SCIPoC: Semantic Classification of Indoor Point Cloud

A study into the possibilities of classifying indoor point cloud using a Deep Learning approach

Mels Smit
m.smit-6@student.tudelft.nl
Delft Univertisy of Technology

Xiaoai Li
x.li-49@student.tudelft.nl
Delft Univertisy of Technology

Zhaiyu Chen
z.chen-26@student.tudelft.nl
Delft University of Technology

Mihai-Alexandru Erbasu
m.a.erbasu@student.tudelft.nl
Delft Univertisy of Technology

Yustisi Ardhitasari Lumban Gaol
YustisiArdhitasariLumbanGaol@student.tudelft.nl
Delft University of Technology

Abstract

With the constantly evolving range of applications for technology the quality and amount of data constantly increases as well. In this growing data environment, there is a constant search to provide more value to all data that is available for as little effort as possible. Our research tries to add such additional value by diving into the concept of classifying point cloud by using deep learning, specifically in the indoor environment. This is done by first doing a neural network comparison and then doing a case study. In the neural network comparison, a look is taken into which of the neural networks that are capable of working with point clouds is best suited for our experiments in the indoor scene, based on the training speed, accuracy, ease of use concerning training on external datasets and setting up the network and space efficiency. After the comparison, we chose to continue with the PointCNN network during the case study. The case study is performed on data the NS (Nederlandse Spoorwegen) provided to us and all test results we got from our experiments can be visualized using the web application we developed along with this project. The purpose of the case study is to add extra value to the indoor LiDAR point cloud the NS has captured from Amersfoort Station by using deep learning to automatically classify assets present in their data. The value is in purposes, such as asset management, where the data does not need possibly hundreds of man-hours to be labelled. This saves a lot of time and also money each time a scan is made. In the case study we found through 4 different experiments that unbalanced data makes for bad results, but when a scene is labelled correctly very good results can be found in a local scene.

Keywords: Point Cloud, Deep Learning, Machine Learning, Neural Network, Indoor Environment, Data Labeling, Semantic Classification, Visualization, Comparison, Case Study, Web Application

TU Delft Geomatics Synthesis Project (2020).

1 Introduction

This paper was written by a team of Geomatics students from the Faculty of Architecture and the Built Environment from the Delft University of Technology as part of a Synthesis Project. This was developed with the intent of allowing first-year students to apply the knowledge accumulated throughout the year in a project which has real-world applications. Moreover, deep neural networks are still not part of the Geomatics curriculum so in this case, we extended our knowledge, skills, and insight far upon what was offered in Geomatics. Finally, this paper is expected to give an overview of those knowledges and possible applications regarding deep neural network and point clouds.

1.1 Problem Definition

Over the last few years, technology has been evolving constantly. New techniques for all kinds of applications kept popping up and with the constant growth of computational power also ideas from long times ago have resurfaced to finally show their worth completely. With this constant growth of information, the collection of data has also increased as well as the detail with which this has been captured. This gave way to techniques that can use this data to learn from input and can apply this knowledge for the classification of other data. This principle also called machine learning has been tested in many environments and has proven to get some very great results. This paper strives to test one of the forms of machine learning called deep learning in an environment which is still somewhat less unexplored, namely the indoor scene. Compared with the outdoor scene, the indoor scene is more complex to parse since it is more costumed and the variety of the indoor features surpasses that of the outdoors. Nevertheless, it does not imply that the indoor scene is inferior than the outdoor scene. To the contrary, the indoor scene is closer to the human habitat which is worthy of exploring equally, if not more. It will also focus on a point cloud data format which is currently less explored than the traditional image-based machine learning. To do this we will test multiple available neural networks who are capable of deep learning based on point clouds, to see what the potential is for this approach. This potential will be tested in a case study in which the neural network(s) will classify unlabeled data and say where certain objects can be found in the captured point cloud data. This can be of great importance for our clients Esri Nederland and the NS (Nederlandse Spoorwegen) as automatically classifying features in data adds a lot of value to the data that is already being captured in the indoor environment. The NS can, for instance, save hundreds of hours of manual labelling of data, if not more, after scans of stations have been made [Kudinov, 2019]. These labelled scans can then be used for all sorts of applications like for instance asset management, where the data can be used to see where objects have been positioned, how many of them were there and to identify if new assets need to be ordered, or directly for exploitative visualization and analysis [Poux, 2019].

To make sure the research we performed was focused on one consistent subject we formulated the following research question:

Which of the available Deep Neural Networks is best suited to syntactically classify individual points in a point cloud (also called semantic segmentation), which captures an indoor environment and which training approaches are best suited to improve performance of the networks?

We took this research question as it focuses on the most core principle on what we want to do which makes our results suitable also for different applications than the case study we performed in this paper. To make sure this research question could be properly answered we also defined some partial research questions as shown below:

- What is the performance for the currently available Neural Networks?
- How does the selection of training features affect the performance of the network?
- What are the characteristics of the indoor environment?
- What are the characteristics of point clouds?

1.2 Development of applications based on deep learning

Development of applications based on deep learning Deep learning is a breakthrough in the field of artificial intelligence with the extraordinary performance in a wide range of applications such as image recondition and self-driving vehicles [Wang et al., 2018].

Esri, as an international supplier of geographic information system (GIS) applications, is investing heavily in the intersection of deep learning and GIS. ArcGIS is the representative software of Esri which allows ford spatial data creation, management and analysis. Machine learning has been deployed in ArcGIS to solve three main problems: classification, clustering, and prediction [Singh, 2019]. Recently, Esri has collaborated with NVIDIA & Cisco Systems on running large GPU workloads in ArcGIS Pro [Brown, 2019]. It is very important and useful for applications based on deep learning because GPU allows for a faster artificial neural network by implementing the matrix multiplication of a neural network [Oh and Jung, 2004]. Currently, ArcGIS Pro allows for data preparation (labelling) for deep learning and then deploying trained neural networks for feature classification. However, the applications related to deep learning are limited to 2D images in the existing products of Esri. In ArcGIS Pro, deep learning module is part of Image Analyst extension where users can label datasets, train neural networks and get predicted results.

In this paper, we propose a web application of deep learning for semantic classification of indoor point cloud. The main functions are divided into file uploading/downloading, segmentation and visualization. Like ArcGIS Pro, the web application integrates the deep learning to fix geographical problems.

1.3 Organisation of the paper

In section 2, we introduce deep learning with an emphasis on the neural networks, as the background knowledge of the paper. Section 3 lists the related work on point cloud features, feature learning on 3D data, and feature learning from irregular domains. In section 4, several deep neural networks are discussed in detail which are to different degree used in this project. Based on the evaluation of these, we select the most suitable pipeline for further experiments in the case study discussed in section 5. A RESTful web application is implemented, the detail of which is addressed in section 6. Section 7 draws the conclusion and section 8 proposes the future research.

2 Theoretical Background

This section presents an introduction to deep learning and fundamentals of neural networks. This theoretical background is required for a better comprehension of the rest of this paper.

2.1 Introduction to Deep Learning

Deep learning is a part of machine learning methods which can select and extract features from the data automatically. There are various algorithms in machine learning, e.g., Support Vector Machines [Chang and Lin, 2011], Random Forest [Liaw et al., 2002], neural networks, etc. In general, machine learning is divided into supervised and unsupervised manners. Both are algorithms that learn from the data, but one requires labelled data for training (supervised) and the other one does not have a labelled input. As for deep learning, it refers to a Deep Neural Network that has deep hidden layers.

Deep learning algorithms, such as recurrent neural network and convolutional neural network, have been applied to diverse fields including computer vision, speech and audio recognition, bioinformatics, and geospatial. In the geospatial sector, many researcher and organisation investigate and apply this algorithm for automatic object detection or to create a digital twin [Zhu et al., 2017]. Those applications are important for efficient local government operations, urban planning and design, utility or asset management, and safety.

2.2 Fundamentals of Neural Networks

As the smallest unit of information processing in the nerve system, neurons can transmit and aggregate information. In 1943, Warren McCulloch and Walter Pitts proposed the McCulloch-Pitts model (M-P model) to simulate biological neurons [McCulloch and Pitts, 1943], whose structure is shown in Figure 1.

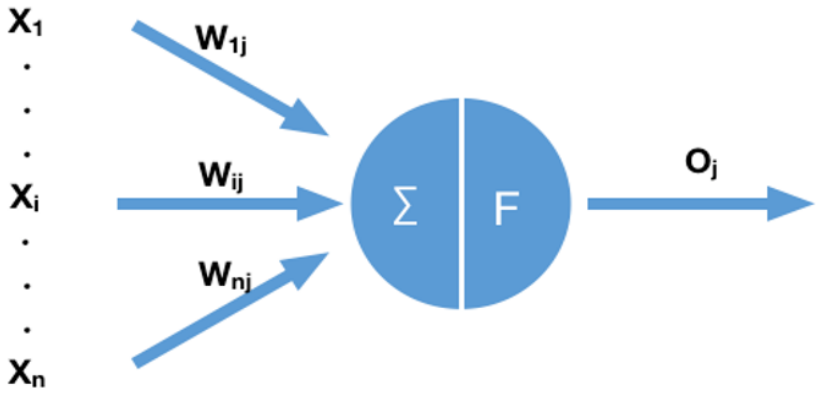


Figure 1: Neural structure in McCulloch-Pitts model

There are many similarities between the structure of the M-P model and actual biological neurons. In Figure 1, $x_{1,2,\ldots,n}$ represents the input and w_{ij} represents the weight. The Σ function is to integrate and accumulate all input signals, and F is the activation function. If the model output value O_j is positive, the neuron is activated, otherwise the neuron is resting. The neuron represented by the M-P model uses a certain weight to accumulate the input signal, and then outputs the current state of the neuron. Note besides w_{ij}, there is another parameter called the bias in the model denoted as θ_j. This model can be expressed by

$$O_j = F(\sum_{n}^{i=1} w_{ij} x_i(j) - \theta_j) \tag{1}$$

In 1974, Werbos first proposed the multi-layer feedforward neuron network error backpropagation theory [Werbos, 1974]. Later, Rumelhart and McClelland developed and perfected this theory [McClelland et al., 1986]. The backpropagation algorithm splits the training process of the neural network into two stages: the forward propagation stage and the backpropagation stage. In forward propagation, the signal starts at the input layer and reaches the output layer via the hidden layer. When there is a difference between the output value and the true value of the label, the error is reversed from the output layer to the middle hidden layer, and the weight and bias parameters of each layer of neurons are updated layer by layer. The back-propagation model intuitively describes the problem of iterative optimization of multi-functions in artificial neural network, that is, to minimize the corresponding loss function (loss function) as the optimization goal, iterative calculation through gradient descent algorithm, adjust the neural network The weights and bias parameters of the elements enable the network to spontaneously "learn" the optimal mapping between input and output.

Convolutional neuron network is a kind of feed-forward neural network. It was first proposed by Hubel and Wiesel in the study of animal visual cortex electrophysiology [Hubel and Wiesel, 1959]. A Convolutional neural network is usually composed of an input layer, convolutional layer, pooling layer and fully connected layer. Unlike general neural networks, neurons in convolutional neuron networks often only have local connection structures, that is, local receptive fields, which means that convolutional neuron networks can use shared weights to reduce the number of parameters, thereby reducing the complexity of the network.

Since its introduction, convolutional neuron networks have been mainly used in image parsing tasks such as image recognition. In 1989, Lecun et al. used a backpropagation algorithm to implement a convolutional neural network for handwritten digit recognition [LeCun et al., 1989]. The research then was applied to the handwritten digit recognition of US postal codes, which greatly improved the efficiency of the postal system, and fully proved the adaptability of the convolutional neuron network in image recognition. After the improvement, the LeNet-5 model has greatly improved the efficiency of character recognition [LeCun et al., 1998], becoming a classic model of convolutional neuron network. Alexnet proposed by Krizhevsky et al. [2012] achieved first place with a crushing advantage in the 2012 ILSVRC competition, and the accuracy rate was 10.9% higher than the second-place algorithm, which proved that deepening the network architecture level can be achieved to a certain extent higher recognition accuracy. In these models, the size of the input image to the convolutional neuron network needs to be fixed, which may cause image distortion and distortion. In response to this limitation, He et al. proposed a spatial pyramid pooling structure that enables the network to adapt to image input of different sizes [He et al., 2015].

3 Related Work

Point Cloud Features Point cloud data is inherently embedded in 3D space and has rich semantic representations [Poux, 2019]. Point cloud features should reflect certain statistical properties that are invariant to certain transformations. Generally the features can be divided into two categories: the intrinsic features [Bronstein and Kokkinos, 2010, Sun et al., 2009, Aubry et al., 2011] and the extrinsic features [Rusu et al., 2009, Ling and Jacobs, 2007, Rusu et al., 2008]. These handcrafted features, if fully exploited for certain types of point cloud data, could possibly fit the task. However, being explicitly designed with human's understanding in mind, these features are limited in certain simple tasks and thus lack the capability to generalise to complex scenarios.

Feature Learning on 3D Data Compared with handcrafted features, the features implicitly learnt by deep neural networks are more robust. Besides the point cloud data, many other types of 3D data can be handled by the neural networks, including voxels and meshes. Volumetric CNN's apply 3D convolutions on the voxelised shapes, instead of processing the raw 3D data which are often unordered and anisotropically distributed and unsuitable for a typical CNN to handle [Wu et al., 2015, Maturana and Scherer, 2015, Qi et al., 2016]. However, these pipelines are constrained by its resolution due to the sparsity of the voxel representations. Multi-view CNNs are another type of workaround which render 3D point cloud or shapes into 2D images and then apply 2D convolution operations to classify them [Su et al., 2015, Qi et al., 2016]. Though these workarounds to some extent can tackle the many-to-one mapping problems such as the classification of a whole scene from the point cloud where only 1 label is required for the point set, they can not generalise to the many-to-many mapping problem such as the semantic segmentation where each point in the set needs to be assigned a label [Qi et al., 2017a]. Moreover, spectral CNN's use of spectral CNNs on manifold meshes while it is currently not feasible to apply the spectral CNNs on point cloud data [Masci et al., 2015].

Feature Learning from Irregular Domains Point cloud is a type of unordered, anisotropically distributed data from irregular domains. Recently there are researches on learning features from these domains. PointNet [Qi et al., 2017a] and Deep Sets [Zaheer et al., 2017] achieve order invariance of input points using a symmetric function over inputs. PointNet++ [Qi et al., 2017b] and SO-Net [Li et al., 2018a] improve the feature learning process by applying PointNet hierarchically to capture the local structures among the point set. PointCNN [Li et al., 2018b] simultaneously learns the weighting of the input features associated with the point and the permutation of the points by applying an $\mathcal{X}$-transformation from the input points. Moreover, with the fact that graph structures are proven to be successful in deep learning [Battaglia et al., 2018, Zhou et al., 2018, Tran, 2018], deep neural networks on point cloud analysis also use graph structures to facilitate the feature learning [Landrieu and Simonovsky, 2018, Wang et al., 2019b]. With this project, we employ several typical neural networks for feature learning from irregular domains.

4 Deep Neural Networks on Point Cloud Classification

As addressed in section 3, feature learning from irregular domains is capable of capturing the anisotropic features for analysing point cloud data. Therefore, in this section, we further discuss a few relevant pipelines namely PointNet, PointNet++, Superpoint Graphs, PointCNN, DGCNN, and VoteNet which are to different degree used in this project. An evaluation of these methods is offered, based on which we select the most suitable pipeline for further experiments in section 5.

4.1 PointNet and PointNet++

PointNet exploits features by extracting a global feature from all point cloud data, which substantially neglect the local embedding of the point features [Qi et al., 2017a]. Nevertheless, it is considered as the fundamental work of point cloud analysis with deep learning. Inspired by CNN's' layer-by-layer abstraction of local features, Qi et al. [2017b] propose PointNet++, which can extract local features at different scales and obtain deep features through a multi-layer network structure. Figure 2 shows the architecture of the PointNet and Figure 3 shows the architecture of the PointNet++.

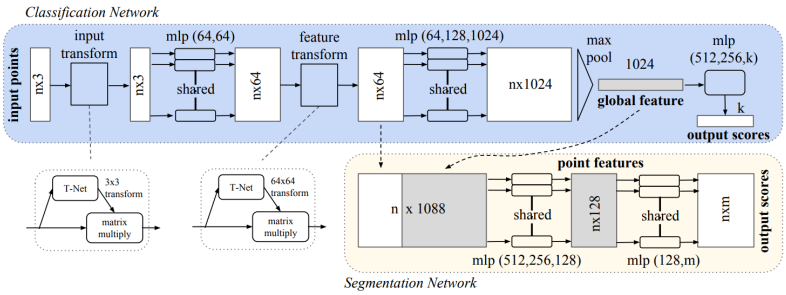


Figure 2: PointNet Architecture. Taken from Qi et al. [2017a].

