additional forward arguments* helps us to execute our pos-model computations without having to introduce unnecessary features to the model's input.

2.2.3 Per point

Finally, we have a model wrapper to help us comprehend minor mistakes of models. By selecting only the scores of one point of interest, It is possible to attribute the features that influence the classification of the point of interest. This can be useful for fine adjustments in the model, such as points from objects too close to each other that were incorrectly predicted (e.g. points from a wall being classified as a door), since the attribute algorithms might attribute big values for points from wrong objects.

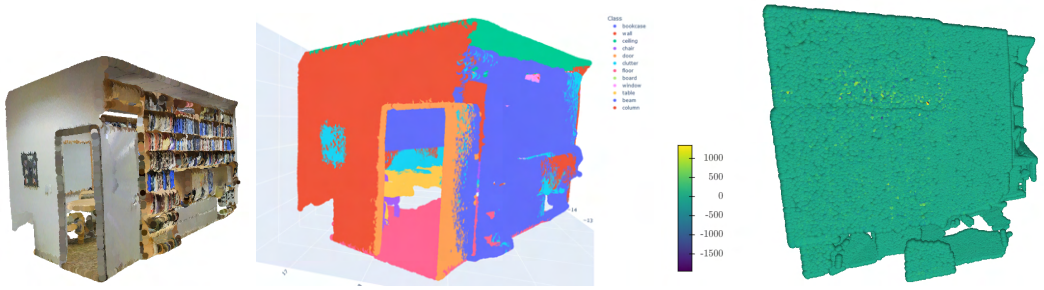
2.3 3D visualization

Attribution values themselves do not offer too much information for humans. We can use data science techniques to retrieve statistical information from them, but the results could be insufficient to explain the model to a human. Our data domain consists of 3D visual information, any type of knowledge related to the points also needs a visual representation to a fully comprehension. Therefore, we offer easy access to 3D plot functions by means of 2 visualization libraries, Plotly and K3D-Jupyter. We use two libraries because we notice undesired behaviors in their individual use, and one library can cover the flaws of the other, letting the user decide which one to use depending on the circumstance. Section 2.3.3 describes the pros and cons of each library and how they complement each other.

We implemented three main types of plots for Nubilum, all utilizing 3D scatter plots and the coordinates feature from the points. The first one is the original point cloud. It contains the colors of the scene environment when scanned. It utilizes as arguments the points coordinates and colors, our main features for the model. In the second one, we can visualize the model's output, the semantic segmentation results. When we apply an $argmax$ function over the scores, we can obtain the predicted class of each point. For this visualization, instead of using the original colors, we use a color scale (matplotlib's *tab20* qualitative colormap[6]), where each color indicates a possible class. We use this color scale because it contains discrete colors and has a good amount of colors to represent the classes of existent datasets. The third plot is dedicated to visualizing the attributions. Figure 2.5 shows examples of the three plots. Section 2.3.3 contains a recommendation for the usage of the three visualization methods according to the plot tools we use.

[6]https://matplotlib.org/stable/tutorials/colors/colormaps.html

Figure 2.5: Visualizations for a point cloud, its segmentation, and its explanation, containing the attribute values computed for each point.



Source: The Author

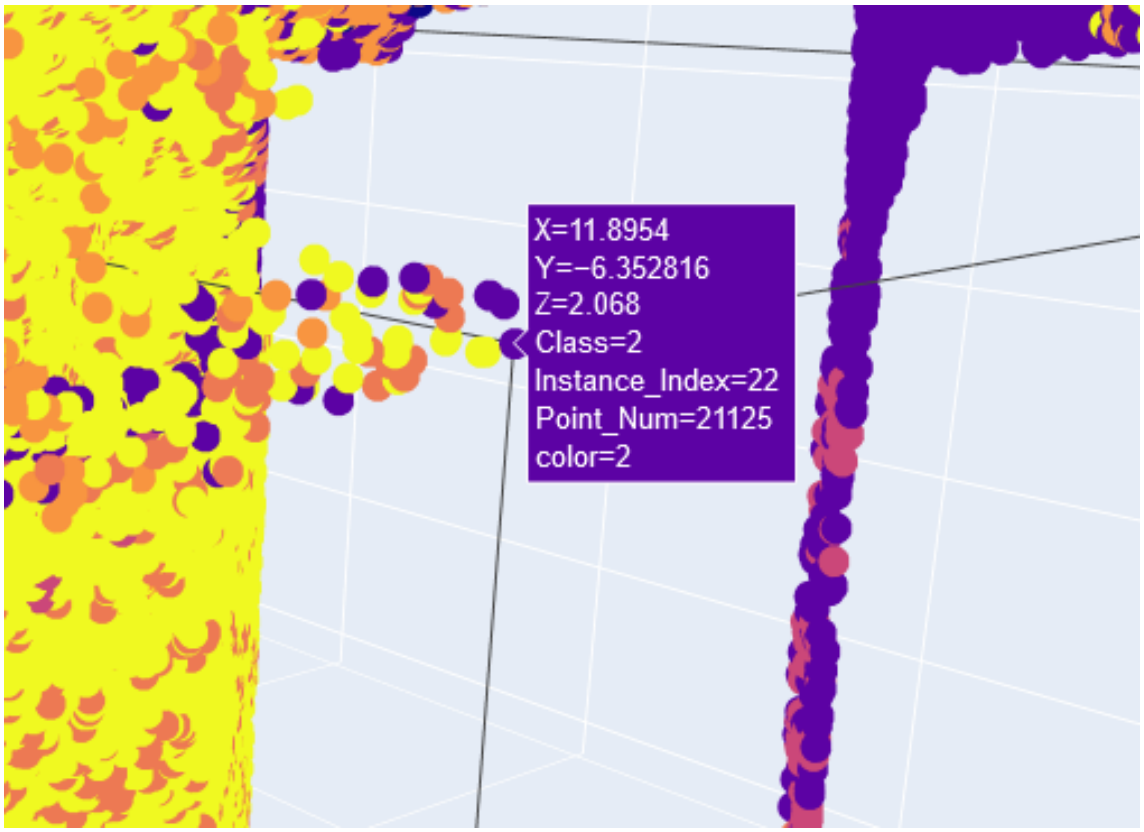
We also provide an assist plot to highlight a point of interest in the point cloud, useful to help us select a point for the point model wrapper (section 2.2.3). Since this assist plot is similar to the original point cloud visualization, where we do not want any extra information in the plot, all considerations towards the plot that shows the original point cloud are valid for this assist plot as well.

2.3.1 Plotly

We utilize the Plotly Express' *scatter_3d* function to draw a 3D scatter plot of the point cloud data. We chose Plotly because it has a particular feature that enhances our comprehension of scene details. By using the mouse cursor, we can hover it over a point to show a pop-up description of the point, similar to what we do with files in a graphical operating system when we want easy and fast access information about a file by hovering over it. Figure 2.6 shows an example of the hover process in the plot. The pop-up description has useful information such as the point coordinates, the color, the point index, and, depending on the type of plot we use, specific information. The segmentation plot contains specific information about the class and the instance index, and the attribute contains the attribute value for the point. We also chose Plotly because of its useful way of organizing the data as a dataframe, making it simple to define what information we want in the plot and in the hover. This includes the name of the class objects of the model, where we can have a color legend that can help us identify which colors are which objects. Figure 2.7 has an example of this color legend.

Unfortunately, point cloud data tends to be large, and Plotly can have some perfor-

Figure 2.6: Plotly's hovering pop-up description.

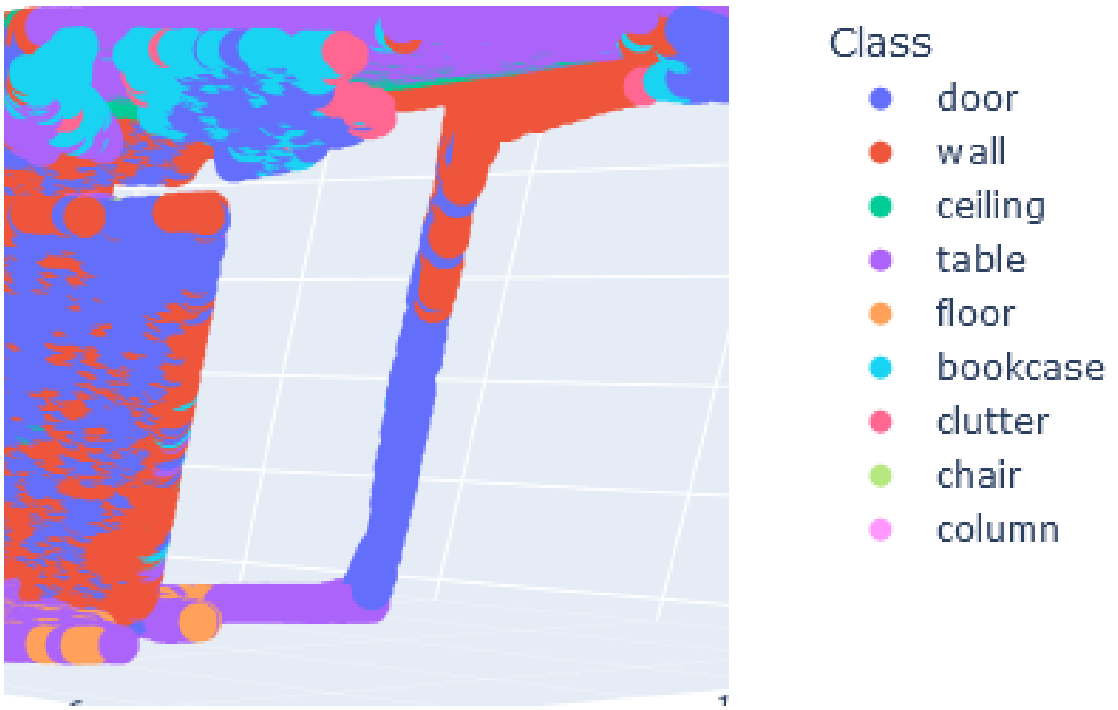


Source: The Author

mance problems by rendering bigger scenes since it uses the CPU for that. Usually, this kind of analysis is done in Jupyter notebooks[7] because it serves as a "playground" to the developer thanks to its fast cell-based development environment. Multiple plots like ours can impair the performance of the notebook. A strategy to mitigate the slow performance is to use other applications, like the browser to render the plot. However, it is not enough for bigger scenes. Another disadvantage of Plotly is the impossibility of using another way to draw points other than a flat circle and the absence of lighting, which could potentially spoil a human 3D spatial visual understanding of the scene. Therefore, we use the K3D plot library to help achieve both performance and spatial understanding.

[7]https://jupyter.org/

Figure 2.7: Plotly's color legend.



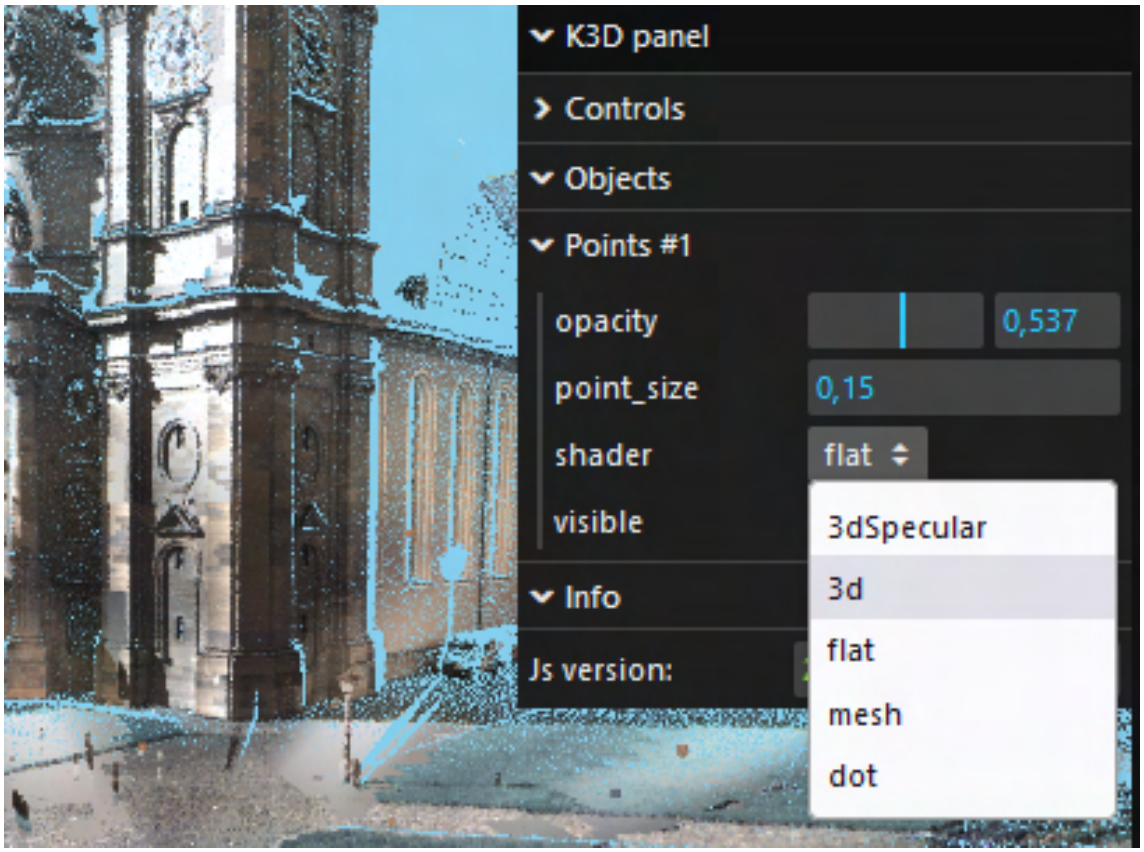
Source: The Author

2.3.2 K3D-Jupyter

We use K3D's *point* function to draw directly the point cloud data. Unlike Plotly, where we can organize all our data in a unique structure of a dataframe and have access to more point information through data analysis, (which is useful for a model's explanations), K3D-Jupyter does not have any means to display additional information other than the colors of the points and a color map. However, WebGL is more rendering-focused, offering support for GPU rendering, and increasing performance. It also offers shading options (Figure 2.8) that can be manipulated in real-time, which makes it easier to understand the 3D scene composition, instead of Plotly's unique flat option.

Added to the lack of means for viewing additional information, K3D also does not have the option to use discrete color scales like Plotly, which makes it difficult to visualize and understand, for instance, the points classification. Even the possibility of creating custom color scales does not solve the problem. Between each color, K3D creates a continuous interpolation between them, leaving the colormap that appears with the plot uninterpretable.

Figure 2.8: K3D's shading options.



Source: K3D-Jupyter documentation

2.3.3 Recommendations for when to use each type of plot

In summary, we describe the pros and cons of each plot library for our work.

Plotly

- Pros
  - Hover pop-up information
  - Possibility to organize data as a dataframe
  - Disponibility of all Matplotlib's color scales
- Cons
  - Slow performance
  - Flat points draw

K3D

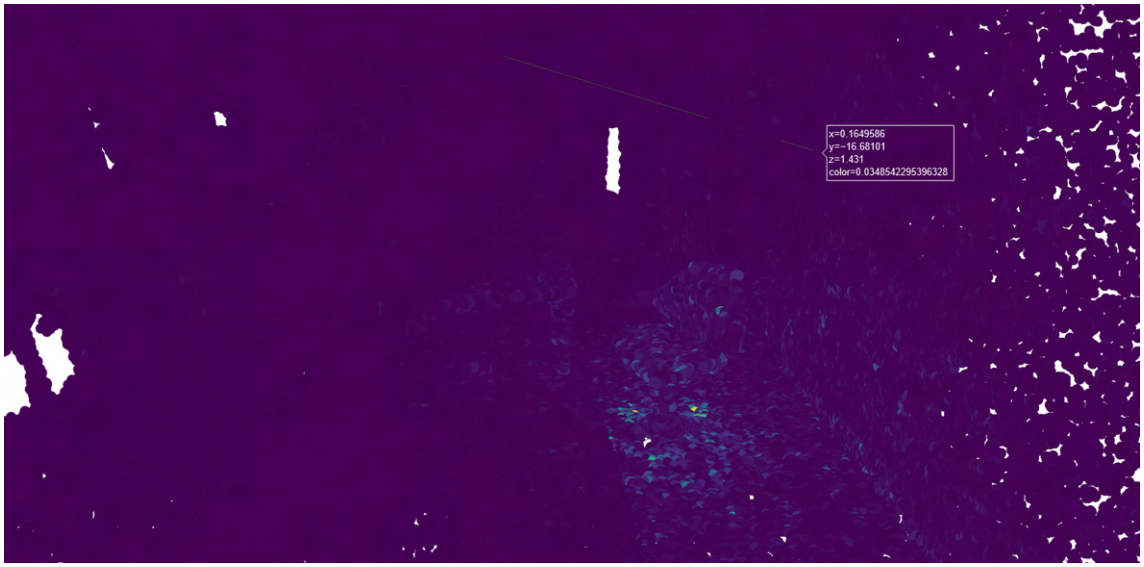
- Pros
  - Fast Performance
  - Shading Options
- Cons
  - Lack of means to show extra information
  - No discrete color scales

Based on the pros and cons, we can direct the user to choose which plot library to use to visualize the point cloud data. In general, we want to give priority to using Plotly's plots because of the extra information by hovering, if the user notices a slow performance, It is recommended to change to K3D plots. The first plot, dedicated to the original point cloud, does not have too much information since It is just the original input. Therefore, It is interesting to use K3D because of its performance and its shading options. The different shading options allow a better understanding of the scene and the objects that compose it. For the second plot, with the model's output, we prefer to use Plotly. We can use the hover's pop-up description to obtain information useful to calculate the attributes later, such as a point classification, and its instance label. For the third plot — the values of the attributes — both libraries have color scales to facilitate the understanding of the attributions. Plotly can still have the pop-up with the exact attribute value for a point, and K3d will always have superior performance. However, Plotly flat point rendering spoils the understanding of the scene, since It is difficult to perceive the position of the objects with the same color in a color scale (Figure 2.9). K3D, by its shading options, allows us to have a better understanding of the scene by using color scales. Regardless of the user's preference, we explicitly define in Nubilum's visualization functions the tradeoff between Plotly and K3D.

2.4 Other functionalities

In addition to the attribute methods, model wrappers, and 3D visualization plots. Nubilum assists developers with other functionalities. The first one was already mentioned in section 2.1.2.1, it creates a baseline according to the number of points of an input. The baseline consists of a uniform distribution of black points through a volume.

Figure 2.9: Example of Plotly's 3D scene attributes for a chair in a conference room. All points have a flat purple color. Making the scene impossible to understand.



Source: The Author

We also offer a method to sum the attributes. The attributes functions give values not directly to the points, but to each component of its features. This means that point positions will have attribute values for x, y, and z coordinates, and the point colors will have attributes for the red, green, and blue channels. We might want to analyze the attributes for each coordinate or channel but it would be more useful to aggregate this information to a unique value for each feature. A point p position attribute would consist of

$$PositionAttr(p) := attr(p_x) + attr(p_y) + attr(p_z) \tag{2.6}$$

where $attr(p_i)$ is the attribute value of a coordinate i from the point p. The same can be done to the color feature

$$ColorAttr(p) := attr(p_R) + attr(p_G) + attr(p_B) \tag{2.7}$$

where $attr(p_c)$ is the attribute value of a channel c from the point p. Finally, we can aggregate the feature attributes to obtain the final attribute of a point p

$$FinalAttr(p) := PositionAttr(p) + ColorAttr(p) \tag{2.8}$$

This feature was created to (1) facilitate the understanding of the scene contributions to lay people, that are not committed to understand the importance of a particular

coordinate or color channel, and (2) to aggregate the results for a full analysis. All the analysis made chapter 3 contains aggregated results from this feature.

3 SOFTGROUP INTERPRETABILITY VIA NUBILUM

To test Nubilum and its functionalities, we propose an interpretation and analysis of an existent model by using Nubilum's methods. We chose the S3DIS dataset and the SoftGroup model. We expect that Nubilum can help us to interpret the results of SoftGroup and understand how the model segments point cloud data. More specifically we want to (1) be able to untangle any black-box model and understand how a model obtains its results what exactly part of the features it considers and how the features relate to each other to generate the output, and (2) understand if Nubilum is capable of delivering good explanations.

We elaborated three groups of questions to help us guide through the usage of Nubilum. The first ones are related to mistakes in the model's decision-making in general.

- When the model makes an incorrect prediction, why it is incorrect?
- When do these mistakes happen? Is there any particular pattern in the input when these mistakes happen?
- In cases where there is a misclassification, what is happening?

The second one is related to the context of a scene and the relation between the features. The scene context has a more semantic meaning for us, meaning that we want to check if the model understands the disposal and relation of the objects in the scene and takes its decision by this context or if it just considers how an object points are disposed of. For example, in an office room, we know by the context that, if we have a desk, there is a good probability that a chair is next to this desk for a person to sit.

- How important is the scene context for the model's decision-making?
- The model cares more about local features or more about the general context where a point or object is inserted (global features)?
- If it considers the scene context, how big is the context that it considers? 1 Meter, more or less?

The third group of questions is related to Nubilum's capabilities to explain the model's predictions.

- Is Nubilum capable of giving the developer enough information through its attributes visualizations?
- With this information, can a human understand the model's mistakes and assumptions of the scene context?
- When the model correctly predicts the points class, does the attribution explanation make sense given the result?
- Are Nubilum's visualization tools easy to interact with Point Cloud data?
- The visualization tools allow a human to understand the attributions by their colors. It is easy to distinguish the attribute values of two or more points by their colors?

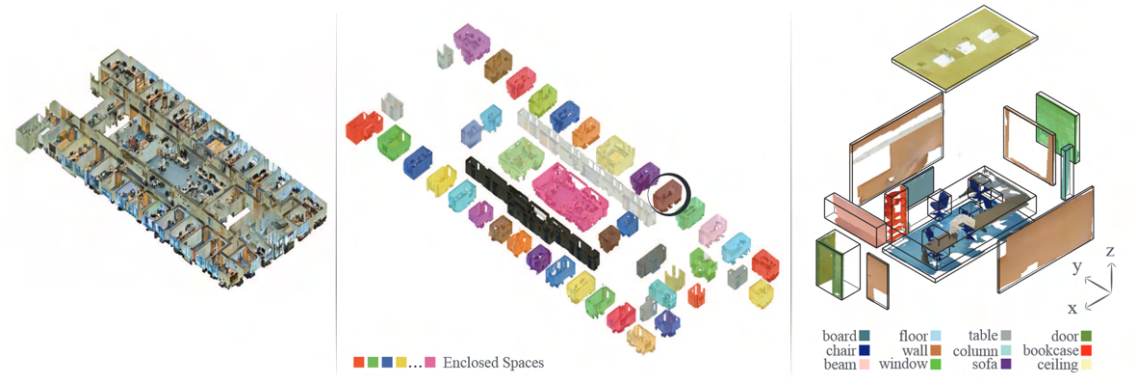
Section 3.5 contains a summary of the results gathered in section 3.4 and answers the above questions.

3.1 S3DIS

The S3DIS dataset was proposed by Armeni et al. (2016). It consists of colored 3D scans of the indoor area of an office building. The data is divided into small areas, such as small offices, corridors, conference rooms, and restrooms, and each of these areas is used as an input for the SoftGroup model. Each input is a collection of points with coordinates and colors. There are 12 main semantic elements divided into two groups, the structural elements (*ceiling, floor, wall, beam, column, window* and *door*) and commonly found items and furniture (*table, chair, sofa, bookcase* and *board*). There is also a 13th semantic element called *clutter*, which is designed to include any other object that does not fit in the other 12 categories. Usually, It is used for small objects such as monitors, cups, computers, and printers (ARMENI et al., 2016). In total, the dataset offers 272 room instances, divided (not equally) into 6 Areas. Figure 3.1 shows an overview of the dataset, where a floor of an office building is divided into areas that constitute the dataset instances.

We chose this dataset mainly because SoftGroup already has implemented a model for it. In addition, indoor scenes, by having a limited space, have a limited number of points and a better rendering performance, which makes the scenes better to visualize with our plot tools.

Figure 3.1: The S3DIS dataset.



Source: (ARMENI et al., 2016)

3.2 SoftGroup

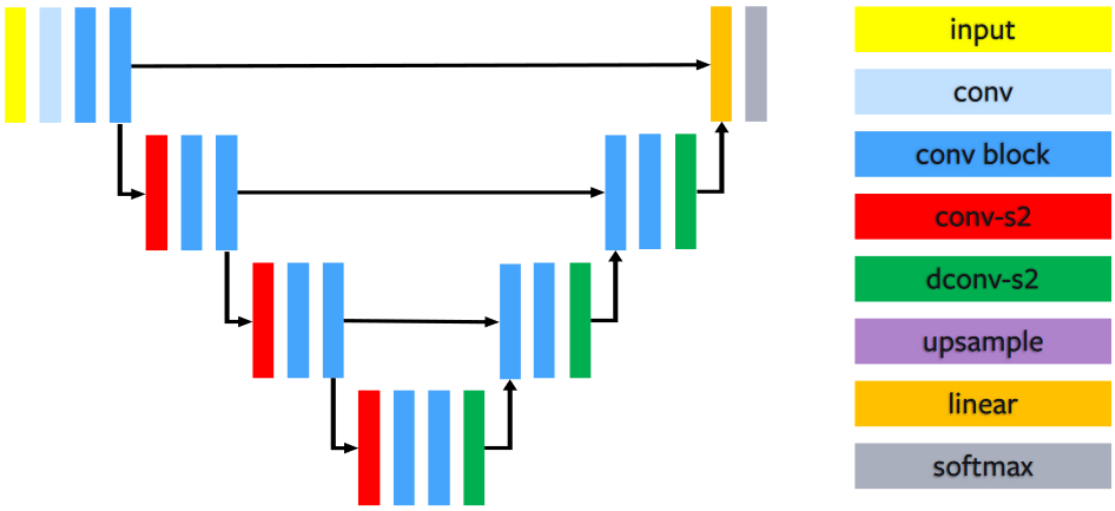
Vu et al. (2022) proposed a model for instance segmentation on 3D point clouds called SoftGroup. The proposed work is more focused on solving the propagation of semantic prediction errors to the instance predictions by grouping the semantic scores instead of defining a unique semantic prediction by them. The first part of its architecture, which the authors call *semantic branch*, uses a U-Net as its basis for semantic segmentation, which is part of the architecture we are interested in using.