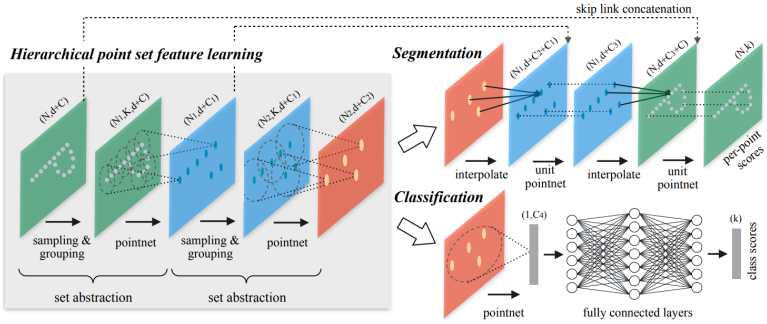


Figure 3: PointNet++ Architecture. Taken from Qi et al. [2017b].

For PointNet, the input is a collection of all point cloud data in one scan, expressed as an $N \times 3$ 2-dimensional tensor (i.e., a $N \times 3$ matrix), where N represents the number of point clouds, and 3 corresponds its spatial coordinates. The mapping of a point set is expressed by:

$$\mathbb{R}^{N^3} \longrightarrow \mathbb{R}^{\mathbb{N}} \tag{2}$$

where N is the number of points.

The input points are first aligned by multiplying a transformation network called T-Net, which resembles a mini PointNet and encodes the geometric transformation upon the point set by learning the conversion matrix to ensure the invariance of the model. After extracting the features of each point cloud through multiple multi-layer perceptrons (MLPs), a T-Net is used again to align the features. Then max-pooling operations are performed on each dimension of the feature to get the final global feature. For classification tasks, the global features are predicted by an MLP to predict the final classification score; for segmentation tasks, the global features are concatenated with the

previously learned local features of each point cloud to encode the per-point features, and then the classification results of each data point are obtained by another MLP.

For PointNet++, the input data are not constrained to be in $N \times 3$ dimension, but a k-dimensional vector encoding other geometric attributes, such as the normals and curvatures. The mapping from the input to the output can be expressed by:

$$\mathbb{R}^{N^K} \longrightarrow \mathbb{R}^{\mathbb{N}} \tag{3}$$

where N is the number of points, and K is the dimension of the feature vector.

PointNet++ consists of the following key modules:

Sampling Lidar can have up to 100k data points in a single frame. If we extract local features for each point, the amount of calculation is tremendous. Therefore, an algorithm that Qi et al. [2017b] propose, farthest point sampling (FPS), is performed to sample the data points first, which works by repeatedly placing the next sample point in the middle of the least-known area of the sampling domain. Compared with random sampling, this sampling mechanism can better cover the entire sampling space.

Grouping To extract the local features of a point, the "locality" of a point is defined as the points within certain Manhattan distance around it, which is usually determined by the size of the convolution kernel of the convolution layer. Similarly, the locality of a point in the point cloud data is partially composed of other points in the spherical space drawn by a given radius around it. The role of the combined layer is to find all the points that make up its neighbourhood after passing through the sampling layer, to facilitate the subsequent feature extraction for each local spatial neighbourhood.

Feature learning Because PointNet provides a feature extraction network based on point cloud data, we can use PointNet to extract features from each part of the combined layer to obtain local features. It is worth noting that though each part given by the combined layer may be composed of a different number of points, after passing PointNet, the characteristics of consistent dimensions are determined by the value of K.

The above layers constitute the basic processing module of PointNet++. If multiple such processing modules are cascaded together, PointNet++ can get deep semantic features from shallow features like CNNs. For segmentation tasks, it is also necessary to upsample the features after downsampling, so that each point in the original point cloud is assigned with corresponding features. This upsampling process is obtained by interpolation calculation of the nearest K neighbouring points.

4.2 PointCNN

For a typical convolution operation, the output changes with the order of the input points, which is an undesired property. Therefore, the input order of the points in the point cloud is the main problem that hinders the operation of convolution. To address the issue, Li et al. [2018b] define a transformation matrix $\mathcal{X}$, which can process the input points with an arbitrary order to obtain a feature that is order-independent. Figure 4 details the $\mathcal{X}$-Conv operator, where $\mathbf{P} = (p_1, p_2, ..., p_k)^T$ is the neighbouring points with $\mathbf{F} = (f_1, f_2, ..., f_k)^T$ as their features. $\mathbf{K}$ represents the convolution kernel and p represents the representative points. The operation can also be expressed more concisely with

$$\mathbf{F}_p = \mathcal{X} - Conv(\mathbf{K}, p, \mathbf{P}, \mathbf{F}) = Conv(\mathbf{K}, MLP(\mathcal{X} - p) \times [MLP_\rho(\mathcal{X} - p), \mathbf{F}]. \tag{4}$$

With the $\mathcal{X}$-Conv operation as building blokes, PointCNN can perform convolutions on the unordered point set, similar to that of 2-dimension grids, as illustrated in Figure 5.

4.3 Superpoint Graphs based Segmentation

Superpoint graphs (SPG) can be used for large-scale point cloud semantic segmentation [Landrieu and Simonovsky, 2018]. SPG is a structure that can well represent the topology of point clouds, and can well express the contextual relationship of images. The point cloud of a typical scene has millions of points, but SPG can downsample them to hundreds of points while maintaining rich feature representations, so that it can be processed with pipelines such as PointNet.

ALGORITHM 1: $\mathcal{X}$-Conv Operator

Input : $\mathbf{K}$, p, $\mathbf{P}$, $\mathbf{F}$
Output : $\mathbf{F}_p$ ▷ Features "projected", or "aggregated", into representative point p

1: $\mathbf{P}' \leftarrow \mathbf{P} - p$ ▷ Move $\mathbf{P}$ to local coordinate system of p
2: $\mathbf{F}_\delta \leftarrow MLP_\delta(\mathbf{P}')$ ▷ **Individually** lift each point into C_δ dimensional space
3: $\mathbf{F}_* \leftarrow [\mathbf{F}_\delta, \mathbf{F}]$ ▷ Concatenate $\mathbf{F}_\delta$ and $\mathbf{F}$, $\mathbf{F}_*$ is a $K \times (C_\delta + C_1)$ matrix
4: $\mathcal{X} \leftarrow MLP(\mathbf{P}')$ ▷ Learn the $K \times K$ $\mathcal{X}$-transformation matrix
5: $\mathbf{F}_\mathcal{X} \leftarrow \mathcal{X} \times \mathbf{F}_*$ ▷ Weight and permute $\mathbf{F}_*$ with the learnt $\mathcal{X}$
6: $\mathbf{F}_p \leftarrow \text{Conv}(\mathbf{K}, \mathbf{F}_\mathcal{X})$ ▷ Finally, typical convolution between $\mathbf{K}$ and $\mathbf{F}_\mathcal{X}$

Figure 4: $\mathcal{X}$-Conv operator. Taken from Li et al. [2018b].

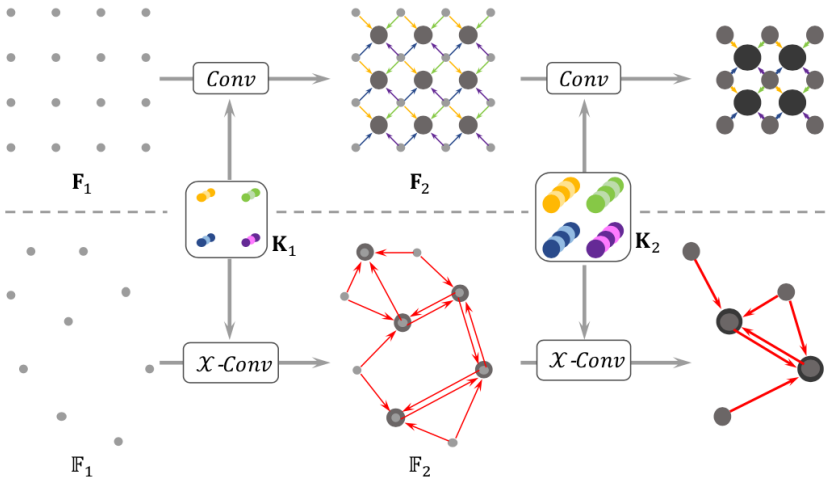
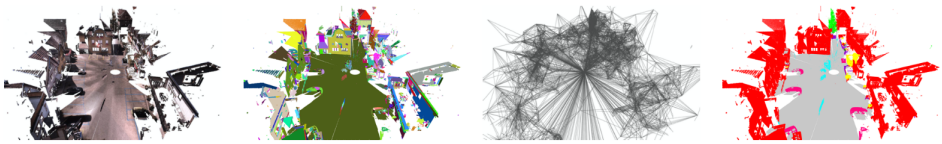


Figure 5: Hierarchical convolution on regular grids (upper) and point clouds (lower). Taken from Li et al. [2018b].

The essence of SPG is a generalized directed graph where nodes represent simple shapes, and edges describe rich edge features to express adjacency relationships. The key steps of using SPG in large-scale point cloud semantic segmentation are as follows: first geometrically segment the original point cloud to get the superpoints. This process is an unsupervised process with high efficiency. Then the SPG can be calculated with the result of geometric segmentation. Finally, the point cloud has compressed from millions to hundreds. The obtained SPG is provided to some contextual segmentation methods where, for instance, PointNet is used. The individual steps in the pipeline are visualised in Figure 6.



(a) RGB point cloud (b) Geometric partition (c) Superpoint graph (d) Semantic segmentation

Figure 6: Visualization of individual steps in using SPG for large-scale point cloud semantic segmentation. Taken from Landrieu and Simonovsky [2018]

4.4 Dynamic Graph CNN for Learning on Point Clouds

Graphs, as a topology structure, are capable of imposing relational inductive bias in data, i.e., the prior knowledge on the problem-solving [Nickel et al., 2015]. The EdgeConv proposed by Wang et al. [2019a] is suitable for CNN-based high-level tasks on point clouds including classification and segmentation. Compared to pipelines operating on each point independently (e.g. PointNet), EdgeConv has advantages incoorporating local neighbourhood information and can be stacked or

recurrently applied to learn global information. As shown in Figure 7, from a point pair x_i and x_j, the edge feature e_{ij} is computed. The output of EdgeConv is calculated by aggregating the edge features.

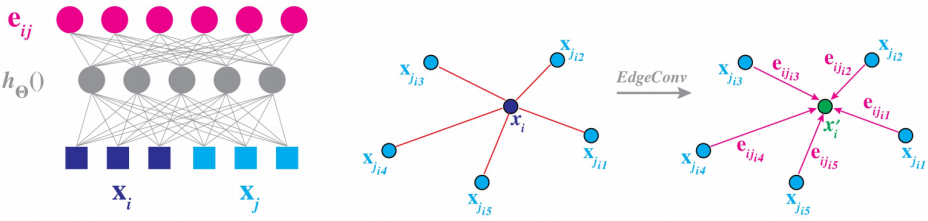


Figure 7: Left: Computing an edge feature. Right: The EdgeConv operation. Taken from Wang et al. [2019a].

Unlike graph CNNs, the proposed Dynamic Graph CNN (DGCNN) is dynamically updated after each layer of the network where the k-nearest neighbours of a point changes from layer to layer, which leads to the diffusion of information non-locally throughout the whole point cloud.

4.5 VoteNet

The VoteNet proposed by Qi et al. [2019] is a point cloud 3D detection framework that directly processes raw point cloud data which does not rely on any 2D detector. This detection network is inspired by the generalized Hough voting process for object detection. Provided the point cloud of the 3D scenario, VoteNet votes for object centres and then groups and aggregates the votes to estimate 3D bounding boxes and semantic classes of objects, as shown in Figure 8.

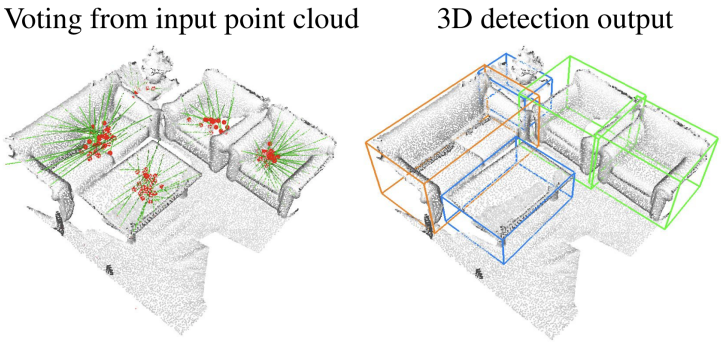


Figure 8: 3D object detection in point clouds with a deep Hough voting model. Taken from Qi et al. [2019].

The pipeline of VoteNet is shown in Figure 8. The entire network can be divided into two parts: one part processes existing points to generate votes; the other part processes virtual points (i.e., the votes) to detect and classify objects.

Given an input point cloud containing N points and (X, Y, Z) coordinates, a backbone network (implemented using PointNet++) learns the depth characteristics of these points, and output a subset of M points, which is considered as seed points. Each seed independently generates a vote through the voting module. Then the votes are grouped into clusters and processed by the proposal module to generate the final proposal.

4.6 Network Comparison

To select the proper network for indoor feature classification on our dataset (see section 5), the performance of these networks are evaluated on the Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [Armeni et al., 2016], among which the visualisation of predictions from PointNet++ and PointCNN are shown in Figure 10. The evaluation results are shown in Table 1. Note that since PointNet is overpassed by PointNet++ with improved architecture design, only PointNet++ is

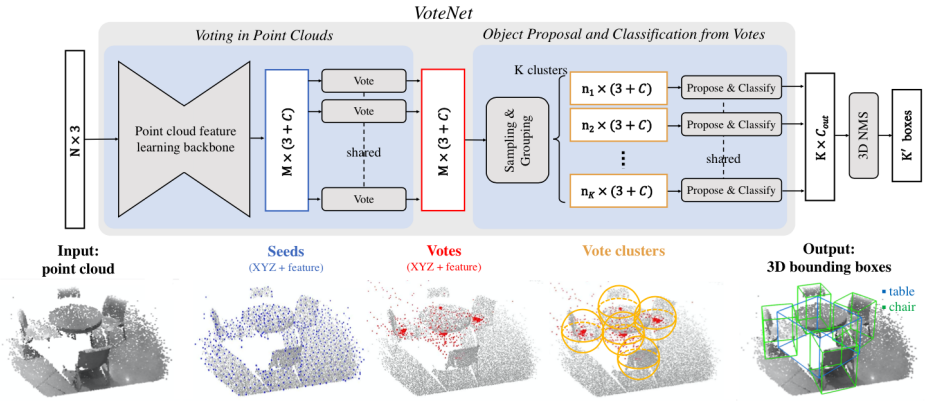
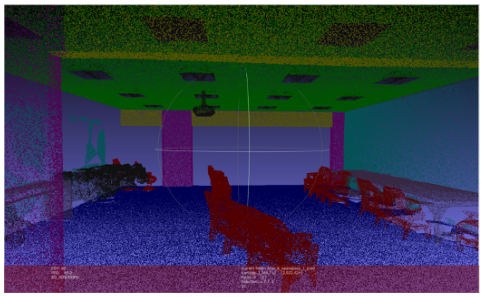
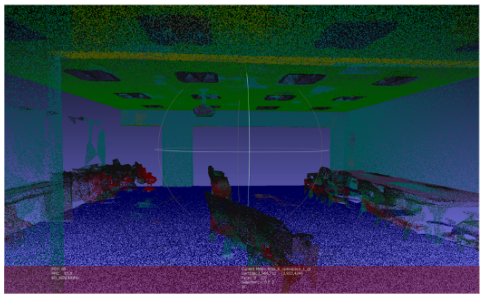


Figure 9: VoteNet Architecture. Taken from Qi et al. [2019].