A U-Net is a specialized type of CNN used for semantic segmentation, proposed by Long, Shelhamer and Darrell (2015). Its architecture resembles a "U" shape because it consists of a contracting path (left side), where we can obtain useful features by downsampling an input, and an expansive path (right side) where we upsample and refine an output. Figure 3.2 shows an example of a U-Net. Unfortunately, Softgroup authors do not enter details about the inner architecture of its U-Net.

We chose SoftGroup because (1) it is one of the works in the 3D segmentation task with an available pre-trained model and with an online repository[1], (2) it supports multiple datasets, including S3DIS, and (3) as it is not the state-of-art in the 3D semantic instance prediction task, according to ScanNet Benchmark[2], and its already known that the semantic branch in fact does mistakes (as the authors propose a method to not propagate these mistakes), it will be useful for our test to have a model that can make mistakes and therefore help us to interpret them.

[1] https://github.com/thangvubk/SoftGroup
[2] https://kaldir.vc.in.tum.de/scannet_benchmark/semantic_instance_3d

Figure 3.2: Example of a U-Net.



Source: (GRAHAM; ENGELCKE; MAATEN, 2017)

This work will abstract only the semantic segmentation of SoftGroup's architecture, and will not utilize parts responsible for the instance prediction. The pre-trained model is trained using the S3DIS instances from areas 1 to 6, excluding 5. Area 5 is used to test the model and will be used to interpret the results with Nubilum.

3.3 Nubilum's test metodology

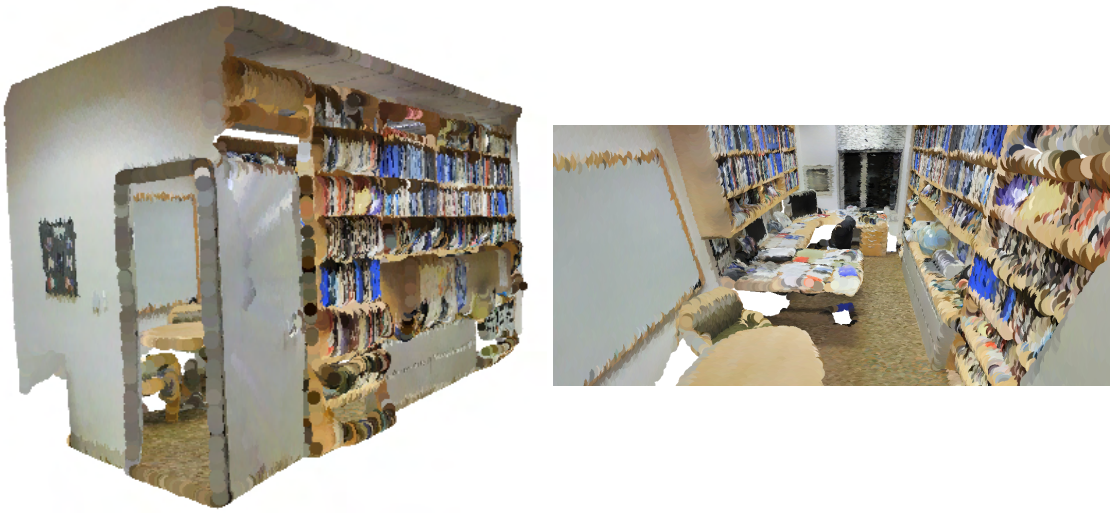
We use Nubilum to interpret 2 different scenes from Area 5, a conference room (Area_5_conferenceRoom_2) and an office (Area_5_office_10). We execute an analysis over the two attribute methods, Saliency and IG, described in sections 2.1.1 and 2.1.2, using the three model wrappers (section 2.2). First, in section 3.4, we share the model's output for both scenes and then we discuss specific areas of interest that we want explanations from Nubilum. After this, we execute the attribution computations for Saliency and IG, and we share the explanations and our conclusions about the explanations, where we also answer the questions we made at the beginning of the chapter. Figures 3.3 and 3.4 shows the two scene we're exploring, the images on the left shows a view from outside the scene, and on the right a view from inside. We use Plotly to show the model's segmentation results, to retrieve further details for our analysis and K3D to show the original scenes' point clouds and the attributions gathered from Captum's attribution methods. All the experiments were conducted in the following hardware:

- CPU: AMD Ryzen 5 3400G
- RAM: 16 GB
- GPU: NVIDIA GeForce RTX 3070

Figure 3.3: Area_5_conferenceRoom_2

Source: The Author

Figure 3.4: Area_5_office_10

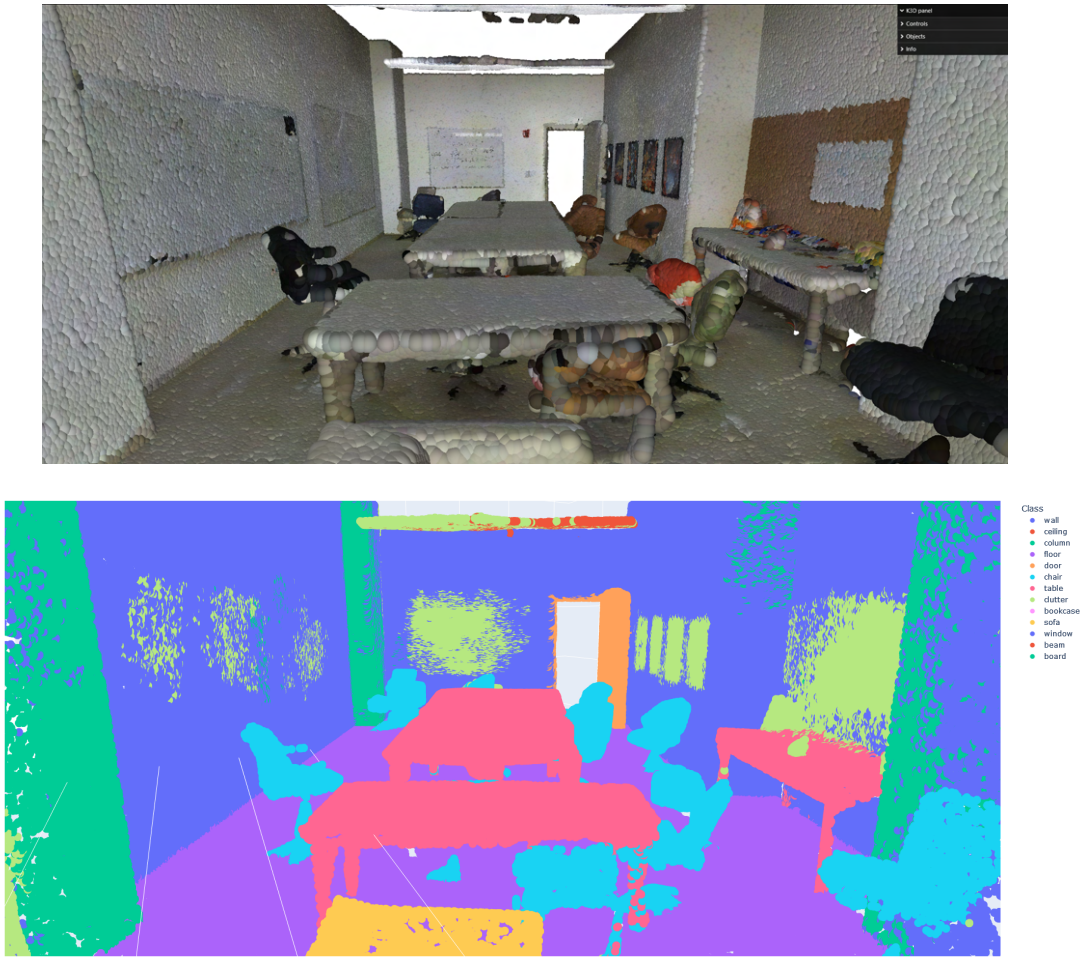


Source: The Author

3.4 Semantic segmentation results

The model achieves a good performance over the two scenes. Figures 3.5 and 3.6 show that most objects' points are well classified. However, there are some inconsistent limits between some of the objects. One example can be found in figure 3.5, in the wall back in the room next to the door, the area with the whiteboard (which is being classified as clutter) are not well limited, and some points are being classified as a wall.

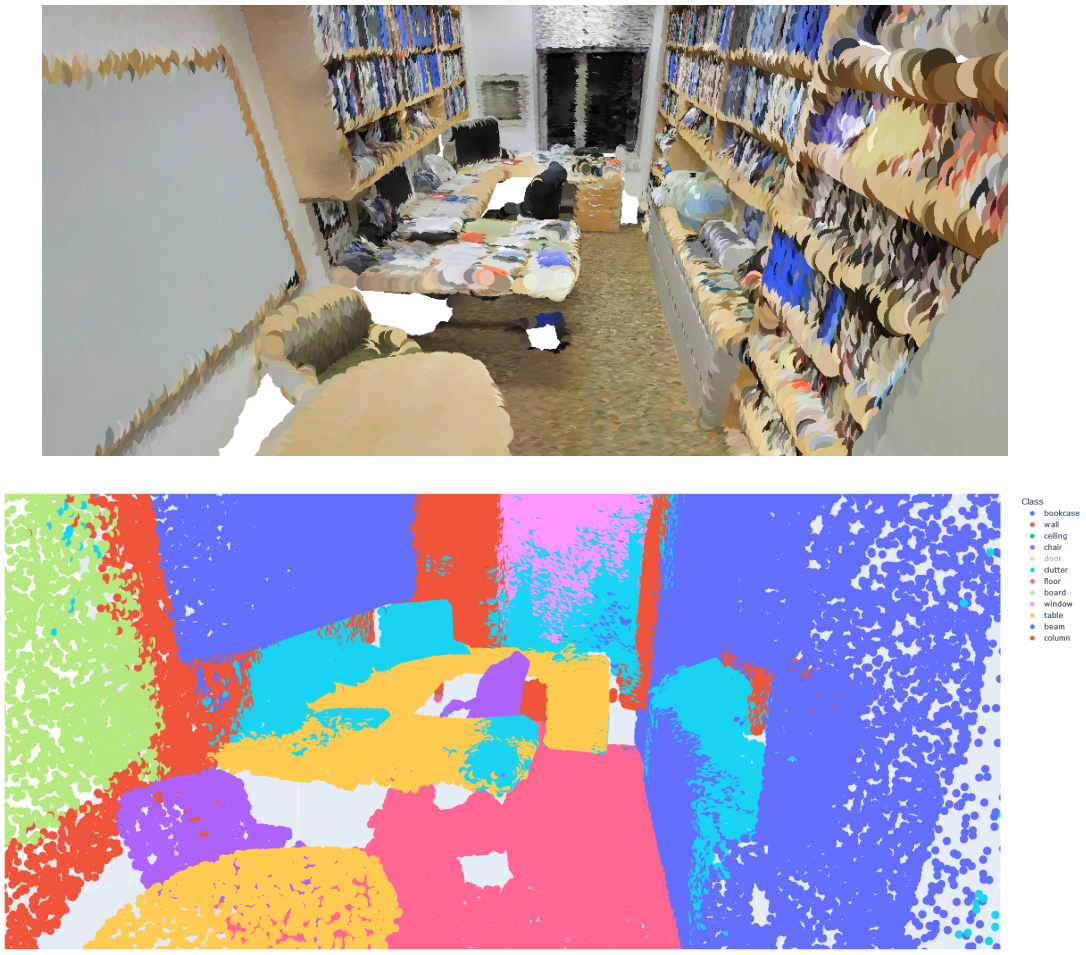
Figure 3.5: SoftGroup's semantic segmentation for the conference room.



Source: The Author

We propose to analyze three aspects of each scene in order to test our model wrappers. The first analysis is scene summaries of specific classes. For the conference room, we chose the *table* class. In a room where all the tables are surrounded by chairs, we want to check if not only the points classified as tables contribute to their classification

Figure 3.6: SoftGroup's semantic segmentation for the office.