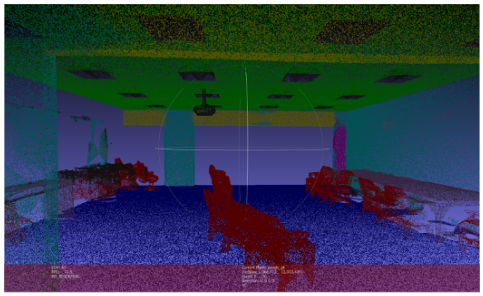
evaluated. Also, due to the VoteNet requires bounding boxes annotation in the training data which is incompatible with the S3DIS, it is not evaluated and also excluded from further experiments.



(a) Ground Truth



(b) PointNet++ Prediction



(c) PointCNN Prediction

Figure 10: Point Classification Result on S3DIS Dataset

Based on the evaluation results and the level of complexity regarding the configurations for different neural networks, we select PointCNN for further experiments in section 5.

Network	Training Time	Training Epoch	OA	CA	IoU
PointNet++	20h	12	0,791	0,600	0.516
PointCNN	16h	94	0,893	0,794	0.714
DGCNN	pretrained	pretrained	0.864	0.711	0.601
SPG	pretrained	pretrained	0.880	0.818	0.714

Table 1: Evaluation of different neural networks on S3DIS dataset. OA means overall accuracy, CA means class accuracy, IoU means Intersection over Union. See section 5.2 for more details on these metrics.

5 Case Study

To test our research, we have been in contact with Esri Nederland and the NS for a use case in the indoor environment. This section will explain the steps taken in this experiment and the problems encountered in the real-world environment where we tested. After this the results for our experiment are presented in two stages, the first being the initial results and the second being the improved results by modifying the training data.

5.1 The experiment

During the neural network comparison part of our research, our contact at Esri Nederland had been working to get in contact with companies that would be interested in providing data for our testing phase. The company that ended up providing this data was the NS. What we got from them was a point cloud captured the inside of Amersfoort Central Station (see Figure 11). This point cloud was an unlabeled LiDAR point cloud with photographic information placed on the points so the points also have an RGB value. The goal of the experiment was to add extra value to the data the NS had captured in this station and potentially also to add value to data captured at other locations that would have similar features. The way deep learning can add value to these datasets is in the classification of features present in the scene. This information can be used for a lot of different purposes but the most prevalent one in our opinion is for asset management, which is also why this experiment was set up in the first place. For the NS it is of high value to be able to know where their assets are in the station and how many of them are present. This is needed as a lot of workers from the NS are not bound to a single station and therefore are not always in an environment they are familiar with, therefore information on where the security cameras are placed, how many garbage cans need to be emptied and how many vending machines need to be refilled for instance is of value for different NS personnel. As the environment in a station is also not necessarily a static one it also happens that these assets are moved to different locations, that a restructuring of the station happens or that some assets break and are removed from the scene for instance. In these kinds of situations, the moment a new scan of the station is made, the assets can also be re-identified from this data within the hour by running a neural network on it, reducing the amount of work when keeping track of the assets.

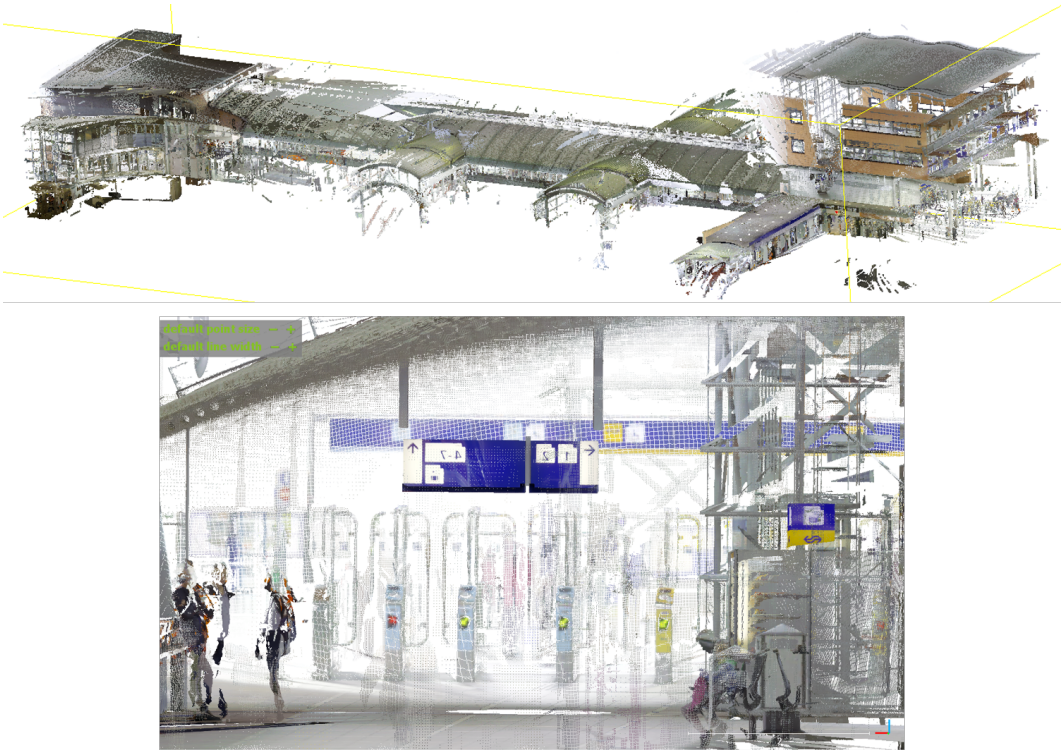


Figure 11: Top: overview of Amersfoort railway station raw data from NS; bottom: sample of point clouds data inside the station. These data was acquired on October 2019 in 19 different scan locations inside the station. File format in .e57 with size around 15GB.

5.2 Workflow

The workflow that we used during the experiment can be divided into three main flows: data pre-processing, data processing, and evaluation (see Figure 12). As for data pre-processing, we use LASTools [Isenburg et al., 2006], ArcGIS Pro [Environmental Systems Research Institute (ESRI), 2012], CloudCompare [clo, 2020], and batch codes for converting, editing, and labeling the data. The raw data file is in .e57 format so we need to do the conversion to produce the generate the .las files. We consider using .las format to handle the data due to our knowledge of that file format from our previous courses in GEO1004, GEO1015, and GEO1016.

After we obtain point clouds in .las format, we split it into several scenes because the raw files are too large to handle (around 18GB). We divided the scenes into blocks based on x and y values (see Table 8 and Figure 55 in Appendix C). The output file size of cut scenes are varied from 9MB to 600MB. Then, we identify over all objects in the station and narrow our objects for the experiment. At the beginning, we only focus on six objects (yellow eggs, garbage cans, couches, ticket machine, vending machines, and entrance gates). Due to our evaluation of the initial results, where we get poor accuracy with unbalanced points, we grow our class number so we ended up having 19 classes. The list of our initial class number is provided in section 5.3 and the final class in section 5.1.

Based on our interest objects, we identify all scenes having that objects and only focus on those. Next, the editing process to segment or remove some noisy points is needed, followed by manual labelling. More detailed about this task is described in section 5.3. With the manual labelling, we obtain the classified point clouds and convert it into text files containing XYZ, RGB, and class code information in one file using LASTools. Then, we develop a batch file to separate xyzrgb and class code into two different text files that follow some arrangements and extension (detail in the Appendix). In this study, we used XYZ and RGB values for training. Besides, we can modify our attributes, for example, if we want to use XYZ only or we add intensity attribute to be learned in the training.

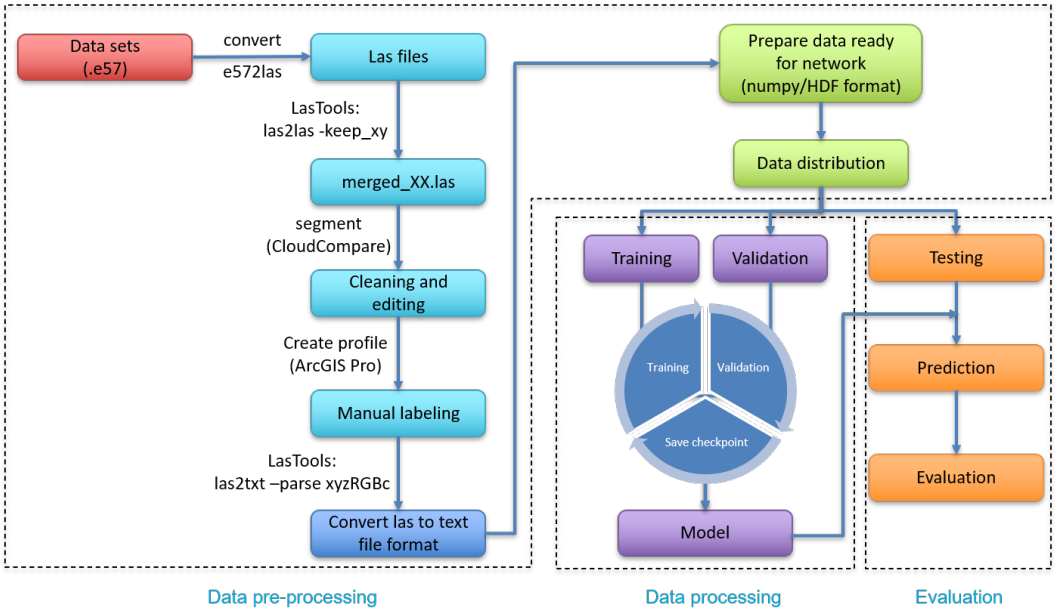


Figure 12: Research workflow

Using the latest files, we can prepare our data sets ready for data processing to correspond to the networks. PointNet++ require the input for training in the numpy (.npy) format containing XYZ, RGB, and class code. So, we develop a python code to convert the text files into numpy format. Then, we adapt some parts of the original code to make it compatible with our class number. As for PointCNN, the input for training is in HDF (.h5) format containing all points and labels within blocks (see section 4.2). PointCNN provides data conversion code, but we need to adjust some parts according to our data. Since the latest files are intentionally developed for PointCNN, we can directly apply the adjusted data conversion.

Next, we distribute the data into training, validation, and testing sets. For the initial experiment, we use two networks which are PointNet++ and PointCNN based on the comparison overview on the previous section. For each, we leave the training run for about 6 hours. For the following experiments, we only use PointCNN with running time varied from 1 to 6 hours. The phase for each network is similar. Once we start the training, the network begins to process the input data within the epoch setting. At a certain stage, e.g. after 500 iterations, it will calculate the accuracy, loss value, and save the model as one checkpoint. Then, it will continue to learn the data. In this case, after we start our network, we monitor the training progress based on the accuracy and loss values. After some time, we can stop the training process when the accuracy value get stagnant shown by the flat curve in the graph.

The model generated from the training is used for testing, a reversed process. In the training stage, we have all the information and try to find the optimum model. In the testing stage, the model is used to obtain the classification code for all points. To evaluate the results, we use accuracy and Intersection over Union (IoU) by comparing predicted and ground truth values.

The overall accuracy describes the ratio between the numbers of points that are equal to truth values with the total number of points. The class accuracy depicts the ratio between the number of points that are equal to true values with the number of points classified for each class. In this study, we set an accuracy of 60% as the threshold for a test to be called successful. Overall accuracy and class precision can be calculated as follows:

$$OA = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{5}$$

and the class precision:

$$CP = \frac{TP}{(FP + TP)} \tag{6}$$

where OA is the overall accuracy, CP is the class precision, TP is the total number of true positive (e.g. if labelled yellow egg predicted as a yellow egg), TN is the total number of true negative (e.g. if labelled non-yellow egg predicted as a non-yellow egg), FP is the total number of false positive (e.g. when labelled non-yellow egg predicted as a yellow egg), and FN is the total number of false negative (when labelled yellow egg predicted as a non-yellow egg). The concept of TP, TN, FP, and FN are depicted in Figure 13. In the overall accuracy, the sum of the numerator is equal with the total number of predicted values that classified correctly in each class, while the sum of the denominator, in this case, is equal to the total number of ground truth points.

On the other hand, IoU expresses the ratio of the overlapping area and the union area between the predicted and the ground truth (see Figure 14). Since point clouds are anisotropically distributed, the concept of overlapping is downgraded from area level to point level. This means that instead of looking at the overlapping areas from two sets in respect to the union of the two sets, we instead look at the number of points which are classified the same in both sets in respect to the union of the two sets. The value range from 0 to 1 or 0 to 100 (in percentage) where 1 or 100 means the prediction matches the ground truth and 0 means the opposite.

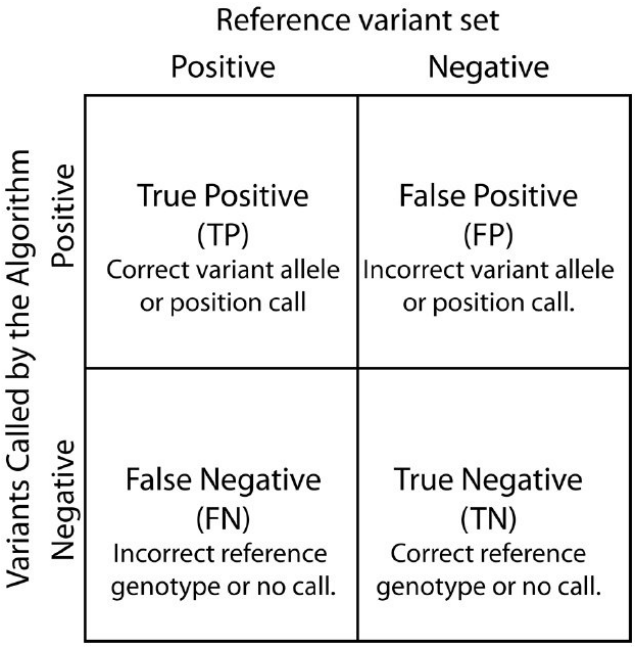


Figure 13: Illustration for True Positive, False Positive, False Negative, and True Negative [Olson et al., 2015]

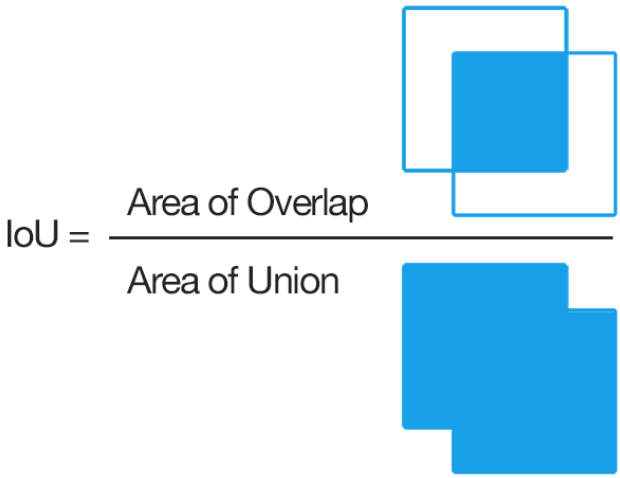


Figure 14: Illustration of Intersection over Union in 2D (Wikipedia [2020]).

5.3 Data labeling

Aside from this preparation work explained in the workflow in section 5.2 there is also the data itself that needs to be prepared for good results. The first data we got from the client was an unlabeled point cloud, meaning no points have been classified as of yet. This is a problem for the training process as the neural networks learn based on the classification of the points and their spatial neighbourhood, so the data needed to be labelled manually. To do this we first decided in consultation with the NS on some of the assets they would be interested in to recognize for the testing of the method, where the quantity and distribution of the points are taken into consideration.

The following assets were chosen:

- **Yellow Eggs (a type of security camera used at the stations)**
- **Garbage Cans**
- **Benches**
- **Ticket Machines**
- **Vending Machines**
- **Entrance Gates**

The objects that are NS specific can also be seen in Figure 15.

Figure 15: On the left a yellow egg can be seen, in the middle there are some ticket machines and on the right there are entrance gates

The labelling of these assets has been done in ArcGIS Pro. Here we could close in on the feature we wanted to label using the profile viewing option, then we could select the points we wanted to label and we could modify the class code belonging to those points to the ones we decided that would represent our objects. As the neural networks we are working with want a continuous numbering in the training data this meant moving away from the standard classification numbers that are used in ArcGIS and using them in a way more suited for our purpose (e.g. classification number 2 normally represents ground in ArcGIS but in our results, it represents garbage cans). This process of labelling has been done for all chosen assets leaving everything that was not one of these assets as unclassified. This turned out not to necessarily be the best option, but more on that in section 5.5.