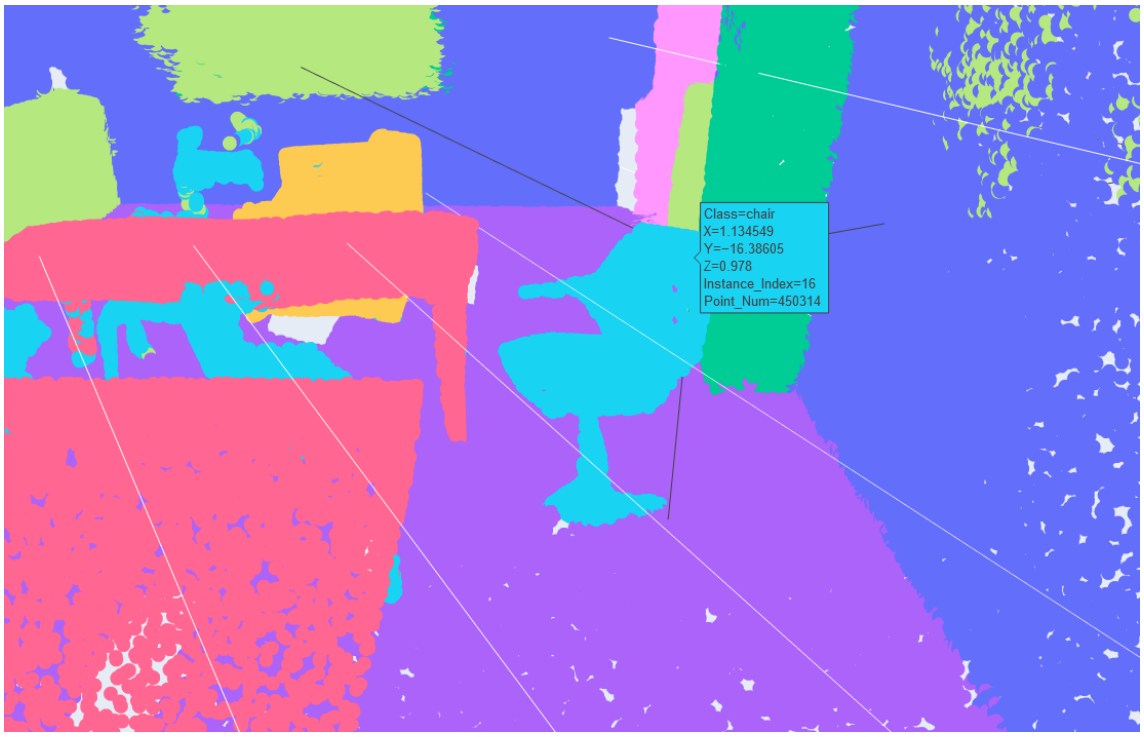
Source: The Author

but also if points belonging to other objects also contribute. For the office scene, we want a summary of the bookcases, some objects like the ones classified as *clutter* and the walls can influence the classification. We also want to check why some points of the door were misclassified as a bookcase.

In the second model wrapper, we want to analyze specific objects of the scene. For the conference room, we chose one of the chairs (figure 3.7), we want to check if other chairs and the presence of a table can contribute to the classification of the selected chair points. For the office, we chose the window at the end of the room (figure 3.8). We noticed that part of the window was misclassified as a clutter and we want to understand why.

For the third model wrapper, we want to analyze specific points of the scenes. We decided to analyze misclassifications in both scenes, however, these misclassifications

Figure 3.7: Conference room's chair to be analyzed.



Source: The Author

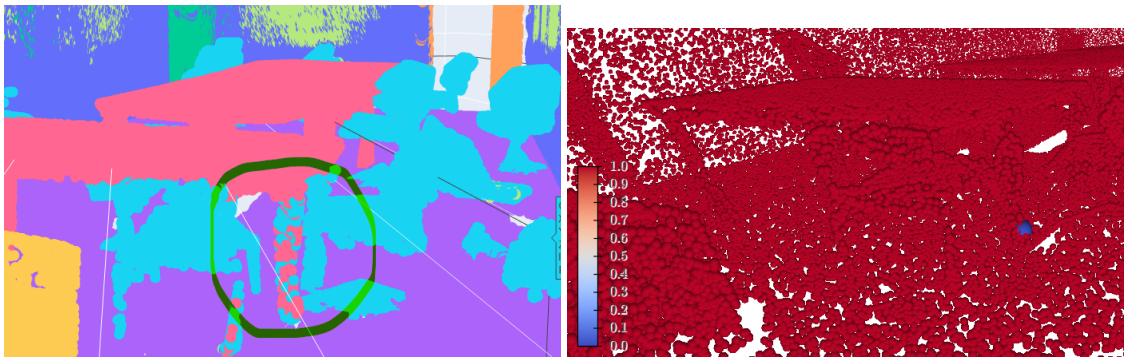
Figure 3.8: Office's window to be analyzed.

Source: The Author

have different natures. The first one, in the conference room, is a point on a group that is being misclassified in one of the table legs. This leg has points being classified correctly as a table and points classified incorrectly as one of the chairs next to it (figure 3.9), and we chose one of the points being classified as a chair. The second misclassification, in the office, is an inconsistent door point, next to a board and chair, where there is no door around (figure 3.10). We believe that the point could resemble a door part and we want to

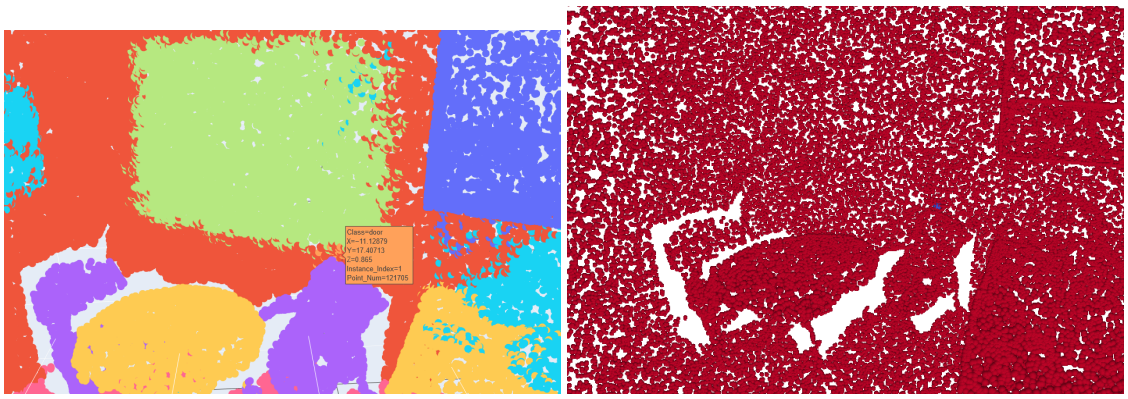
verify if the points around it contribute to the incorrect prediction.

Figure 3.9: Conference room's table leg point.



Source: The Author

Figure 3.10: Office's inconsistent door point.



Source: The Author

3.4.1 Nubilum's explicability results

As said before, we use the saliency and integrated gradient methods to get the attributes. We gather 6 groups of attributes using each method. Section 3.4.2 describes the results from Saliency for both rooms and section 3.4.3 describes the results from IG. Despite the similarity between the two methods, we have differences in the attribute values from one method to another that are interesting to comment on.

The first one is related to the difference we have in the magnitude of the value, which is evidenced by the color mapping used in the plots. Saliency does not have too

many negative values in its attributes, which ends up leaving the point cloud with a darker color of the color scale we used (*Viridis*). In IG, thanks to the accumulation nature of the method, we see more negative values accumulated, which ends up leaving most of the point cloud with blue colors.

The second difference is in the importance of the attribute values. IG refines more the attribute values, leaving high attribute values only for the points considered the most important over all the straight line between the baseline and the input. In saliency, we have more immediate and basic results, attributing points based on the gradients for a unique input. We can summarize this with saliency attributing points that can be relevant for an analysis, and IG attributing which points are really important and which are not.

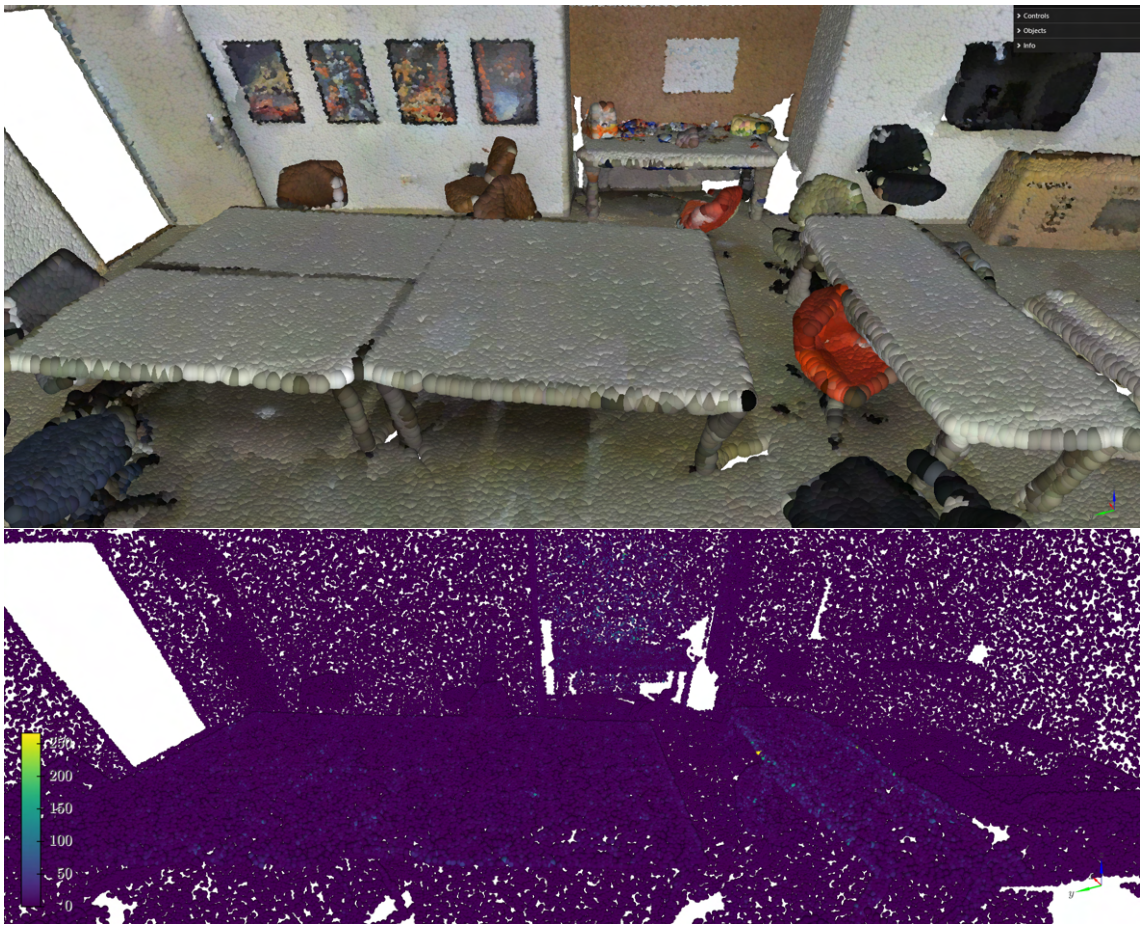
3.4.2 Saliency results

3.4.2.1 Conference Room

The summary analysis of the table brought to us a good insight. As can be seen in figure 3.11, we have points with relatively high attribute values (in blue) in the points belonging to the tables, mainly on the sides of the table. What is interesting is that we have a table next to a wall, and points in this part of the wall also have high attribute values. We interpret this result as an influence of other scenes from S3DIS, as most office scenes always have a table next to a wall.

For the attributes in the chair object, as figure 3.12 shows, we can notice an area around the chair that is being considered for the chair's prediction. There is a radius of about 1 meter where the points inside of it have relatively high attribution values. One can say that objects like chairs are influenced by the ambient that they are inserted, especially from the floor. We calculated the ratio between the chair's attributes and the scene attributes, and we obtained a value of 0.0944, which means that of all contributions to the chair's prediction, less than 10% was from the chair itself. Specifically for this case, we also modified the scene by shifting the chair's position, creating new scenes, and we conducted the same experiment on these new scenes. The new scenes are the chair fallen to the floor, the chair on the table and the chair on the ceiling. Figures 3.13, 3.14 and 3.15 show the scenes, classification of the chair and the attributes for the class predicted. The chair on the table and on the ceiling were predicted as cluster, which means that the local features are indeed less important than the scene context. Table 3.1 contains the predicted

Figure 3.11: Summary of the attributions of the class *table* using saliency.