There were also some harder decisions to make in the labelling process. As the data was captured from different points in the station the points do not always align properly. This can be due to distance errors, reflection in glass or other reflecting objects, people walking through scanned areas, occlusion by other objects etc. An example of this can be seen in Figure 16, here there is a pretty dense capture of the yellow egg camera in the middle of the image but to the right there is also a shade that captures the same object but does not place it at the right place in space. Now we need to choose to label these misplaced points either as a yellow egg or unclassified, or possibly even remove them. All of these actions have a different impact on the training result.

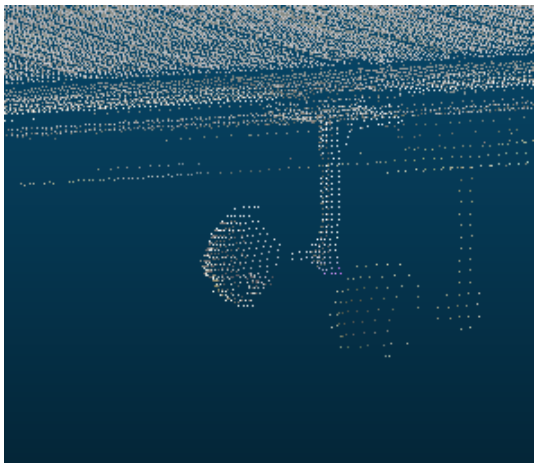


Figure 16: The yellow egg is split into two parts

Another problem in the labelling process was that of misplaced colours that were added to the scene. As mentioned in Section 5.1, the data used in this experiment is LiDAR which is enhanced with image data to provide colour. This was however not always done most optimally as some moving objects in the scene, most people, are draped onto points that should not be that colour. These issues are to be expected in a scene filled with constantly moving individuals, but they can nevertheless make training harder as the RGB information is not always a trustable factor as can be seen in Figure 17. Once again the choice to add or not add this data is something that influences the result.

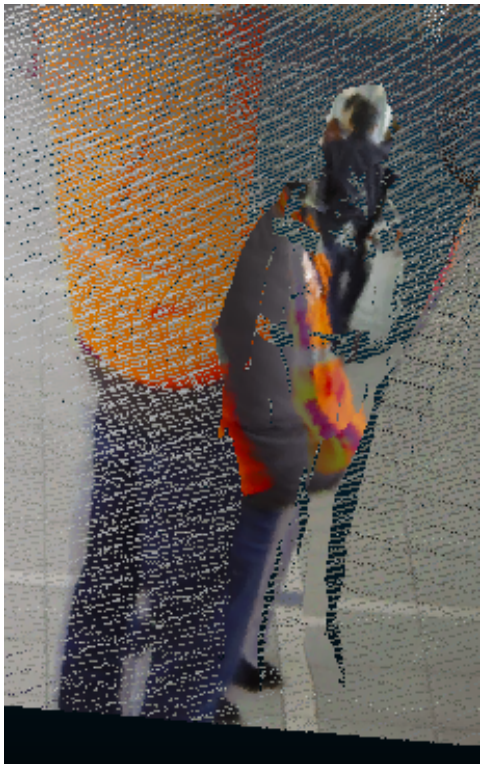


Figure 17: The man standing in the image is also draped onto the floor corrupting the colour of the floor points

5.4 Initial results

With the initial labelled data sets on Amersfoort, we use PointNet++ and PointCNN as our network for training. The initial results show that the networks cannot converge (see Figure 18). Based on the evaluation, the overall accuracy is higher than 80% for both of the networks. However, the class accuracy for all classes, except unclassified, are nearly zero. It turns out that the high overall accuracy occurs due to abundant unclassified points and it affects the other classes. From this intermediate result, we find some drawbacks as follows:

1. The number of points belong to each classes are not balance. After the labelling process, we still have unlabeled points more than 90%.
2. The geometry of many objects are not fully covered by point clouds. For example, points on a ticket machine are only captured from one side, couches are mostly occluded by people, etc.

Concerning those issues, we rearrange the possibility for our experiments as follows:

1. For data pre-processing:
   (a) re-label the scenes by adding more class to reduce the number of unclassified points;
   (b) remove noisy points such as people, floating objects, objects with less points, etc.
2. For data processing:
   (a) try to train and test on the labelled data in a small scene. From this we can expand the experiments to even larger scenes and distribute the data into training and testing sets;
   (b) assign different weights for each class. Classes with less points are considered to have higher weight, vice versa;
   (c) try to exclude RGB values or add intensity in the data for training.

We applied some improvements based on those possibilities and discuss the results in the following section.

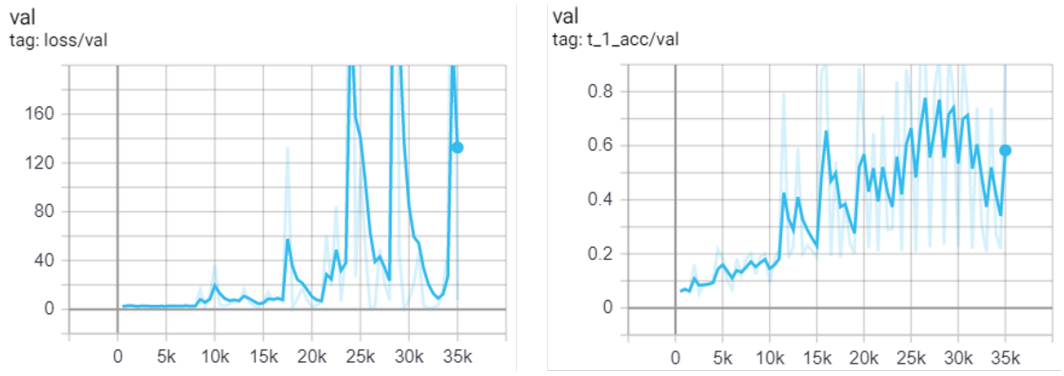


Figure 18: Loss and accuracy values based on validation.

5.5 Improved results

The results acquired in the initial results seemed to depend heavily on the way the data was labelled. As we have seen in the results the overall accuracy was quite high, but when looking at the classes separately it became quite clear immediately that they were all hugely under-represented. The remaining unclassified points in our scene where about 99% of the data, this meant all points of interest shared a bit more than 1% for the training. As a consequence of this unbalance it seemed that the network mostly learned that it is best to almost always guess unclassified. As already discovered in some investigation done by Esri (Kudinov [2020]) unbalanced classes can cause very bad results in machine learning. The classical approach to get rid of such a problem would most often be adding more training data. Unfortunately, in our case, we see such a mismatch in features of interest and data available in a station that adding a station or two would probably not solve this issue.

So we came up with different ways to compensate for this problem in the data which we have put to the test to see which worked best in our indoor scene.

The first idea was making the training scene a lot smaller and more focused on wanted assets than unclassified ones, to reduce the data that is unclassified making the training data more balanced. For the results see section 5.5.1.

The second idea was to split the unclassified data into more classes, while still keeping all data. This would reduce the impact of the unclassified data but would not increase the percentage our wanted features were in the total data scheme. For the results see section 5.5.2.

The third idea was something similar to what is also done in the investigation done by Esri (Kudinov [2020]), meaning focusing on one object during training in local neighbourhoods of that object and only using the class for the neighbourhood and an others class. This would increase the percentage of available data of interest for a specific feature but will reduce the total training data by a lot. For the results see section 5.5.3.

5.5.1 Training on the small scenes

Based on the experiences on the initial results, we changed our approach to now trying to train on several small scenes. These scenes were clipped to focus on wanted assets and noise in the scene such as people, floating objects, etc. was removed (see Figure 20). To reduce the influence of the unclassified class we added new classes, which are floor, SOS pole, and advertisement board on the ground. For this experiment, we did not separate the data for training and validation at first to try whether we can get some results or not. This is because in the initial result even using the network on that what it had been trained on produced bad results. Afterwards, we adapt the experiment to have separate sets for training, validation, and testing. For this smaller experiment, we trained the network for almost two hours (see Figure 19). The results depicted a promising model. After we tested on the same data sets, the overall accuracy and mean IoU are 0.98 and 0.86 respectively, with unclassified now being the worst-performing class instead of the best performing class. Table 2 shows more detailed metrics accuracy and IoU for each of the classes.

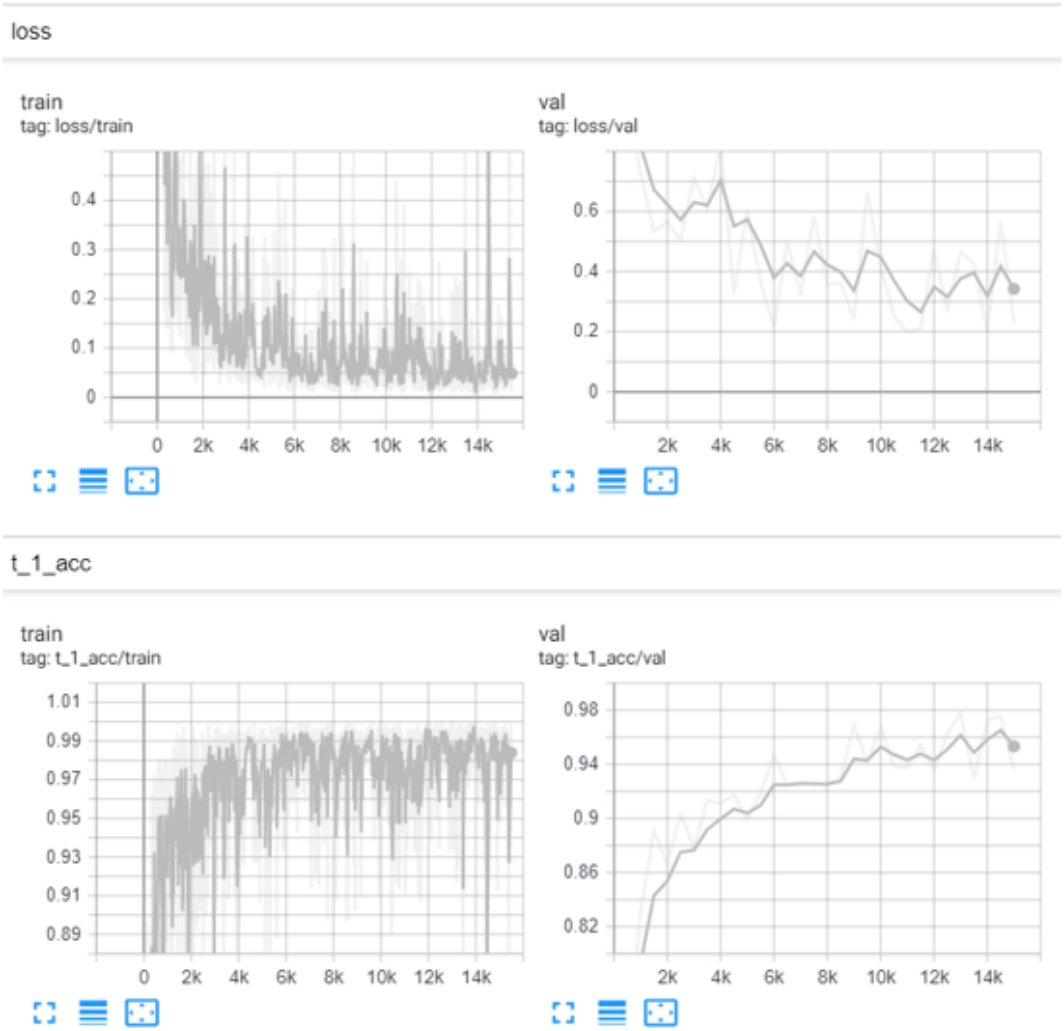


Figure 19: Loss and accuracy values of training and validation in four small scenes.

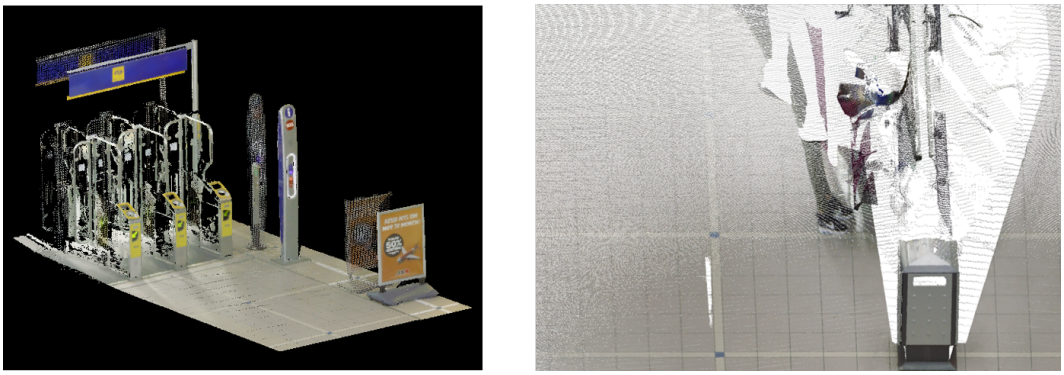


Figure 20: Smaller and cleaned datasets.

As we can see in Figure 21 and Figure 22, the results depict almost identical classes between predicted and ground truth points. Entrance gates were detected correctly even though it is a complex shape. As for the floor, we could not label the class perfectly due to the way the objects were placed such as the couch in Figure 20 so some points were classified wrongly in the training process. Thus,

Metrics / Classes	Unclassified	Garbage can	Couch	Entrance gate	Floor	SOS pole	Info boards
Class Accuracy	0.77	0.97	0.98	0.99	0.98	0.99	0.99
Class IoU	0.26	0.96	0.91	0.98	0.97	0.99	0.96

Table 2: Metric results from several small scenes using PointCNN.

we can see that some points of the floor in ground truth are classified as unclassified points. However, the predicted results interpret these points correctly as floor. Due to this, our metric results are in some places even better than is now shown as the ground truth is also mistaken in some percentages. We can notice this issue mostly in the metric of the unclassified class where the accuracy and IoU values are lower than other classes.

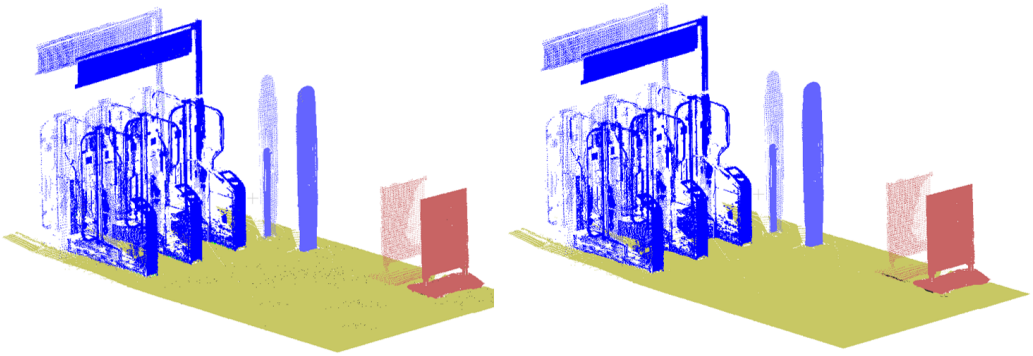


Figure 21: Visualisation result of predicted (left) and ground truth, so what we manually labeled (right) point clouds.

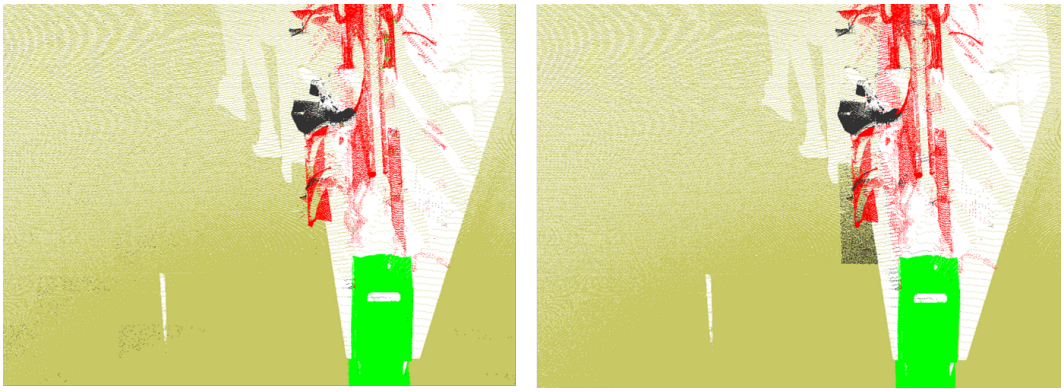


Figure 22: Visualisation result of predicted (left) and ground truth, so what we manually labeled (right) point clouds. Some points on the ground truth are mislabeled, but detected as the correct class.