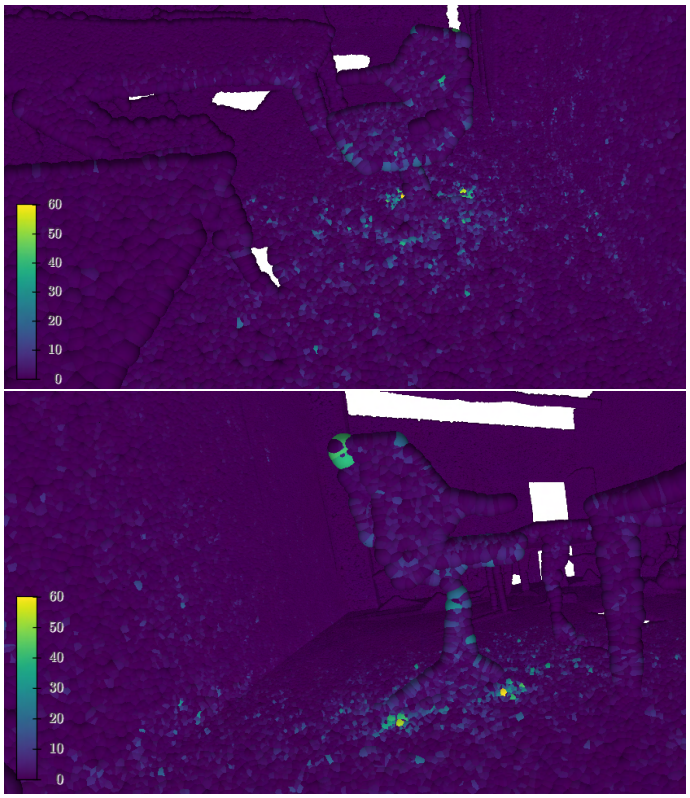
Source: The Author

class and its attribute ratio of the chair for all 4 scenes. The attribute ratio measures the contribution of all points form a particular group, in this case the chair, over the total contribution of the scene, equation 3.1 describe how the attribute ratio is computed, where n is the number of points in the scene, $Attr(i)$ is the attribution value of the point i and $\boldsymbol{m^a}$ is a binary mask where m_i^a indicates if the point i belongs to the object of interest a. We can see that the scenes where the chair was predicted incorrectly have bigger ratios, and we conclude that if the model can't understand the context where an object is inserted, it uses the *clutter* class for the object.

$$AttrRatio(a) := \frac{\sum_{i=1}^{n} Attr(i) \times m_i^a}{\sum_{i=1}^{n} Attr(i)} \tag{3.1}$$

An area of influence can also be detected in the analysis of the table's leg point being misclassified as a chair. The points from the chair next to the table's leg directly influence our point of interest. Since most of the table points are concentrated on the

Figure 3.12: The attributions for a specific chair at the conference room using saliency.



Source: The Author

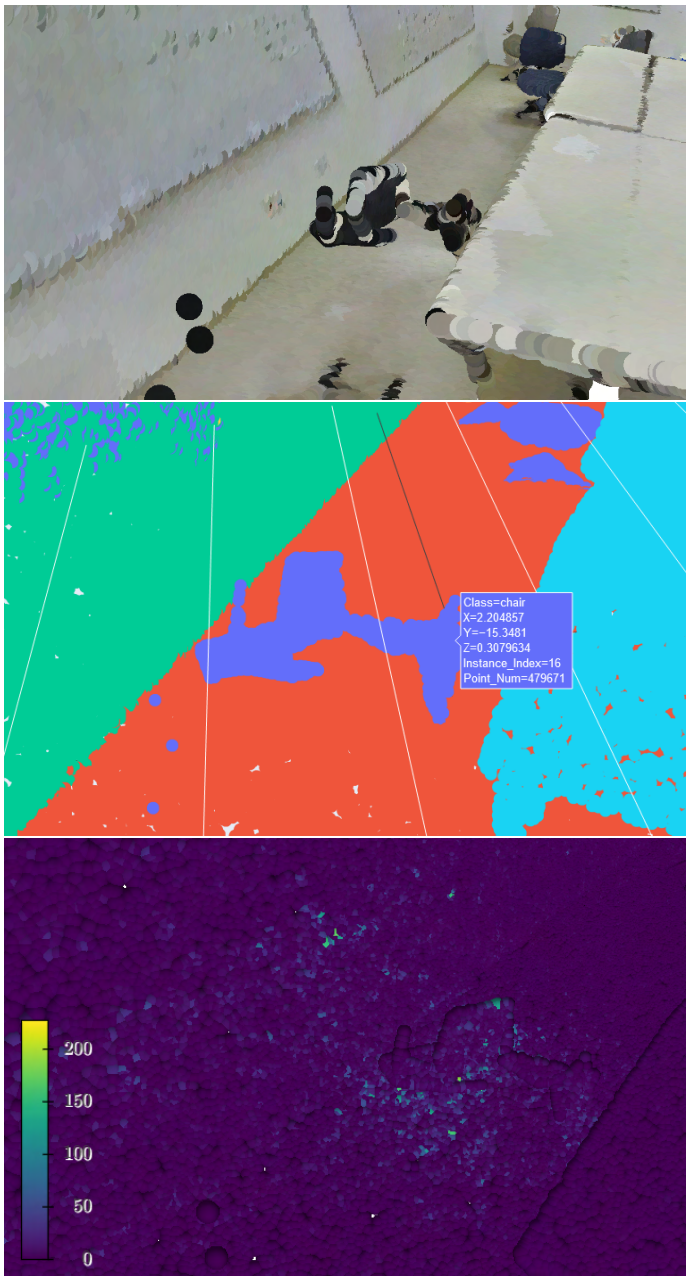
table's surface, points that are situated in a more distant area are influenced by other objects around them, in our case, as figure 3.16 shows, our point is being influenced by the chair next to the table's leg.

3.4.2.2 Office

The summary attributes for the bookcases are also similar to the tables in the conference room. Figure 3.17 shows that there are significant attribute values for the cabinets in the lower part of the bookcase and almost none for the objects above them. For the part of the door that was misclassified, one can say that it has the influence of the bookcase next to it, however, we do not have high attributes for the part of the bookcase next to the door. This could indicate the presence of high cabinets that can be similar to doors in other scenes since only the cabinets have significant attribute values.

For the window attributes (figure 3.18), differently from chairs in the conference room, we can not notice a clear area of influence, however, we can notice an aspect of its local features. The window prediction has a good contribution from the structure of

Figure 3.13: Scene, predicted class and attributes for the fallen chair.

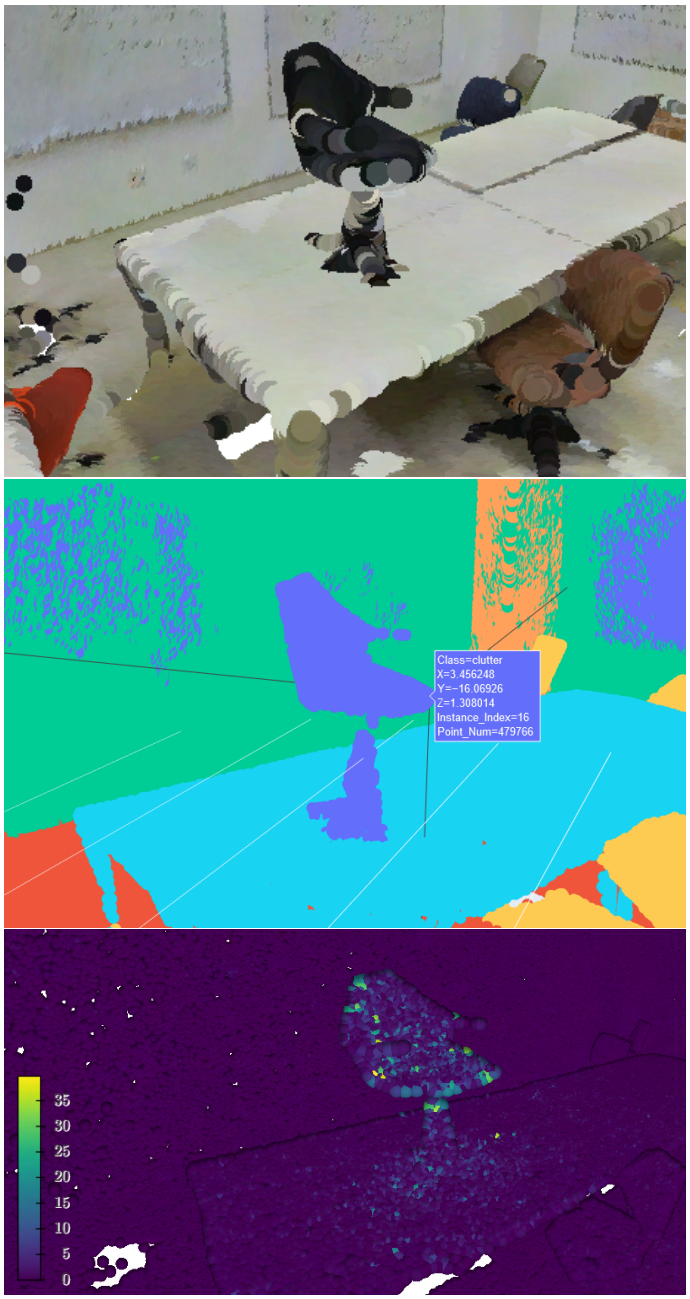


Source: The Author

its frame, but nothing from the area of the glass. The scenes from S3DIS have a black pattern with an amount of reflection from the room in their windows objects. Since the reflection can not be identified as a pattern and may change due to factors like the sensor position, the room object disposal, and lighting, the points where we have this reflection are not being considered by the model to predict windows.

For the point analysis of the office (figure 3.19), where a point is being misclassified as a door, despite a good visualization of the computed attributions that Nubilum

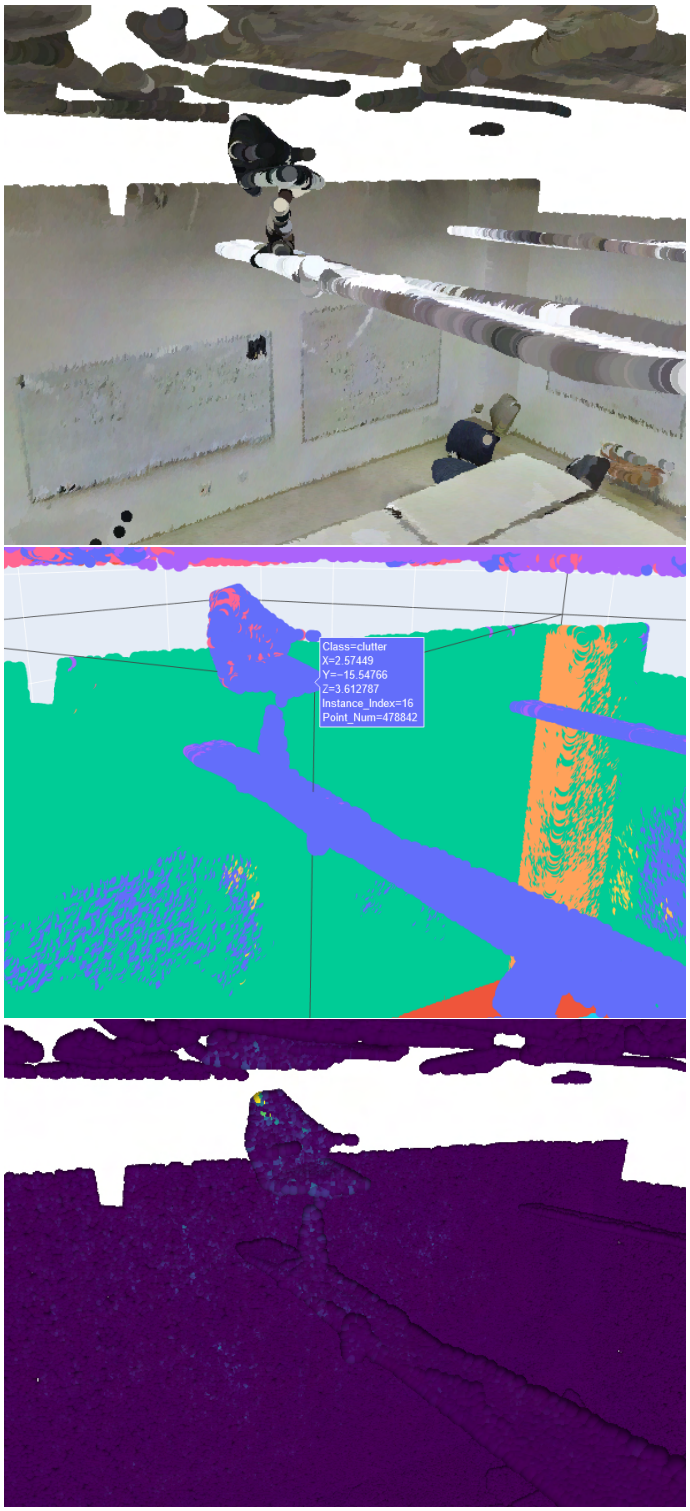
Figure 3.14: Scene, predicted class and attributes for the chair on the table.



Source: The Author

offers, these attributes weren't sufficient to interpret the situation. We can only see an influence from the board next to it and none from the office's door.

Figure 3.15: Scene, predicted class and attributes for the chair on the ceiling.



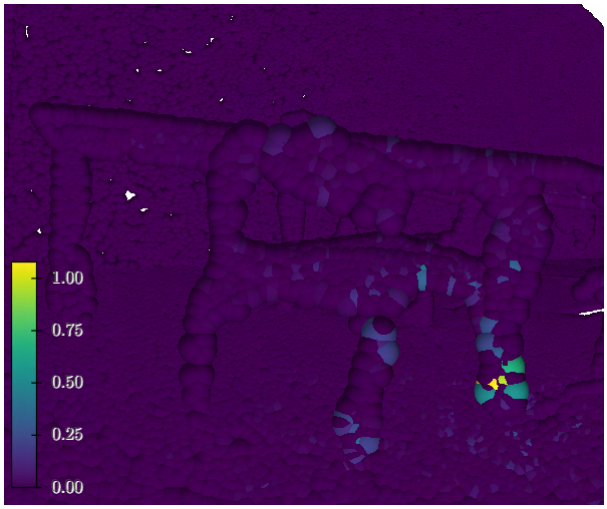
Source: The Author

Table 3.1: Predicted class and attribution ratio for the 4 chair positions.

Scene	*Predicted class*	*Attribute ratio for the pred. class*
Original	*chair*	0.0944
Fallen chair	*chair*	0.0779
Chair on the table	*clutter*	0.2888
Chair on the ceiling	*clutter*	0.1660

Source: The Author

Figure 3.16: Attributions for the misclassified table's leg point using saliency.



Source: The Author

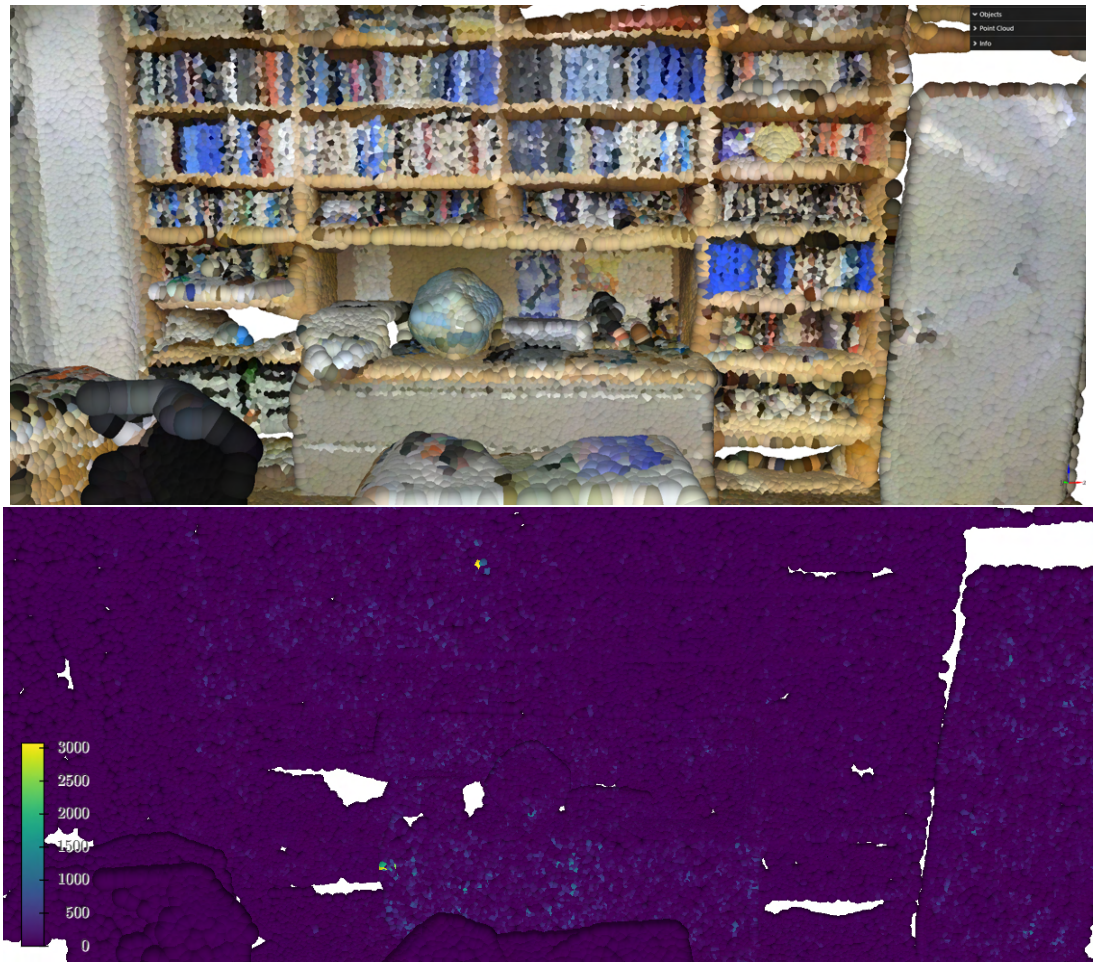
3.4.3 Integrated Gradients results

Our IG attributions were all computed using the Riemann sum as its integral approximation method, the baseline we proposed (section 2.1.2.1), and a number of steps of 50. We computed the attributions with other bigger numbers of steps, but the differences in attribute values were minimal compared to the increasing computational cost.

3.4.3.1 Conference Room

Figure 3.20 shows that the attributions summary for the table class in the conference room is slightly different when we use IG. The points belonging to the bigger table in the room do not have too much influence now. We believe that this is because we only see tables of this size in conference rooms, other scenes like lobbies and offices have smaller tables. This becomes more evident when we see high attribute values on the small side of the big table, which has the same size as the sides of other tables. The great difference

Figure 3.17: Summary of the attributions of the class *bookcase* using saliency.

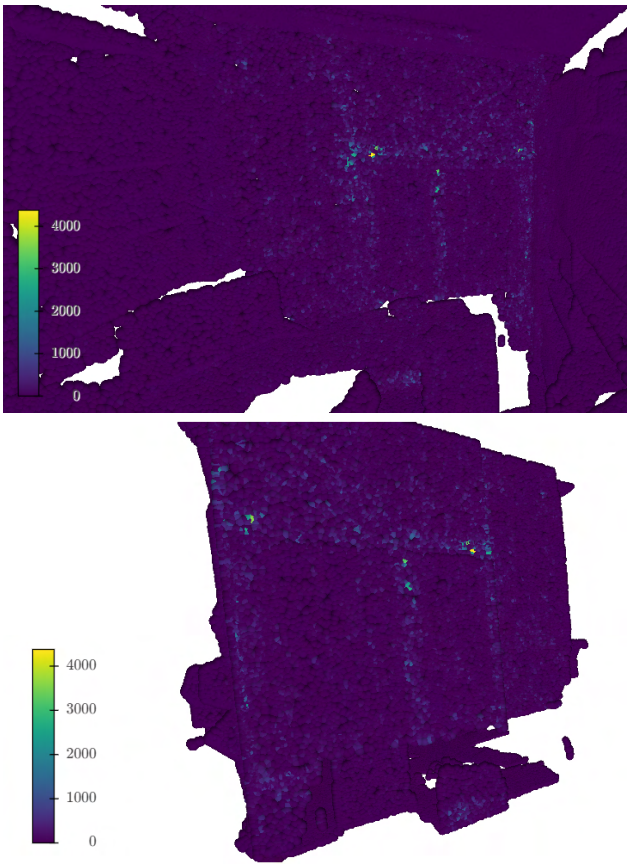


Source: The Author

that IG brought is that the influence of the wall on the table next to it is now smaller. The contribution of the wall points can be relevant at first but, as we accumulate the gradients, other points end up being more relevant.

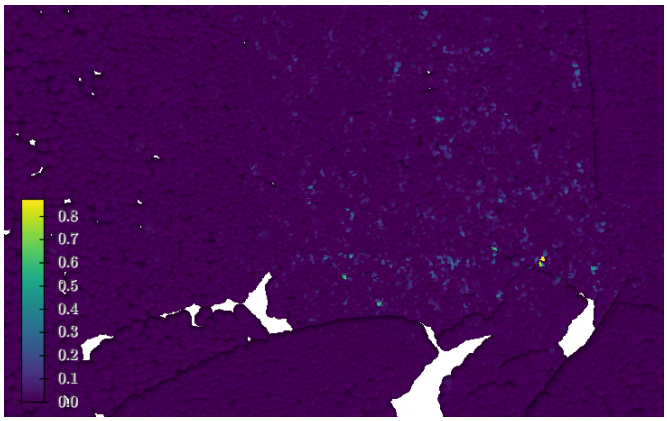
We can also see this change of relevance when we compare the attributes from IG with the ones computed from the saliency method to the chair (figure 3.21). The area of influence does not exist anymore and the local features of the chair became more relevant. Our analysis of the table's leg changed when we compared it with the saliency (figure 3.22). We notice some negative values now in the chair and positive values directly in the rest of the points classified as chairs in the table's leg, making them more relevant.

Figure 3.18: The attributions for the window at the office using saliency.



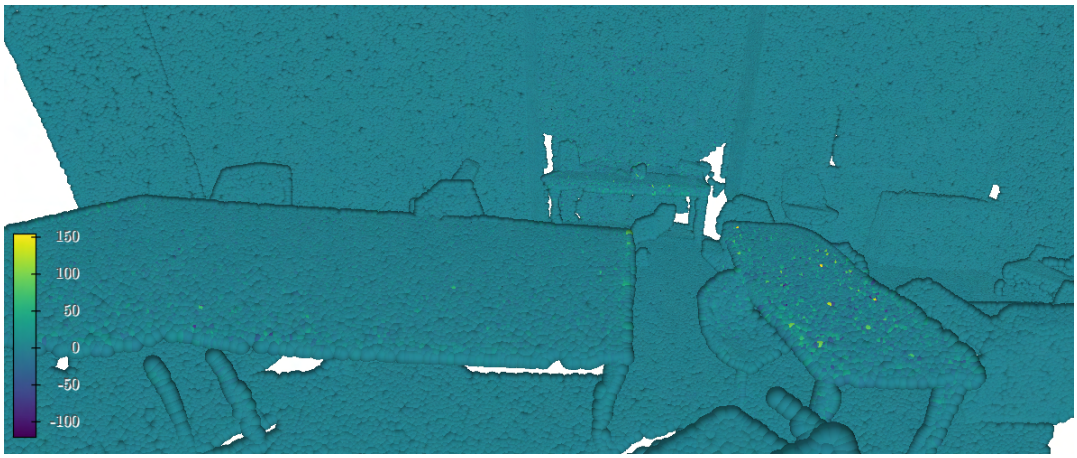
Source: The Author

Figure 3.19: Attributions of the point misclassified as a door using saliency.



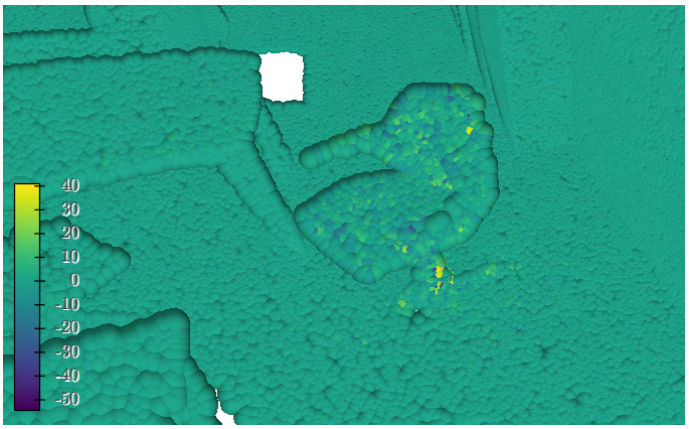
Source: The Author

Figure 3.20: Summary of the attributions of the class *table* using IG.



Source: The Author

Figure 3.21: The attributions for a specific chair at the conference room using IG.