Next, we tested our trained model to another data set which was not included in our training. This scene had a couch, a garbage can, floor, and unclassified points. The model as shown on the left in Figure 23 gives a reasonable output for the floor even though in this class we had not yet added the floor to the data itself. The result for the garbage can is also fairly accurate as well as the couch. The accuracy for the garbage can and couch are 0.92 and 0.62 respectively, from which the lower result with the couch is understandable as labelling this is once again fairly hard to accurately do so some points were already misclassified. Based on the results, we have some points wrongly classified as entrance gate (155), SOS pole (20), and advertisement board (10,286), but these numbers are

fairly low in comparison to the whole scene. As for both the floor and the unclassified points, we cannot calculate the accuracy correctly due to the unavailable label for the floor on this scene, but the classification seems reasonable as the number of points classified as floor is 3,882,214 in the prediction and the number of unclassified points in the testing data is 3,940,379 which are fairly close.

Metrics/ Classes	Un-classified	Garbage can	Couch	Entrance gate	Floor	SOS pole	Info boards
Number of points	3,940,379	742,483	116,940	-	-	-	-
Positive	7,938	788,479	110,710	155	3,882,214	20	10,286
True Positive	7,027	734,884	87,552	-	-	-	-
Class Precision	0.89	0.93	0.79	0.00	0.00	0.00	0.00
Class IoU	0.00	0.92	0.62	0.00	0.00	0.00	0.00

Table 3: Evaluation of testing a different data set using pretrained model in the small scenes.

As can be seen by the results shown in Table 3 this approach performs substantially better, so a better balanced and more cleaned dataset seems to work quite nicely to improve the results.

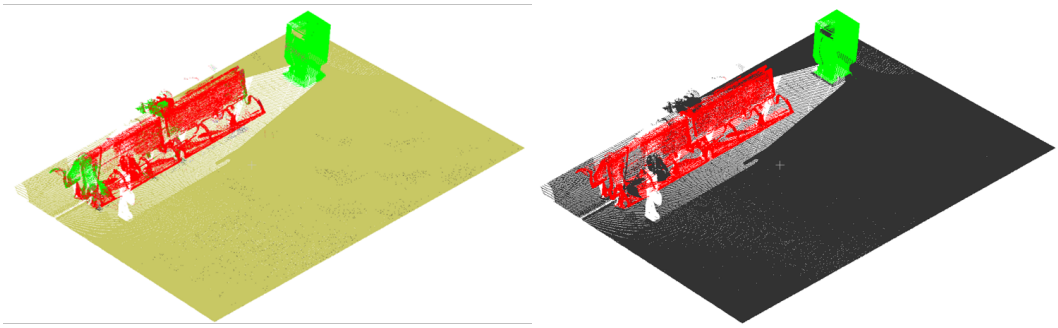


Figure 23: Visualisation result of predicted (left) and ground truth (right) point clouds. In the ground truth in this example, the floor was still labeled as unclassified. Thus, the accuracy for unclassified and floor did not represent correct values.

5.5.2 Training on the large scenes

In our second idea for a possibly better result, we take a similar approach as in the first idea, but deploy it on a larger scale. We now take a way larger and higher scene and label a total of 18 different classes (see Table 4), while also removing a lot of noise from the scene. Some of the more notable new classes are the ceiling, walls and information monitor classes. We then trained a new network on these new data sets and fully separate the training and testing data, for a more fair validation set. We trained with the PointCNN network for about four hours with the same setting as the previous experiments.

For this experiment, we take three larger scenes, which are about 4-8 times as big, along with the smaller scenes from the previous experiment. These sets are separated into a training and a testing set. Two of the larger scenes are included in the training and one is included for testing. However, when we training the network, we got unrealistic values for both the learned accuracy as well as the loss value. While the training process gave an accuracy above 0.90, the validation value showed very low accuracy and there was a very high loss value (see the orange lines in Figure 24). This data reflected that our network has some overfitting issues (see Figure 49), so we tried a different approach. We now removed one larger scene and re-trained our data. Surprisingly, the network improved but the values are still unreasonable with an overfitted model (the accuracy was stagnant around 0.3-0.4). So, when checking our data once more, we learned that there are some areas which were not completely

Index	Class	Index	Class
0	Unclassified	9	Advertisement boards on the ground
1	Yellow eggs	10	Pillars
2	Garbage cans	11	Walls and Posters
3	Couches	12	Digital info monitoring
4	Ticket machines	13	Station ceiling
5	Vending Machines	14	Escalator
6	Entrance gates	15	Lamps
7	Floor	16	Stairs
8	SOS poles	17	Info signs in the air

Table 4: Classification code used in the experiments.

labelled correctly. This was influencing the result more in this larger scene that it seemed to do in the smaller scene but to be sure that it would not influence the result further we removed this large scene from the training set as well, making it so we are training with only one large scene. After retraining the network once more we finally had a reasonable value with loss below 4 and accuracy around 0.6 (see the red lines in Figure 24). Even though the values are improved, the graph still shows overfitting. After four hours of training, the accuracy was not improved, so we stopped the training to see the result.

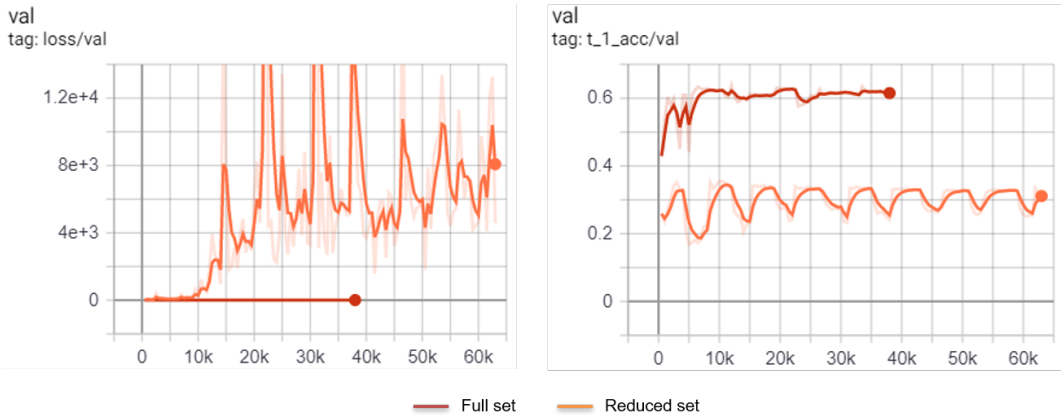


Figure 24: Loss and accuracy values of validation on the full and reduced data sets.

This trained model that had an accuracy of 0.63 and a loss of 3.79, was then put to the test and our two larger scenes which were excluded for training were now used as testing data. Figure 25 depicts the result of one larger scene near the main entrance. Most of ceiling and floor points were classified correctly but, most of the other objects were not. For instance, yellow eggs are visible in this scene. When we take a closer look (see Figure 26), we found that yellow eggs were misclassified as well, with a lot of differently classified points close to each other. We suspect this is due to the small number of points that represented the yellow eggs in the training data once more showing a similar result as in section 5.4. Another reason might have also been that the geometry affected the results, as in this case, we did not have full yellow eggs to train with.

Another scene from the results is shown in Figure 27, where we have the same scene as in Figure 21 but now with the noise prevalent. The left figure depicts the predicted in the large scene, which we clipped to show the same extent as the others, the middle is the ground truth without noise and the right one is the result in the small scene. We learned through this experiment that even though our data shows the same scene with a training set that had no training data removed from it but only added extra data, that the results can still differ. We do expect a lot of the clutter that appears in the large scene to the left can be because of the noise, so the network performance is affected by this condition.

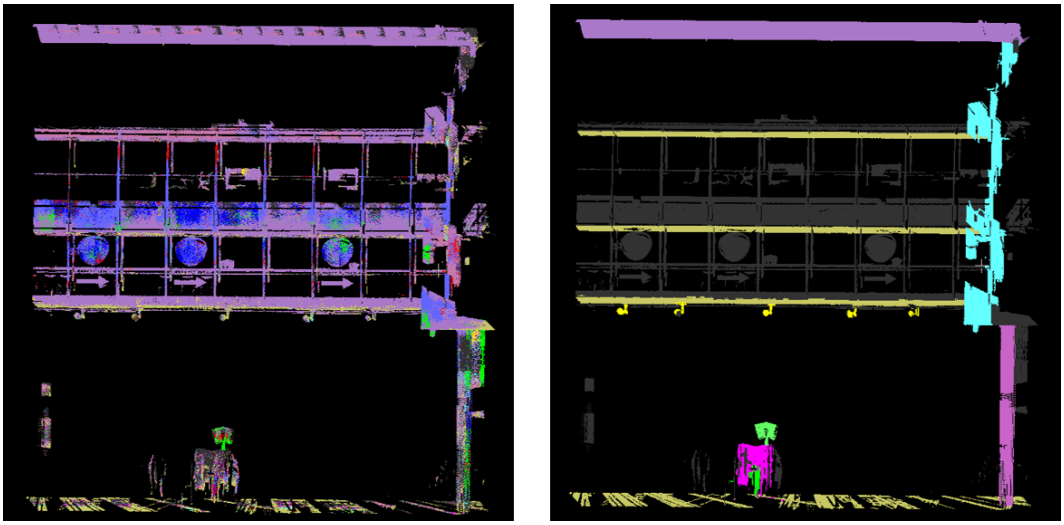


Figure 25: Visualisation result of predicted (left) and ground truth (right) point clouds from testing based on new trained model in the larger scenes.

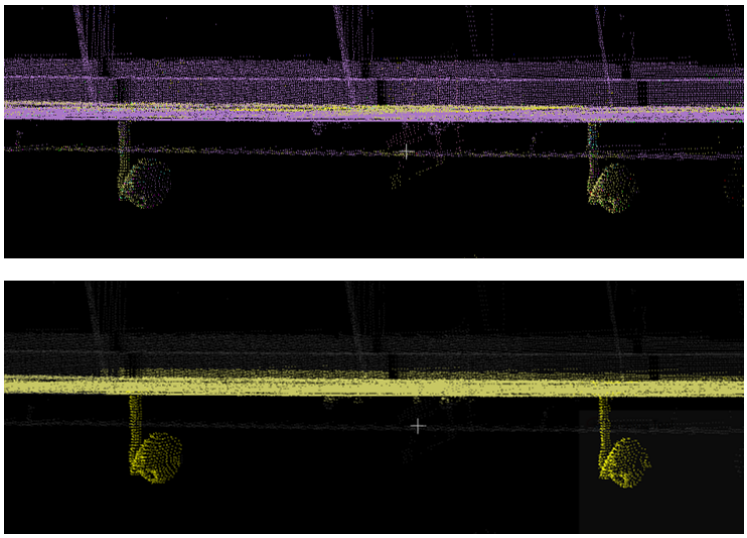


Figure 26: Yellow egg classified as floor. Top: predicted, bottom: ground truth.

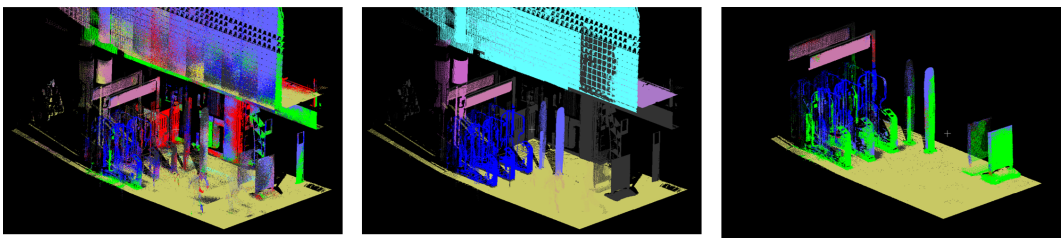


Figure 27: Various classification on one objects. Left: predicted (large scene), middle: ground truth, right: predicted (small scene).

5.5.3 Training focused on one object

In this experiment, we focus more on the yellow eggs so other objects were clipped out and we limit the bounding box only to the yellow eggs. We notice that we have several yellow eggs inside the training. Thus, we separate them into training, validation, and testing set. At the first trial, the train and validation accuracy could reach above 80%. However, the accuracy values from testing are 0.29 for yellow egg and 0.83 for unclassified. Considering the yellow egg is a small object, we try to change the block size of our data set in the preparation stage before training. The original setting is 1.5 and we change it to 0.5. We do not have any reference in this choice, purely trial and error. After we re-train, the output have improved. For each train and validation accuracy are now 0.99 and 0.97 respectively. As for evaluation, we obtain 0.87 for yellow egg and 0.96 for unclassified. The difference result between the first and second trial is depicted in the Figure 28 where the shape of the yellow egg is detected much better in the second trial than the first one. For this, we are still unsure how the block size affects the accuracy results.

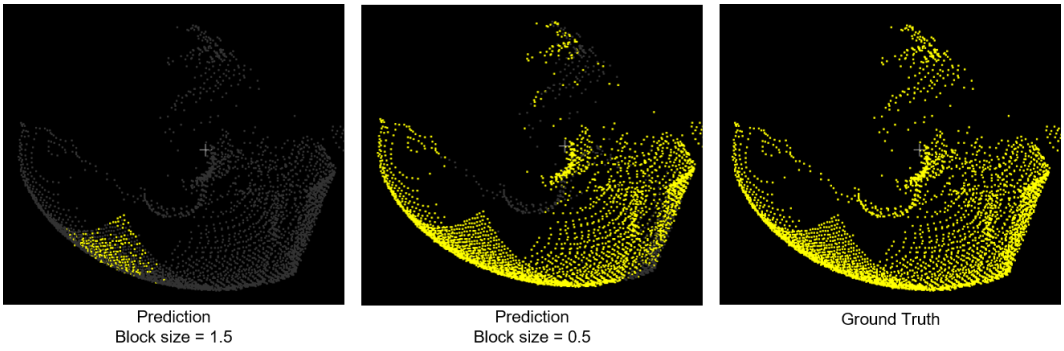


Figure 28: Results of yellow egg only.

Within this model from the yellow egg only, we test on a large scene. The output (Figure 29) illustrates that all objects become yellow egg. We suspect this might due to our training set is not varied enough and the area is too small, so it will only learn a little information.

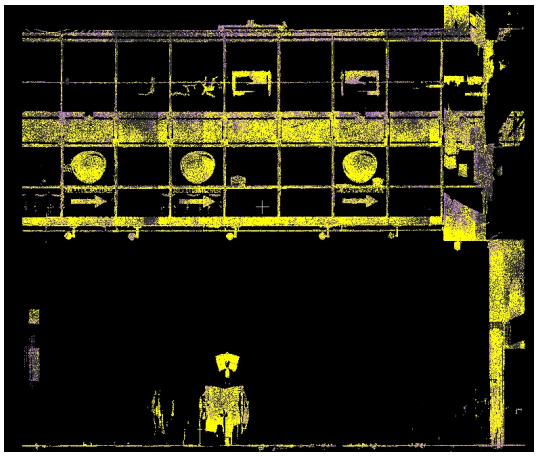


Figure 29: Test on a large scene using the model from yellow egg only.

6 RESTful implementation of a web application

The web application is developed for clients to get the semantically segmented datasets in a more friendly way. The neural networks have high requirements for the computer where they run including the operation system, GPU, memory size, Cuda and cuDNN. Therefore, compared to desktop or offline apps, web apps are better choices for projects based on deep learning which do not require any installation and only run on the browser. Besides, the computation part can be delivered to a high-performance server where the results can be generated more efficiently.

From section 4 we can see that PointCNN can provide better training performance and take less training time compared to other neural networks like PointNet++ and DGCNN. Therefore PointCNN is adopted in this web application to provide clients with more accurate segmentation results in an efficient way.