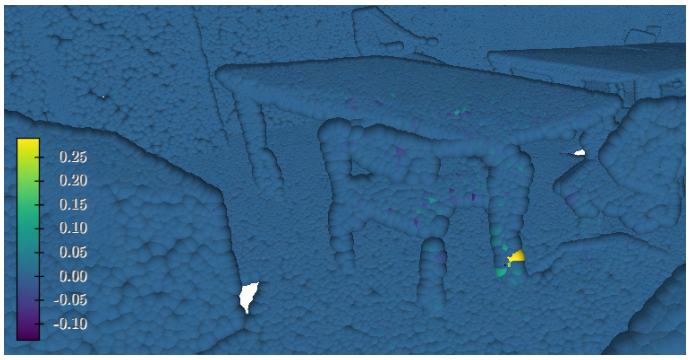
Source: The Author

3.4.3.2 Office

Figure 3.23 shows that the summary of the attributes for the bookcase class has a more equal distribution of the high attributes. The points that contribute the most now are the ones representing the books, instead of only the cabinet in the lower part. The door now does not have high attribute values, confirming to us that they do not have the importance that the other points have for the total class store and possibly revealing a lack of confidence in the prediction that was incorrect.

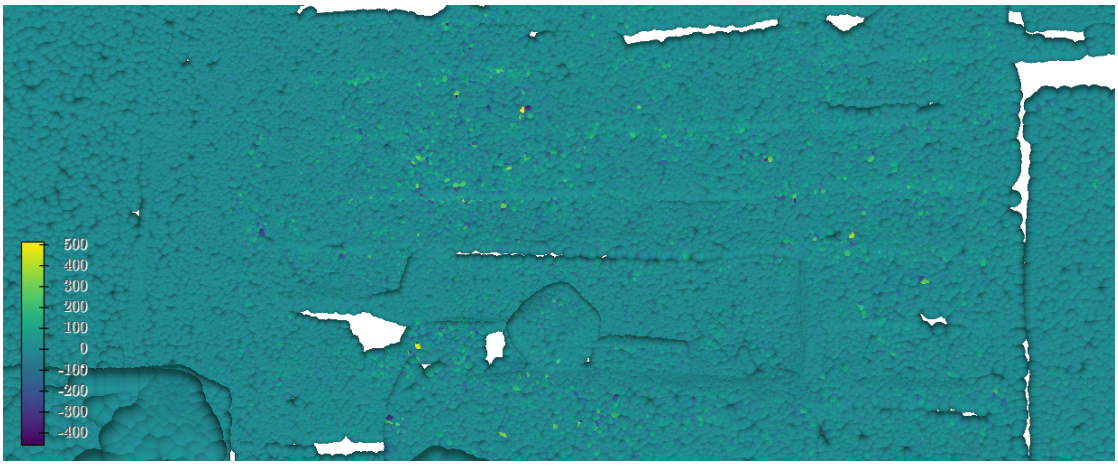
The window attributes had some changes too (figure 3.24). There was a refinement over some global features like the wall, where they became less relevant. There are now

Figure 3.22: Attributions for the misclassified table's leg point using IG.



Source: The Author

Figure 3.23: Summary of the attributions of the class *bookcase* using IG.



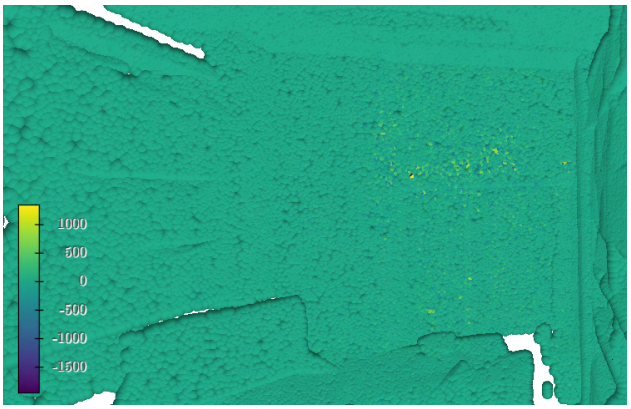
Source: The Author

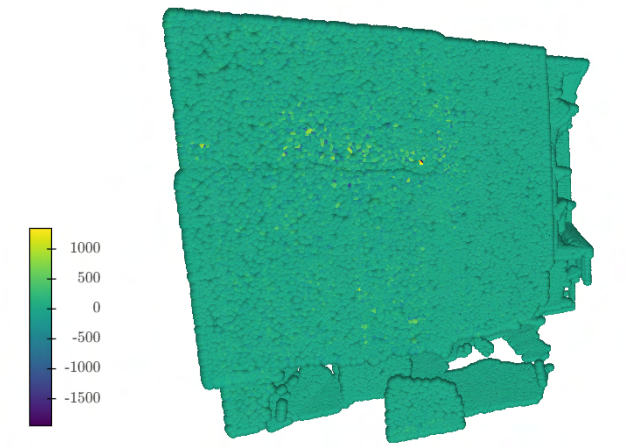
more contributions over the superior part of the window, which is common in the window structures of all scenes, but the glass area is still irrelevant. The point misclassified as a door does not have any more influence on the board (figure 3.25), only the other points classified incorrectly as a door.

3.5 Results summary

With our results, we can now answer the questions we made previously at the beginning of the chapter. Related to the mistakes, thanks to the attribute methods and visualization techniques we could understand some of the incorrect predictions. The majority of them are caused by the influence of other misclassified points in the same area

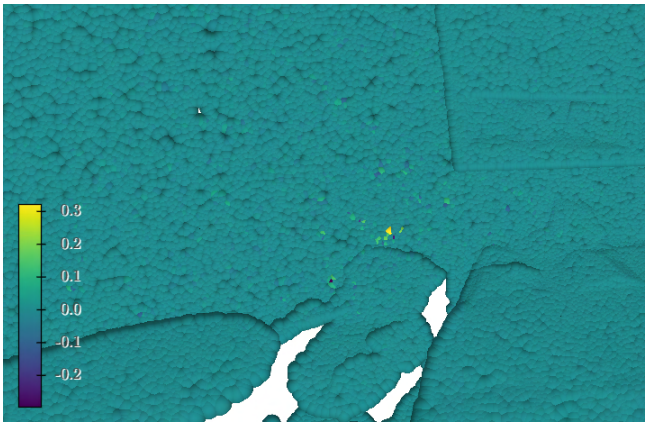
Figure 3.24: The attributions for the window at the office using IG.





Source: The Author

Figure 3.25: Attributions of the point misclassified as a door using IG.



Source: The Author

and objects close to the area misclassified, the table's leg results show a good example of this happening.

The context of the scene is in fact important to the predictions, but not as important

as the local features. We could say that each point has an area of influence, as we increase the analysis radius, the smaller are the contributions of the points at the end of this radius. Saliency attributes show to us, by the results of the chair instance, using the measurements of a real room, that the area of influence is around 1 meter. Unfortunately, we can not still identify at which scale these contributions became smaller. We identify that the receptive field of the network is large. When we use the model wrapper to analyze an entire object, almost every point in the scene has an attribute value. Points far from the area of interest have attribute values near to 0. The discrepancy between the results between Saliency and IG can also be explained by these values near 0. The accumulation executed by IG tends to ignore these points with values near 0 and only the points near to the area of interest have more importance, since their attributes values are not near 0 and the IG sum can reach bigger values of attributes.

Nubilum in fact helps us to understand better the predictions of a semantic segmentation model. The plot tools help us navigate easily through the scene. Despite the limitations of the color scales, we can identify and differentiate high attributes from low attributes, and the highlight of areas for the analysis of correct predictions, using the attribute values, makes sense. Unfortunately, we have to deal with the tradeoff between information and performance between the plot tools, and sometimes we can not understand more subtle differences among the points. As we saw with the misclassification of the door point, not all the mistakes can not be interpretable only by the attributions, therefore more elaborated methods are needed for full comprehension.

4 CONCLUSION AND FUTURE WORKS

This work presented Nubilum, a Captum's extension library dedicated to explaining semantic and instance segmentation models for point cloud data. We described the attribute methods we use from Captum, saliency, and integrated gradients. We developed three model wrappers that can be used in these attribute methods to analyze different magnitudes of a scene prediction and we developed 3D visualization methods for a point cloud, its segmentation, and explainability of its features. We also used Nubilum to analyze SoftGroup predictions of two scenes from the S3DIS dataset. The attributions computed allied to the visualization plots allow us to have a good understanding of different scene features and how they are important for segmentation model predictions.

This is a preliminar version of Nubilum and we are still improving it by adding new features and fixing its problems. We believe that the main problem in Nubilum rests on its plot tools. Both are limited in performance, lack of information, and shading options. Plotly and K3D also allow options to support two or more visualizations in the same plot, but the rendering of multiple instances of point clouds in the same plot is impractical, Plotly already has performance problems with just one instance and K3D can not use more than one color mapping in one plot. Also, the necessity to use two plot tools instead of just one doubles our dependency on generic tools. For future work, it would be interesting to develop a plot tool that shares the benefits from both Plotly and K3D, focused exclusively on point cloud data, and that is interactive enough for real-time point or area selection.

We understand that the magnitude values of the attributes and their distribution mapping over a color scale can be problematic for some meticulous analysis of the scene. For example, a lot of scenes contain attributes computed with values over a hundred, leaving us with a difficult view of smaller but relevant attributes. A future study focused on the normalization or equalization of attributes for visualizations as well as a coverage of color scales that can optimize our understanding of a scene would be welcome.

Sturmfels, Lundberg and Lee (2020) explores the impact of different baselines for attribute methods like the IG in images. Baselines with just one color, like ours, tend to not highlight areas that share the same color of the baseline (remember that equation 2.1 has a factor $(x_i - x'_i)$ multiplying their integral. For a color feature, this factor could result to zero). Sturmfels, Lundberg and Lee (2020) proposes another baseline such as blurred images and multi-use of a Gaussian baseline to correct this problem. A similar study,

for point cloud baselines should be done in the future to achieve good baselines for point cloud analysis.

We believe that Nubilum is a great start for developers who chase interpretability in semantic segmentation over point cloud data. The implementation of more post-hoc methods would bring more robustness to our library and would make it more complete. We are experiencing just the beginning of the study of new methodologies and techniques for semantic segmentation on point cloud, and we envision Nubilum can have a big participation in shaping the future of this rapidly evolving field and unlocking new insights and applications.

REFERENCES

ALI, S. et al. Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence. **Information Fusion**, v. 99, p. 101805, 2023. ISSN 1566-2535. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1566253523001148>.

ANTON, H.; BIVENS, I.; DAVIS, S. **Calculus, 10th Edition**. Wiley, 2012. ISBN 9781118404003. Available from Internet: <https://books.google.com.br/books?id=xSkcAAAAQBAJ>.

ARMENI, I. et al. **Joint 2D-3D-Semantic Data for Indoor Scene Understanding**. 2017.

ARMENI, I. et al. 3d semantic parsing of large-scale indoor spaces. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016.

AUCLAIR, A.; COHEN, L.; VINCENT, N. A robust approach for 3d cars reconstruction. In: ERSBØLL, B. K.; PEDERSEN, K. S. (Ed.). **Image Analysis**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 183–192. ISBN 978-3-540-73040-8.

ENGELMANN, F. et al. Know what your neighbors do: 3d semantic segmentation of point clouds. In: LEAL-TAIXÉ, L.; ROTH, S. (Ed.). **Computer Vision – ECCV 2018 Workshops**. Cham: Springer International Publishing, 2019. p. 395–409. ISBN 978-3-030-11015-4.

GRAHAM, B.; ENGELCKE, M.; MAATEN, L. van der. **3D Semantic Segmentation with Submanifold Sparse Convolutional Networks**. 2017.

GUO, Y. et al. Deep learning for 3d point clouds: A survey. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 43, n. 12, p. 4338–4364, 2021.

HAFIZ, A. M.; BHAT, G. M. A survey on instance segmentation: state of the art. **International Journal of Multimedia Information Retrieval**, v. 9, n. 3, p. 171–189, Sep 2020. ISSN 2192-662X. Available from Internet: <https://doi.org/10.1007/s13735-020-00195-x>.

HEIDE, N. F. et al. X3seg: Model-agnostic explanations for the semantic segmentation of 3d point clouds with prototypes and criticism. In: **2021 IEEE International Conference on Image Processing (ICIP)**. [S.l.: s.n.], 2021. p. 3687–3691.

KOKHLIKYAN, N. et al. **Captum: A unified and generic model interpretability library for PyTorch**. 2020.

LAVADO, D. et al. **Low-Resource White-Box Semantic Segmentation of Supporting Towers on 3D Point Clouds via Signature Shape Identification**. 2023.

LEE, M. S.; YANG, S. W.; HAN, S. W. Gaia: Graphical information gain based attention network for weakly supervised point cloud semantic segmentation. In: **Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)**. [S.l.: s.n.], 2023. p. 582–591.

LONG, J.; SHELHAMER, E.; DARRELL, T. **Fully Convolutional Networks for Semantic Segmentation**. 2015.