6.1 Architecture

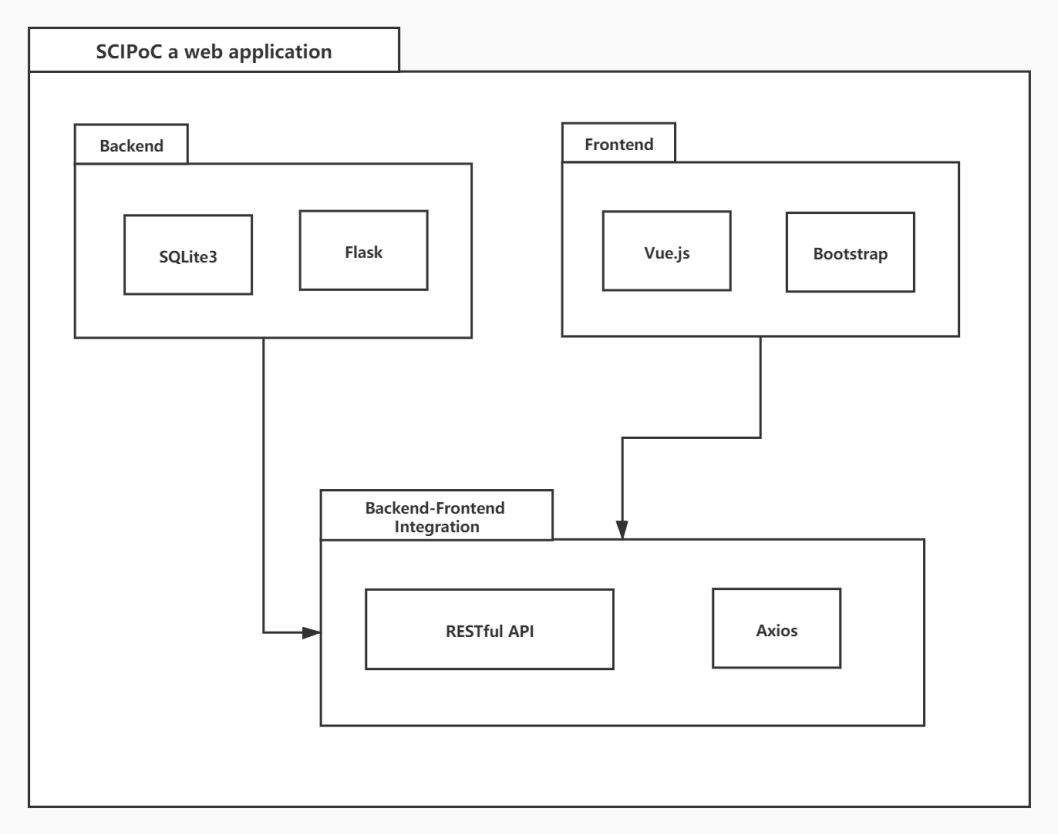


Figure 30: The technical architecture of the web application

The web application is written in $JavaScript$ and $Python$. Figure 30 shows the technical architecture of the web application.

For the frontend part, Vue is used as the main frontend javascript framework. There are three reasons why we choose Vue: first, it is a progressive framework which enables you to write an app very quickly; second reusable components are used which can reduce code redundancy and make the code structure clearer and more logical; finally, the documentation of Vue is constantly being improved and updated which is very friendly to programmers especially for the beginners. To make the web pages highly responsive Bootstrap is used as the Front-end CSS Framework.

For the backend part, we use $Flask$ which is a lightweight web application framework. One of $Flask$'s greatest strengths is that it is based on Python which makes Flask flexible and extensible by using a wide range of external libraries and add-ons. Besides, the neural network PointCNN is

implemented by Python. Therefore, integrating PointCNN into our backend is easier. $SQLite3$ is used as our database. It is lightweight and does not require a separate server process. Compared to the traditional file system storage, using the database can build relationships between different files and can better handle concurrent access using some forms of locking. Besides, when we set sessions in our database, some intermediate and useless datasets can be deleted automatically. Because of its minimalism and simplicity $SQLite$ is usually used to prototype an application. Considering that deep learning usually requires massive datasets for better training performance, a larger database such as $PostgreSQL$ or $Oracle$ will be needed. But at the prototype stage, using $SQlit$ is a better choice.

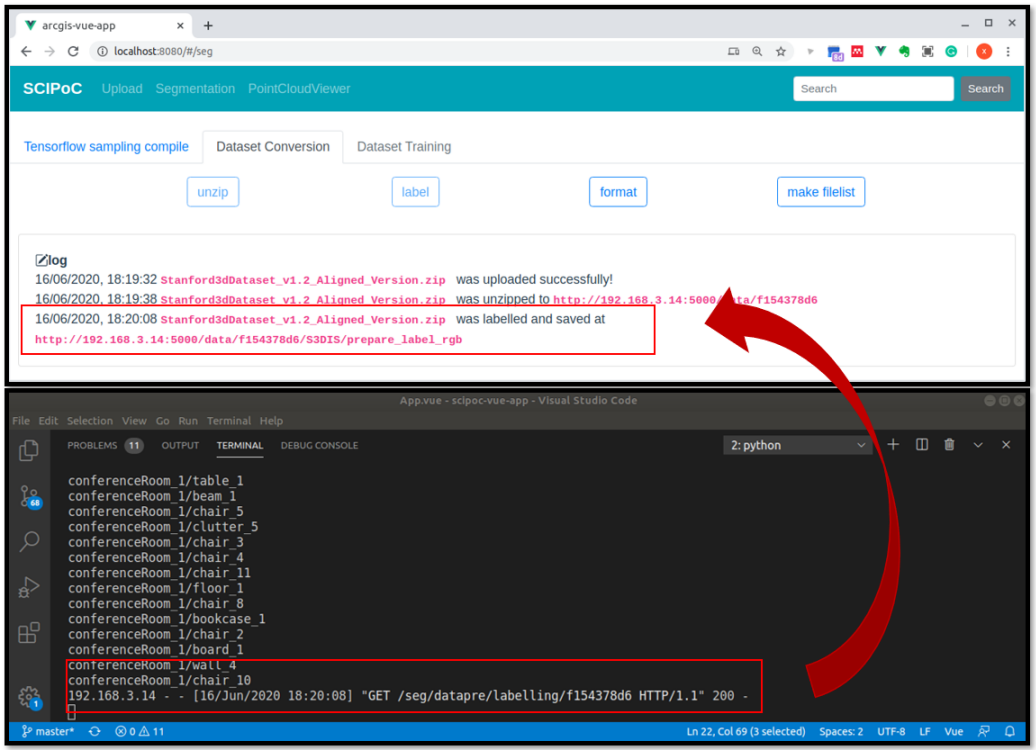


Figure 31: Communication between front and back ends

To send data between clients and the server, we use $Axios$ to connect the frontend and the backend. Figure 31 shows an example of the communication between the front and back ends. $Axios$ is one of the most popular libraries for HTTP communications (GET, POST, PUT, etc.) because of its ease of use. **REST** stands of Representational State Transfer. It's an architecture of standard design between computers, making communication more easier for systems [Abidi, 2019]. $Flask - RESTful$ as an extension for $Flask$ allows for quickly building RESTful APIs.

6.2 Functionality

The web app is designed for users to carry out semantic segmentation and view the results in an easier and interactive way. Figure 32 shows the home page of the web app. The web app provides three main functions: dataset uploading/downloading, semantic segmentation and result visualization.

6.2.1 Dataset Uploading/Downloading

Figure 33 shows the function of dataset uploading. The format of the datasets are constrained to $.zip$ files. One reason for the zip constraint is that we can have all files in a single zip archive. Another reason is that the datasets can be compressed to reduce storage which allows for faster and easier transmission on the web. After the prediction process is finished, users can choose to download the semantically segmented datasets.

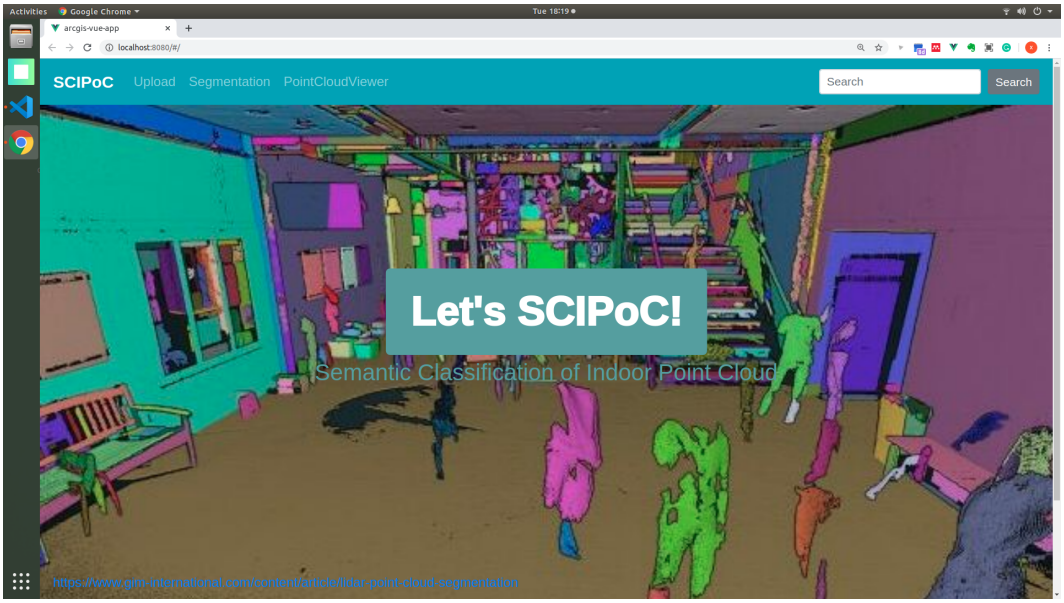


Figure 32: Home page of the web app

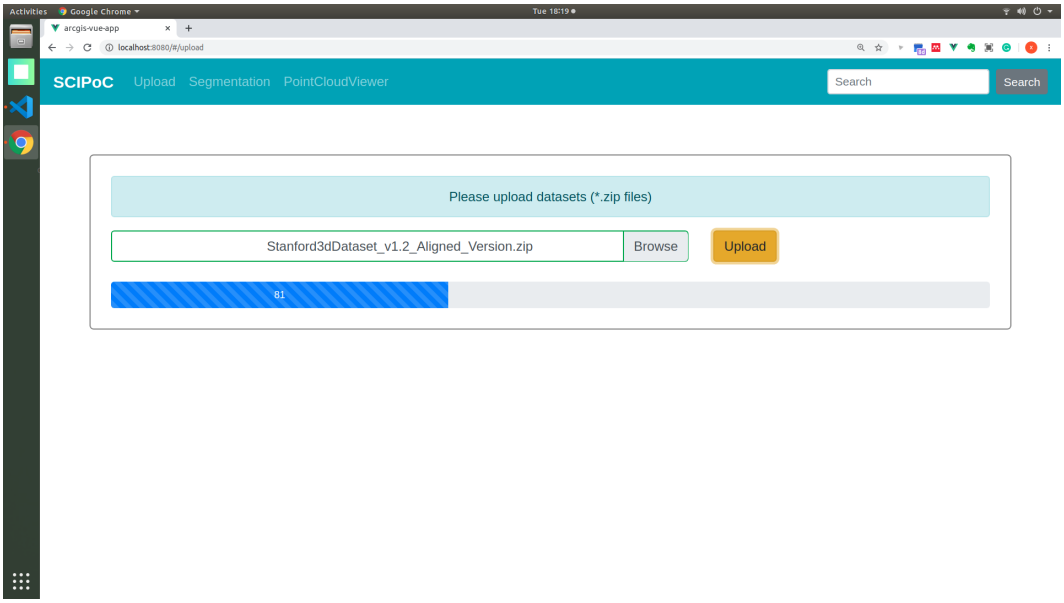


Figure 33: Dataset uploading

6.2.2 Semantic Segmentation

Figure 34 shows the function of semantic segmentation. It includes several steps: Tensorflow sample compile, dataset conversion and dataset training. After finishing each small step, the timeline will be recorded. And the segmentation results are in $.obj$ format.

6.2.3 Visualization

For 3D point cloud visualisation, a cross-browser JavaScript library $ThreeJS$ is used due to its simplicity. It shows $webGL$'s great capability without the requirements of learning WebGL. First, the $.obj$ file is parsed. The 3D vertices and colours are stored in two different arrays. Then all 3D vertices are used to get the centre (average) coordinate of the current point cloud to set the camera.

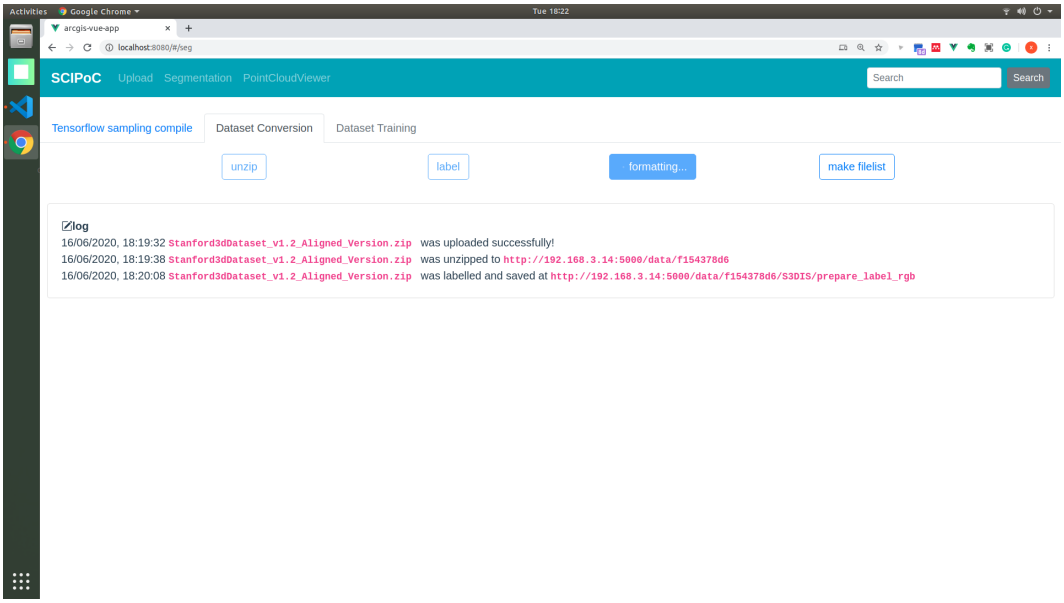


Figure 34: Semantic segmentation

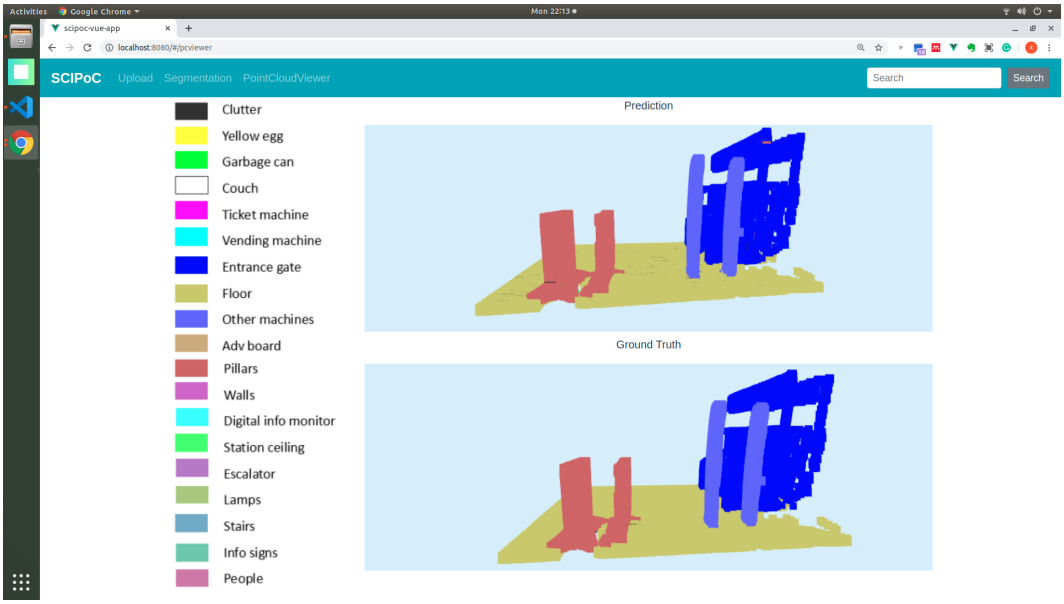


Figure 35: One good result of semantic segmentation

After the camera setting, each point is rendered as a small square with a fixed area and attached the corresponding semantics to present the label. Basic 3D scene operation is implemented like rotation, zooming in and zooming out. These interactive operations can offer the users an insight to "understand" the indoor scene. Figure 35 shows one good semantic segmentation result while the result in Figure 36 is a little messy with many different kinds of entities in the scene.