MATRONE, F. et al. Bubblex: An explainable deep learning framework for point-cloud classification. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 15, p. 6571–6587, 2022.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: **Advances in Neural Information Processing Systems 32**. Curran Associates, Inc., 2019. p. 8024–8035. Available from Internet: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

SCHWEGLER, M.; MüLLER, C.; REITERER, A. Integrated gradients for feature assessment in point cloud-based data sets. **Algorithms**, v. 16, n. 7, 2023. ISSN 1999-4893. Available from Internet: <https://www.mdpi.com/1999-4893/16/7/316>.

SIMONYAN, K.; VEDALDI, A.; ZISSERMAN, A. **Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps**. 2014.

STURMFELS, P.; LUNDBERG, S.; LEE, S.-I. Visualizing the impact of feature attribution baselines. **Distill**, 2020. Https://distill.pub/2020/attribution-baselines.

SUNDARARAJAN, M.; TALY, A.; YAN, Q. **Axiomatic Attribution for Deep Networks**. 2017.

VU, T. et al. **SoftGroup for 3D Instance Segmentation on Point Clouds**. 2022.

APPENDIX A — NUBILUM DOCUMENTATION

A.1 nubilum.attr submodule

This submodule contains all extensions for Captum's attribute methods.

A.1.1 Class NubilumSaliency

```
class NubilumSaliency(Saliency)
```

Saliency implementation dedicated to the point cloud.

A.1.1.1 Constructor

```
NubilumSaliency(forward_func: Callable[..., Any])
```

Parameters

- **forward_func (Callable):** The forward function of the model or any modification of it.

A.1.1.2 Method: attribute

```
attribute(self, inputs: TensorOrTupleOfTensorsGeneric,
          target: TargetType = None,
          abs: bool = True,
          additional_forward_args: Any = None
         ) -> TensorOrTupleOfTensorsGeneric
```

Calls the attribute method from Captum Saliency.

Parameters

- **inputs (TensorOrTupleOfTensorsGeneric):** The input tensors for which saliency is computed.
- **target (TargetType, optional):** The target index in which the attributes will be computed. Defaults to None.

- **abs (bool, optional):** Returns the absolute value of gradients if set to True, otherwise returns the (signed) gradients if False. Defaults to False.
- **additional_forward_args (Any, optional):** Additional arguments to pass to the forward function. Defaults to None.

Returns

- The computed attribution tensors.

A.1.2 Class NubilumIntegratedGradients

```
class NubilumIntegratedGradients(IntegratedGradients)
```

Integrated Gradients implementation dedicated to the point cloud.

A.1.2.1 Constructor

```
NubilumIntegratedGradients(forward_func: Callable[..., Any])
```

Parameters

- **forward_func (Callable):** The forward function of the model or any modification of it.

A.1.2.2 Method: attribute

```
attribute(self,
          inputs: TensorOrTupleOfTensorsGeneric,
          baselines: TensorOrTupleOfTensorsGeneric = None,
          target: TargetType = None,
          additional_forward_args: Any = None,
          n_steps: int = 50,
          method: str = 'riemann_middle',
          internal_batch_size: Union[None, int] = None,
          return_convergence_delta: bool = False
        ) -> TensorOrTupleOfTensorsGeneric:
```

Calls the attribute method from Captum Integrated Gradients.

Parameters

- **inputs (TensorOrTupleOfTensorsGeneric):** The input tensors for which integrated gradients are computed.
- **baselines (TensorOrTupleOfTensorsGeneric, optional):** Baseline tensors. Baseline tensor shapes must be equal to the input tensors. Defaults to None.
- **target (TargetType, optional):** The target index in which the attributes will be computed. Defaults to None.
- **additional_forward_args (Any, optional):** Additional arguments to pass to the forward function. Defaults to None.
- **n_steps (int, optional):** Number of steps for the integral approximation. Defaults to 50.
- **method (str, optional):** Method to be used for the integral approximation. Defaults to 'riemann_middle'.
- **internal_batch_size (Union[None, int], optional):** Size of the batch. Defaults to None.
- **return_convergence_delta (bool, optional):** Indicates whether to return convergence delta or not. Defaults to False.

Returns

- The computed attribution tensors.

A.2 nubilum.forward submodule

This submodule contains all three model wrappers developed to be used in the attribute methods.

A.2.1 Class InstanceWrappedModel

```
class InstanceWrappedModel(torch.nn.Module)
```

This model wrapper is dedicated to being used in instance segmentation models. The forward function receives as additional forward arguments the ground truth instance labels and one specific instance label. Any attribute function executed using this wrapped model will calculate attributes over the points that have its ground truth instance label equal to the label of interest. The idea is to understand how the model classifies the individual objects in the scene, checking which points contribute to the result and how the group of points of the instance of interest is influenced by other points from other objects in the scene.

A.2.1.1 Constructor

```
InstanceWrappedModel(model: Any)
```

Parameters

- **model (Any):** A PyTorch's neural network model

A.2.1.2 Method: forward

```
forward(self, point_coords: Tensor, point_colors: Tensor,
        instance_labels: Tensor, label_of_interest: int,
        output_scores_key_name: str = None) -> Tensor:
```

Calculates only the scores for the points whose instance labels is the label of interest. It returns a tensor with the scores of the instance points for each class. Each index of all tensors must identify the exact same point.

Parameters

- **point_coords (Tensor):** Model's Input Feature. Tensor containing the points coordinates.
- **point_colors (Tensor):** Model's Input Feature. Tensor containing the points colors.
- **instance_labels (Tensor):** Additional forward argument. Tensor containing the GT points instance labels.
- **label_of_interest (int):** Additional forward argument. The instance label is to be verified.

– **output_scores_key_name (str, optional):** Additional forward argument. Indicates the key for the semantic prediction scores if the output is a dictionary. Defaults to None if the Output is already a tensor with the semantic predictions.

Returns

– A tensor with the scores of the instance points for each class.

A.2.2 Class PointWrappedModel

```
class PointWrappedModel(torch.nn.Module)
```

Model wrapper dedicated to finding attributes related to a unique point. It permits us to understand what other points influence the classification of a point of interest.

A.2.2.1 Constructor

```
PointWrappedModel(model: Any)
```

Parameters

– **model (Any):** A PyTorch's neural network model

A.2.2.2 Method: forward

```
def forward(self, point_coords: Tensor, point_colors: Tensor,
            point_of_interest: int,
            output_scores_key_name: str = None) -> Tensor:
```

Calculates the scores of only one point, for each class.

Parameters

– **point_coords (Tensor):** Model's Input Feature. Tensor containing the points coordinates.

– **point_colors (Tensor):** Model's Input Feature. Tensor containing the points colors.

– **point_of_interest (int):** Additional forward argument. The point is to be verified.

– **output_scores_key_name (str, optional):** Additional forward argument. Indicates the key for the semantic prediction scores if the output is a dict. Defaults to None if the Output is already a tensor with the semantic predictions.

Returns

– A tensor with the scores of the point of interest for each class.

A.2.3 Class SummarizedWrappedModel

```
class SummarizedWrappedModel(torch.nn.Module)
```

Model wrapper dedicated to understanding the prediction of the entire point cloud. Since attributions methods are made for classification tasks and not segmentation tasks, the forward function of this model wrapper tries to summarize the classifications of all points by summing the scores of the final prediction of each point. For more details, check https://captum.ai/tutorials/Segmentation_Interpret#Interpreting-with-Captum

A.2.3.1 Constructor

```
SummarizedWrappedModel(model: Any)
```

Parameters

– **model (Any):** A PyTorch's neural network model

A.2.3.2 Method: forward

```
def forward(self, point_coords: Tensor, point_colors: Tensor,
            output_scores_key_name: str = None) -> Tensor:
```

Sums the scores of each class. The score of the ith class of a point j is only used in the sum if, and only if, the class i was the class predicted for the point j.

Parameters

– **point_coords (Tensor):** Model's Input Feature. Tensor containing the points coordinates.

- **point_colors (Tensor):** Model's Input Feature. Tensor containing the points colors.
- **output_scores_key_name (str, optional):** Additional forward argument. Indicates the key for the semantic prediction scores if the output is a dict. Defaults to None if the Output is already a tensor with the semantic predictions.

Returns

- A tensor with a sum of all predicted scores for each class.

A.3 nubilum.utils submodule

This submodule contains all the visualization methods as well as other functions such as the baseline creation and the sum of attributes.

A.3.1 Function: check_tensor_shapes

```
check_tensor_shapes(tensors) -> bool:
```

Checks if the tensors inside an iterable object, like a list or tuple, have the same shape

Parameters

- **tensors (Iterable):** Iterable object containing tensors.

Returns

- True if tensors have the same shape, False otherwise.

A.3.2 Function: show_poi

```
show_poi(poi_index: int, coords: Tensor) -> None:
```

Shows the point cloud with the point of interest in evidence. It uses K3D with a coolwarm color scale.

Parameters

- **poi_index (int):** Index of the point of interest
- **coords (Tensor):** Coordinates of each point.

A.3.3 Function: sum_point_attributes

```
sum_point_attributes(attributes: TensorOrTupleOfTensorsGeneric,
                     target_dim: int = -1) -> Tensor:
```

Performs an element-wise summation over the attributes, followed by a sum of the elements in the target dimension in the resulting tensor. Useful to aggregate attribution for any kind of point features.

Parameters

- **attributes (TensorOrTupleOfTensorsGeneric):** Tuple of tensors that describes each point attributes. The tensors must have the same shape.
- **target_dim (int, optional):** Target dimension where the last sum will occur. Defaults to -1, the last dimension.

Returns

- A tensor containing the sum of all tensors element-wise and with the last dimension added.

A.3.4 Function: create_baseline_point_cloud

```
create_baseline_point_cloud(input_coords: Tensor)
                            -> Tuple[Tensor]:
```

Baseline based on a cubic uniform point distribution. The colors returned will be all black. The volume used uses the same min and max bounds of the coordinates. If we can not perfectly divide the number of points in the rectangular volume, we add the remaining points randomly through the volume.

Parameters

– **input_coords (Tensor):** Coordinates of the input in a size (N, 3).

Returns

– Tuple containing the coordinates of the baseline points and their colors.

A.3.5 Function: show_point_cloud

```
show_point_cloud(coords: Tensor, colors: Tensor,
                 size: float = 0.1) -> None:
```

Plots the Point Cloud using K3D.

Parameters

– **coords (Tensor):** Coordinates of the points, in the shape (N, 3).

– **colors (Tensor):** Colors of the points, in the shape (N, 3). It assumes that the colors are in the RGB format with interval [0, 255].

– **size (float, optional):** Points size in the plot. Defaults to 0.1.

A.3.6 Function: show_point_cloud_classification_k3d

```
show_point_cloud_classification_k3d(coords: Tensor,
                                    classifications: Tensor,
                                    size: float = 0.1) -> None:
```

Plots the classification of each point using K3D. It is not capable of holding extra information, but it has a better performance than show_point_cloud_classification_plotly. Recommended to use when Plotly's performance spoils the 3D interaction.

Parameters

– **coords (Tensor):** Coordinates of the points, in the shape (N, 3).

A.3.7 Function: show_point_cloud_classification_plotly

```
show_point_cloud_classification_plotly(coords: Tensor,
                                       classifications: Tensor,
                                       instance_labels: Tensor = None,
                                       classes_dict: dict = None,
                                       size: float = 0.1) -> None:
```

Plot the classification of each point using Plotly. It can hold extra information such as instance labels and prediction meanings.

Parameters

- **coords (Tensor):** Coordinates of the points, in the shape (N, 3).
- **classifications (Tensor):** The predictions indices for each point
- **instance_labels (Tensor, optional):** Object instance labels for each point. The order of the points must be the same as the coordinates and classifications. Defaults to None.
- **classes_dict (dict, optional):** Dictionary containing the meaning of each prediction index. Defaults to None.
- **size (float, optional):** Points sizes in the plot. Defaults to 0.1.

A.3.8 Function: explain_k3d

```
explain_k3d(attributes: Tensor, coords: Tensor,
            attribute_name = None) -> None:
```

Plots the point cloud with its attribute values using K3D. It does not offer the exact value of the attributes but its performance and scene understanding are better than Plotly's explanation.

Parameters

- **attributes (TensorOrTupleOfTensorsGeneric):** Attributes for each point.

- **coords (Tensor):** Coordinates of each point.
- **attribute_name (str, optional):** Name of the point data in the plot. Defaults to None.

A.3.9 Function: explain_plotly

```
explain_plotly(attributes: Tensor, coords: Tensor,
            template_name: str = 'simple_white') -> None:
```

Plots the point cloud with its attribute values using Plotly. Useful for a thorough analysis of the point's attribute values, but it has a poor interaction and scene understanding thanks to Plotly's point rendering.

Parameters

- **attributes (TensorOrTupleOfTensorsGeneric):** Attributes for each point.
- **coords (Tensor):** Coordinates of each point.
- **template_name (str, optional):** The template style to be used for the plot. Defaults to 'simple_white'.