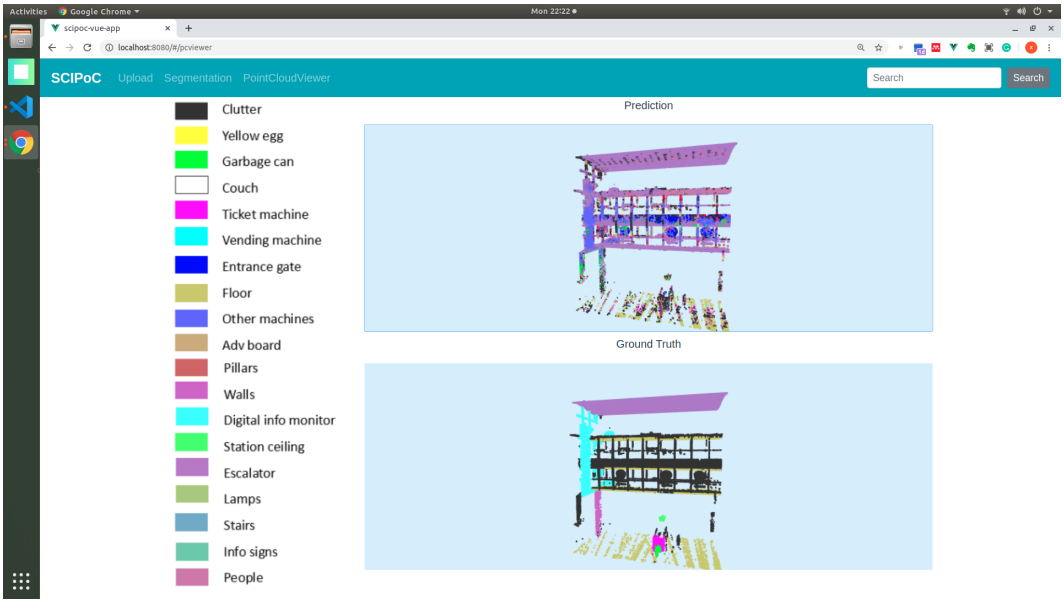


Figure 36: One bad result of semantic segmentation

7 Conclusion

Point cloud data are inherently embedded in 3D space and has rich semantic representations. However, they are often unordered and anisotropically distributed thus unsuitable for a typical CNN to handle. Recently, with the advancement of feature learning from irregular domains, several deep neural networks are available for point cloud analysis.

In this study, we have compared several neural networks for semantic segmentation of point cloud data and implemented one of them to an indoor scene, namely the Amersfoort railway station. Based on the initial comparison, PointCNN and Superpoint Graph give a better result when compared to the other networks. From the user perspective, PointCNN is more flexible to modify and adapt. PointCNN also has better training performance and takes less training time compared to other neural networks like PointNet++ and DGCNN. Thus, we use PointCNN as our network for Amersfoort case study and web application development.

Amersfoort railway station consists of objects like the entrance gates, ticket machines, couches, information boards, etc. These objects are different from the S3DIS indoor scene that we used for the neural network comparison; where the objects mostly are table, board, chair, and other stuff inside an office room. This meant that our data needed to be labelled separately and used to train a new network as indoor features differ a lot more than outdoor features. In total, we split the unclassified data in 18 different classes.

This study has explored several experiments. From the initial experiment, an unbalanced number of points for each class downgrades the training accuracy significantly. To improve this result new experiments are done. In the next experiment, we use subsets from the data, remove the noise from these, and re-train again without separating the training and testing set. The results indicate that PointCNN works can work very well in a smaller environment with less noise and a more balanced class distribution. However, in the third experiment, we have a low accuracy in the larger scenes due to harder to label data, more complex environments and more choice in classes.

Based on these experiments, the quality of the data, the labelling task, and the proportion of the number of points per class can all affect the network significantly. We also noticed in these experiments that the results are very dependent on the training location. This means that the network is pretty well adjusted to the scene it is trained on but as indoor scenes differ a lot in furniture and overall aesthetic it makes a trained network for one location less suited to use in another location. For instance, a network that is trained on a station would be of little use in a hospital as the features it has to recognize are simply too different.

Additionally, we built a web application to integrate part of our research results into a workable application. Users can upload their initial datasets through the web interface then get the semantically segmented results after the segmentation process is done. Besides, we also provide the visualization function in this web app where users can look into the results in a more direct way.

8 Future Research

There are some parts for improvement. First, training a neural network always takes more than 10 hours. It would be better if users can look into the training process through the monitoring interface. Second, only one neural network is adopted. Although in general PointCNN has better performance than other neural networks, some neural networks like DGCNN could provide better results in some specific cases. Third, the web app is currently not connected to the products of Esri. Lastly, our pipeline for semantic segmentation does not inherently support object-level detection from the point cloud which requires post-processing such as clustering, while the possibilities of other end-to-end pipelines for directly extracting objects from the point cloud need to be explored.

9 Acknowledgements

For the research done in this paper, we would like to thank Niels van der Vaart from Esri Nederland for guiding the tools from Esri and for providing us with the ArcGIS Pro Beta version we used during the project. We also want to thank Remco Bunder from the NS for providing us with the opportunity and the data to perform a practical use case on Amersfoort railway station. Last but not least we also want to thank our mentors from the TU Delft, Edward Verbree, Jesús Balado Frias and Martijn Meijers, for providing us with guidance and help throughout the whole project.

References

CloudCompare (version 2.10.2) [GPL software], 2020. URL http://www.cloudcompare.org/.

S. Abidi. Tutorial: Building a RESTful API with Flask - Kite Blog, 2019. URL https://kite.com/blog/python/flask-restful-api-tutorial/.

I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

M. Aubry, U. Schlickewei, and D. Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 1626–1633. IEEE, 2011.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

M. M. Bronstein and I. Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1704–1711. IEEE, 2010.

R. Brown. Running Versatile GPU Workloads in ArcGIS Pro Virtualized Environments with NVIDIA Tesla T4, 2019. URL https://www.esri.com/arcgis-blog/products/arcgis-pro/3d-gis/running-versatile-gpu-workloads-in-arcgis-pro-virtualized-environments-with-nvidia-tesla-t4/.

C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

Environmental Systems Research Institute (ESRI). Arcgis pro release 2.5.1, 2012.

K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9): 1904–1916, 2015.

D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

M. Isenburg, Y. Liu, J. Shewchuk, J. Snoeyink, and T. Thirion. Generating raster dem from mass points via tin streaming. In *International Conference on Geographic Information Science*, pages 186–198. Springer, 2006.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

D. Kudinov. Pointcnn: replacing 50,000 man hours with ai, 2019. URL https://medium.com/geoai/pointcnn-replacing-50-000-man-hours-with-ai-d7397c1e7ffe.

D. Kudinov. Object extraction from mobile lidar point clouds with machine learning, 2020. URL https://medium.com/geoai/object-extraction-from-mobile-lidar-point-clouds-with-machine-learning-cb15fcbb5597.

L. Landrieu and M. Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Li, B. M. Chen, and G. Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018a.

Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. In *Advances in neural information processing systems*, pages 820–830, 2018b.

A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

H. Ling and D. W. Jacobs. Shape classification using the inner-distance. *IEEE transactions on pattern analysis and machine intelligence*, 29(2):286–299, 2007.

J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

J. L. McClelland, D. E. Rumelhart, P. R. Group, et al. Parallel distributed processing. *Explorations in the Microstructure of Cognition*, 2:216–271, 1986.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

K. S. Oh and K. Jung. GPU implementation of neural networks. *Pattern Recognition*, 37(6): 1311–1314, jun 2004. ISSN 00313203. doi: 10.1016/j.patcog.2004.01.013.

N. Olson, S. Lund, R. Colman, J. Foster, J. Sahl, J. Schupp, P. Keim, J. Morrow, M. Salit, and J. Zook. Best practices for evaluating single nucleotide variant calling methods for microbial genomics. *Frontiers in genetics*, 6:235, 07 2015. doi: 10.3389/fgene.2015.00235.

F. Poux. The smart point cloud - structuring 3d intelligent point data. page 268, 2019.

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

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

C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017b.

C. R. Qi, O. Litany, K. He, and L. J. Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9277–9286, 2019.

R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3384–3391. IEEE, 2008.

R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

R. Singh. Deep Learning + GIS = Opportunity, 2019. URL https://www.esri.com/about/newsroom/arcuser/deep-learning/.

H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pages 1383–1392. Wiley Online Library, 2009.

P. V. Tran. Learning to make predictions on graphs with autoencoders. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 237–245. IEEE, 2018.

J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu. Deep learning for smart manufacturing: Methods and applications. *Journal of Manufacturing Systems*, 48:144–156, 2018. doi: 10.1016/j.jmsy.2018.01.003. URL https://doi.org/10.1016/j.jmsy.2018.01.003.

Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019a.

Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019b.

P. Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

Wikipedia. Jaccard index, 2020. URL https://en.wikipedia.org/wiki/Jaccard_index.

Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

X. X. Zhu, D. Tuia, L. Mou, G.-S. Xia, L. Zhang, F. Xu, and F. Fraundorfer. Deep learning in remote sensing: A comprehensive review and list of resources. *IEEE Geoscience and Remote Sensing Magazine*, 5(4):8–36, 2017.

Appendices

A Project plan

In this section, we will explain the plan we have to follow during the Project. This entails the project overview which we revised twice since the PID report, the structure of the project, including the roles that each of us has fulfilled in the team and our meeting schedule, a revision of our MoSCoW, our updated planning for the project and the updated rich picture.

A.1 Project overview

During our project, the end goal gradually evolved bit by bit. At first, the main idea we presented in the PID report was to have the focus of the project on testing the multiple prevalent point cloud-based neural networks and to integrate these into ArcGIS via a web application. This then after the third week of the project turned into a project where we would first test the neural networks and then afterwards take the best performing one and use it on a use case which would be provided by a third party. This research would then have the documentation needed for third parties to use our code via an ArcGIS integration which was still being developed. Then the third evolution of our project came to be as the data provider became the NS and the data provided was an unlabeled set which showed how most clients of Esri usually collect their data. This gave us an environment where we could test our code in a real live test case and first hand with potential clients. This did, however, introduce a factor which we initially introduced as a won't-have in the MoSCoW for the project, which now became a must-have to produce results. So the final version of our project, in the end, became a neural network comparison for the first half of the project which after that was tested on a manually labelled dataset which had a lot of noise and irregular point quality as well as a complex environment. This introduced a lot of problems on the way but as can be seen in section 5.5 we were able to get some promising results even in this kind of scene as well as get some results on bad practices in this kind of environment.

A.2 Project Structure

The structure also had some changes during the project which were mostly in the platforms we used for sharing data. The task division and roles also in practice changed somewhat and the meetings we held also got a bit bigger at the end of the project. In this section, we will go over all these changes in the project structure and the reasons as to why this happened.

A.2.1 Platforms used

Originally the main platform we would use for sharing code and data was going to be GitHub. In the end, this turned out to be quite redundant for our workflow so it gradually phased out of our project. Github is a great platform for sharing code and reviewing it with each other but our project turned out to be less coding than we initially thought. This is because a lot of deep learning code is done by the implementation, instead of by the programmers that use a framework for it. We still needed to do a lot of data pre-processing and we needed to get everything to work on our computers which took a lot of time but all of this had to be done locally meaning the code made was useless for the others of the team if they did not install all the specific requirements as well. This made our workflow change from sharing everything with everyone to a dedicated set-up which we made mobile on itself or kept installed at the pc Zhaiyu dedicated to this project which we could access remotely using teamviewer[1]. This saved us a lot of time with hard installations and lengthy downloads, as well as saving performance as most of our laptops were performing worse than the dedicated pc. The set-ups we made mobile themselves were hosted using google colab which made a remotely accessible cloud hosted computer which ran our code on a server hosted by Google. As already said GitHub was used less and less for code sharing but also for file sharing we moved away from GitHub. This happened at the moment we got access to more and more data, which we needed to be accessible and transferable between our members for the pre-processing steps. As GitHub is not a platform that is made for sharing, big datasets and binary files we decided to switch to google drive which is suited for this

[1] https://www.teamviewer.com/

purpose. This is also because we needed the data to be hosted on google drive to be able to use it with google colab so this came as a natural transition from one platform to the other. After this transition the only reason we used GitHub was for the scrum board which we now simply managed separately during the meetings as the start of the day, making GitHub completely redundant.

Another platform that we used a lot during this project which was not previously mentioned in documentation was our discord server. This has been used for our meetings with the team as well as with the clients, for sharing papers, lectures and other background information and for presenting demo's to the client among other purposes. So this has also been a very valuable asset to us in these days as our main platform for communication with the team as well as with the clients.

A.2.2 Task Division and Roles

During the project, each of the team members had their own role which gave them responsibility for a certain part of the project, to make sure actions were taken efficiently and to make it clear for the team members and the mentors and clients who they needed to contact for which subject. The division was as follows during the project:

- **Project Manager (Mels Smit):** The Project Manager has the task to manage the project as the role suggests. This means fulfilling tasks like making the minutes for the meetings, keeping people on track during the meetings, setting the priorities for sprints, maintaining the scrum board, etc.
- **External Manager (Mihai-Alexandru Erbașu):** The External Manager has the task to manage the contact with the client and the mentors. This means external communication will (most often) go through one person for the team making the communication and asking of questions more structured.
- **Technical Leader (Zhaiyu Chen):** The Technical Leader has the task of managing the technical project. This means having the final say in which software is used, being up to date with what libraries other members are using, checking if the software used by members is compatible with each other, etc.
- **Quality Analyst (Yustisi Ardhitasari Lumban Gaol):** The Quality Analyst has the task of controlling and continuously checking the preliminary and final work. This means testing the results of the code as well as going over the final version of the report, presentations and demos to ensure their quality.
- **Product Manager (Xiaoai Li):** The Product Manager has the task of ensuring the final product is what the client wants. This means formulating a concrete view of the wanted product, putting in work for user satisfaction, having final say in user interface based decisions, etc.

During the projects, these roles did not mean that the members did not have an obligation to help with other tasks instead, a lot of members did work belonging to other roles throughout the whole project. For instance, Mels and Mihai-Alexandru both helped with ensuring the quality by preparing data for the training phase, Yustisi had a lot of influence in the technical department due to being involved in the training of the networks, Xiaoai had a lot of contact with the client from Esri and Zhaiyu helped a lot with the people management in the more technical discussions. So everyone was involved a lot in all tasks to make everything go smooth during the project.

A.2.3 Meetings

The meeting schedule during the project has been fairly consistent. Every Monday, Wednesday and Friday we had a meeting at the start of the day at 9:00. During these meetings, we would discuss what everyone had been doing since the last meeting, any problems that might have arisen and what everyone would be picking up for the next day. We would also keep each other informed about personal problems so everyone knew what was going on in each others lives in these weirder times. On Friday we were joined by the client and Mentors at 10:00 to give them an update on what was done this week and to discuss the steps for the week after. From the start of June onwards these Friday meetings were also joined by Remco Bunder the representative from the NS. In the last week of the project, the meetings were increased to every day of the week to finish the project in time.

A.3 Revision of the MoSCoW

MUST	***SHOULD***
- Output: semantic point cloud - Indoor features: the most common objects in our data set - 1x neural network approach - Input: unlabeled point cloud - Training set: already labeled - The result quality must be measured using the accuracy of the confusion matrix, at least 60%	- Compute the result in a reasonable time - Disseminating the Labelled Point Clouds on the web - Output: literal object (instance level) - Indoor features: other objects - Should try on different ML algorithms - The result quality should get 75%
COULD	***WON'T***
- Output: meshes, LoD representations - Could implement the code as ArcGIS add-in - ArcGIS output	- Label the training

Table 5: Outdated MoSCoW Prioritization of Requirements and Objectives

In our project, we had decided to use the MoSCoW prioritization technique to determine the objectives of this project. This method considers 4 different subdivisions which show the importance of certain objectives:

- ***MUST*** is the category of objectives which mandatorily need to be achieved by the end of the project
- without accomplishing these goals, the project may be considered as incomplete;
- ***SHOULD*** represents those tasks which are extremely nice to complete and which we will put a great emphasis on. While we desire to accomplish all of the mentioned ***SHOULD***s there may be the case that we cannot achieve them all, but the project would still be considered complete;
- ***COULD*** are the objectives which would be nice to have, but we shall only try to accomplish them if and should we be done with the ***MUST***s and ***SHOULD***s;
- the last category is represented by ***WON'T***, which contains a list of tasks which we won't deal with throughout this project. While they might have been nice to do, we usually add here objectives which are outside our scope or which couldn't be accomplished in the already-set time frame.

Throughout our project, some of these MoSCoW priorities came to be under a lot of pressure due to the change in direction our project has taken. As a consequence, some priorities switched places in the MoSCoW ordering and some new priorities came to life as well. On the next page we will go over the MoSCoW table as shown in Table 5 to go over the categories one by one, explaining what the initial priority meant at the start of the project and how it changed over the project afterwards. We will also talk about the newly added priorities and the whole discussion will be summarized in Table 6 showing the MoSCoW requirements at the end of the project.

MUST	***SHOULD***
- Output: semantic point cloud - Indoor features: the most wanted assets for the NS - 1 fully implemented neural network approach - Input: unlabeled point cloud - Training set: already labeled - Labeling the NS data for experiments	- The result quality must be measured using the accuracy of the confusion matrix, at least 60% - Compute the result in a reasonable time - Indoor features: other objects - Should try on different ML algorithms
COULD	***WON'T***
- Disseminating the Labelled Point Clouds on the web - Output: literal object (instance level) - Output: meshes, LoD representations - Could develop an web application to visualize the results - ArcGIS output	

Table 6: Updated MoSCoW Prioritization of Requirements and Objectives

1. ***MUST***
    - *Output: semantic point cloud* - We would like that our output is in the form of a point cloud, where every point has a semantic label, representing one of the object categories which we define in our software;
      This priority stayed rather similar to the initial version. For some more concrete information, we decided upon 18 different classes which could be used for semantic classification and the output is not one point cloud of the whole scene but several smaller ones which when joined still show the whole scene.
    - *Indoor features: the most common objects in our data set* - We would like our neural network to recognize a few of the most common objects which can be found inside of a building, such as chairs, tables, doors, walls, and a few more others;
      The indoor features we wanted to find at first have completely changed at the end of the project. The features we are now looking into are the assets wanted by the NS. This meant less available training samples and more complex shapes so to say we need to find the most common objects is not good enough anymore, therefore the requirement which still is a must-have is to classify indoor features which are wanted assets from the NS.
    - *1x neural network approach* - We would like to have (at least) one fully functioning neural network, so we need to carefully analyze which neural network would be the most suited for our 3D application;
      This also stayed the same. It was a must-have to have 1 neural network that could work for the whole process and it still is.
    - *Input: unlabeled point cloud* - Upon training the neural networks using deep learning techniques, we would like to input an unlabeled point cloud in our software. This unlabeled point cloud should have a labelled counterpart, which would allow us to check for the quality of our classification process. It should be noted that, should a point cloud contain RGB information for its points as well (if the point cloud was generated from photogrammetry, for example), it will be considered as a property of the point (so each point will have different values for the different colours), and not as a label.
      This priority also stayed the same as this is what the networks will be tested against in the end.
    - *Training set: already labelled* - We will perform the training for our neural network using only labelled data sets, which we need to gather and then check the quality for each data set individually;
      This is the same as this is the only data you can feasibly use for training, however, the underlying thought with this was that we would not be labelling the data ourselves which in the end needed to be done to make the project feasible to get any results at all.
    - *The result quality must be measured using the accuracy and precision of confusion matrix; at least 60%* - We would like to ensure that our software manages to classify unlabeled point clouds with an accuracy of at least 60%. Anything less would be considered as a not successful result.
      This priority in the end became an almost unfeasible one. As we worked with unlabeled data from a complex scene with a relatively low amount of training samples, a lot of noise in the data, corrupted colours and incomplete captures of objects we could simply not guarantee good results. This must-have was completely based on the idea of a clean cohesive and balanced training set, which in the end we did not get to use so the strive to an accuracy of at least 60% became a should-have not a must-have in our project.
2. ***SHOULD***
    - *Compute the result in a reasonable time* - We will try to achieve a result which has all other properties mentioned in this document in less than 5 minutes, for a point cloud representing the interior of an entire building;
      Taking the training not into account as this has to be done beforehand, the classification process takes about 5-15 minutes on the sizes we used for testing depending on the density of points, the extends of the area and the metrics needed to be computed. For the whole station, this once again takes longer so within 5 minutes is not a result we can achieve with this dataset. This does however heavily depend on the number of

points that make up the scene as close to scan locations the point density is suddenly in the order of millions for quite little extra information. As a consequence the NS data set is a lot larger than it needs to be at about 860 million points which all have to be processed and processing more than a hundred million points in a neural network within a minute is simply unrealistic to ask from our laptops, so the initial goal was too ambitious.

- *Disseminating the Labeled Point Clouds on the web* - It also ranks with high priority for us the task of segmenting the point cloud into objects (so one group of points would represent a single object, such as a chair, or a table), which in the end could be displayed on a web-application (or perhaps an ArcGIS StoryMap), on which a user has the possibility of selecting and turn on or off each available object;
  This issue, in the end, did not get picked up all that much. With the shifting priorities in the project, the focus kept returning to properly classifying the points we had been given so less and less time was left for processing these points afterwards as well. As a result, this issue was pushed back somewhat to the level of a Could have for our project.
- *Output: literal object (instance level)* - We would like if we can group multiple points such that one grouping represents a singular, atomic object (so besides the classification we'd also have an object id);
  As said for the previous priority as well the further processing of points became more and more a Could have in the project so this underwent the same change.
- *Indoor features: other objects* - Should we have extra time available, we can try to introduce other indoor objects which could be detected by the software
  With the addition of the NS data, the classification of other objects stayed a very welcome feature as it meant additional balancing of the datasets and it could be picked up during the training of the network without much additional work once the data has been properly labelled. So this issue stayed a Should have, which we also fulfilled in some of our experiments.
- *Should try different machine learning algorithms* - At first we shall focus on only one machine learning methodology, but it would be extremely useful if we'd manage to compare multiple machine learning algorithms and to conclude which would be more suited for our application;
  During the project, we have continuously kept multiple networks up to date to work with the data we had at the current state of the project, so this continuously stayed a high priority in our project which is why this kept on being a Should have throughout our whole project.
- *The result quality should get 75%* - Improving the quality from an average of 60% to 75% would mean a much more accurate identification of features, which in turn would give a much more trustworthy result.
  With the reduced priority for achieving a 60% accuracy rate the 75% accuracy rate also reduced in importance. In the end, it was still something that would be nice to strive towards but it was never a priority for us anymore which is why we removed it from the MoSCoW entirely.

3. ***COULD***

- *Could implement the code as ArcGIS plugin* - We will start by creating a piece of software which takes a PC and labels it, but it would be amazing if we could integrate this piece of software as a plugin in ArcGIS, which would drastically improve the usability and would allow Esri users use our software in their specific applications. We had multiple reasons of why we chose to go with ArcGIS instead of other similar pieces of software, besides the fact that a client for this project is Esri: it is one of the most used programs in practice and, because it requires a paid subscription, it is well maintained and updated (as compared to Open-Source projects, which are mainly maintained by community volunteers). At the same time, Esri constantly provides up-to-date data to all its customers, and the GUI is easy to use and understand;
  In the end, this task became less of a focus for our project. The integration with ArcGIS turned out to be a quite labour intensive process, so the decision was made to visualize our results as a web application instead which was advised by one of our mentors.

Using this web application our process could still be easily visualizable for clients of our service and the output data is fully compatible with ArcGIS. So the task itself was modified somewhat but it still stayed a Could have.

- *ArcGIS output* - adapting the result such that it complies with the ArcGIS standards;

  As said in the previous point the output data we produce is fully compatible with ArcGIS so this Could have has been fulfilled and stayed a Could have.

- *Output: meshes, LoD representations* - Applying multiple methods of transforming the groups of point clouds (grouped by individual objects) into either triangulation meshes, or perhaps even trying to reconstruct the object as a representation having a certain Level of Detail.

  This idea kept in our heads but in the end, there was simply no time to complete it, so it stayed a Could have.

4. ***WON'T***

   - *Label the training* - we believe that labelling the training data ourselves might be too difficult, and would require a lot of time and effort, so we have decided to try and find only already-labelled data in our training process.

     This changed drastically during the project. At first, we did not want to do this for reasons shown above, then we changed this line of thought when we got the NS data and wanted to only label the assets we were interested in to reduce the amount of time spent on this task as well as the difficulty of it in general, but after results with this line of thinking, we needed to switch to different labelling approaches for different kinds of experiments which indeed was both difficult and time-consuming. In the end our only WON'T have become one of our most needed MUST have's.

A.4 Work Plan

As already said our project didn't entirely go as planned initially due to more chances arising and shifting of directions for the project. This meant that the work plan we set up at the start of the project (see Figure 37) was not going to hold up anymore. This planning was originally already not holding up because we had expected to have some data from a third party to use in our experiments by the end of sprint 3, but due to COVID 19 this process was delayed a lot causing the third party to change into the NS and the data was only available to us since sprint 6. As a result, our project became very busy in the last weeks to make sure everything still went according to plan. These changes of our planning are shown in Figure 38. Both of these schematics follow a similar colour coded structure as can be seen in the legend. When looking at this planning the changes in our project can be seen quite well in the blocks that suddenly introduced a lot of new issues after the midterm presentation and once again in sprint 6. This meant fewer opportunities for demo's and a lot more research as well. The planning at the end of the project, therefore, became a lot busier than initially thought as well which caused some long evenings but in the end, the team managed to pull through.

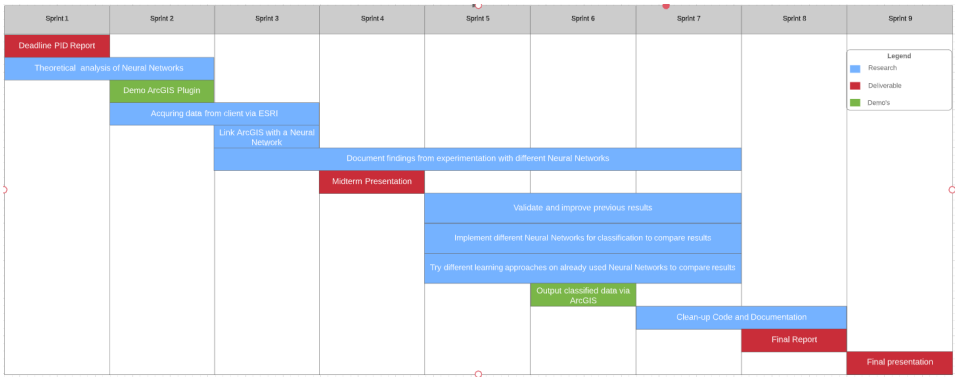


Figure 37: The Gantt Chart that visualizes our originally thought timeline during the project

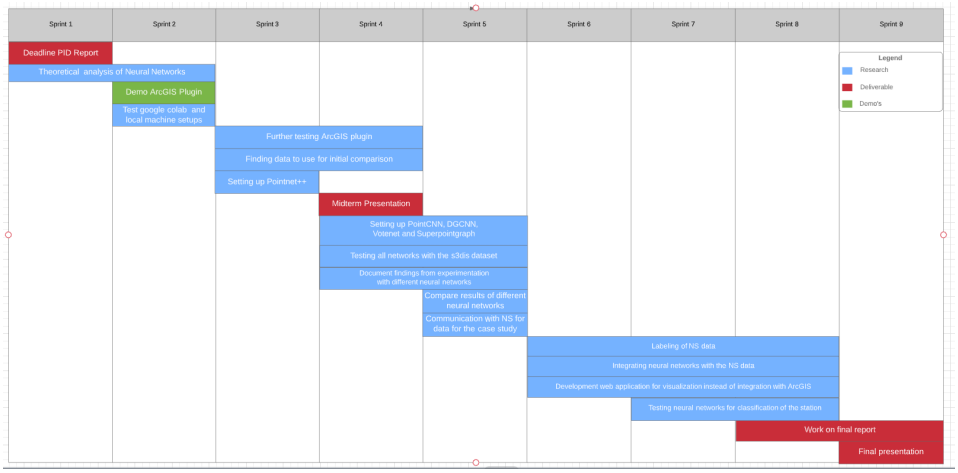


Figure 38: The updated Gantt Chart that visualizes our project actually went

A.5 Rich Picture

To explain the scope of this project comprehensively, we have created a rich picture diagram. It depicts the relationship between the parties involved in the project, the problem case that needs to be solved, the expectation of the different parties, the processes that need to be done, and also the possible obstacles during the project. In the rich picture (see Figure 39), our project is defined by three stakeholders: the team members, the clients, and the mentors. Together, we want to do research focusing on indoor point cloud classification using deep learning algorithms. The interests and concerns of each party are stated in the different colours in the diagram. Firstly, our team will be focusing on the processing algorithm and the web development. Secondly, the client-side will provide supporting materials needed for this project, such as data sets and learning materials. Finally, the mentors will guide us with the knowledge to trigger our creativity to work on this project.

Throughout the project, this rich picture has stayed fairly consistent as the parties have all upheld their parts in the project. The only change made was to the clients part as the third party, the NS, was introduced into our project. This added a question from their side about the possibilities for asset management into our project.

Figure 39: Rich picture

B Limitations

This section describes some results encountered during the workflow, including the hardware, the usability of ArcGIS Pro Beta, ArcGIS Pro integration, issue with third party data, and Coronavirus pandemic.

B.1 Hardware

For this project, only one PC was available for all tasks since Google Colab does not work efficiently for all tasks. Installation in Google Colab often experienced incompatible packages. Google Colab itself already provides almost all requirement for the neural networks, but the version is often too up-to-date. Some packages can indeed be changed manually, but some others are too complicated to edit. For example, when setting up PointCNN in Google Colab, the user should downgrade both the TensorFlow and the CUDA version which leads into an unsatisfied path directory, leading into yet another error. Besides, it is also difficult to debug in Google Colab.

As an input for the next deep learning project, it is better to have more than one PC since the training process takes time. Also, the labelling process requires more suitable hardware due to heavy data set to be visualised at a time, even though the data had been truncated into smaller blocks.

B.2 Problems with ArcGIS Pro for Labeling

During the project, we had to label quite a lot of data in the end. To do this we were allowed to use the beta version of ArcGIS 2.6. ArcGIS allowed us a lot of freedom in carefully classifying points and modifying class codes in batches for instance, but not everything went smoothly during the process. The laptops we were using to run ArcGIS Pro were all at recommended settings or just a bit lower for version ArcGIS 2.5 so the performance should be smooth, but we experienced that when visualizing a dense patch of points which have a lot of points very close to each other or when visualizing just a lot of points in general spread over a bigger area that some problems started arising. Firstly the graphic cards we have in our laptop could simply not keep up with the visualization, even when changing the settings of ArcGIS Pro to the lowest possible ones from a visualization point of view. This resulted in long loading times to see your selection for labelling at certain moments and sometimes crashed the program as a whole. Because of the intensity on which our GPU's were running it also caused some of our laptops to overheat at times. We also experienced the issue of not all points being visualized at times, which caused some selection during manual classifications to be inaccurate as some points which had to be labelled were not shown and therefore fell outside of the selection range. This caused some badly labelled objects in our initial results, but after finding them they have been fixed. Another problem was with the loading of the datasets. As our data from the NS was about 18 GB it was too large to efficiently visualise in ArcGIS Pro, we only learned about this later during a meeting with the client so this caused some delay in our work. To make sure this became less of an issue we split the dataset into 26 parts initially and afterwards we split these in 5 by 5-meter tiles to greatly reduce the amount of data loaded into ArcGis at the same time.

B.3 Poor Results

This section provides some results from other experiments besides the one we delivered in the main text. The TensorBoard plots are smoothed to give an understanding for training in general.

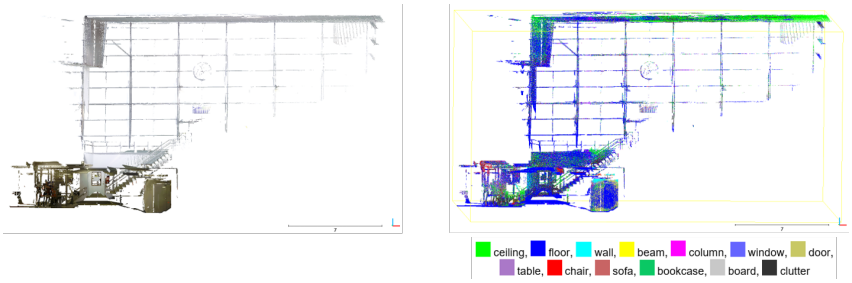


Figure 40: Result of applying pretrained model from S3DIS directly to Amersfoort data set. We notice that the pretrained model is not compatible to be used in our data set since the objects are totally different.

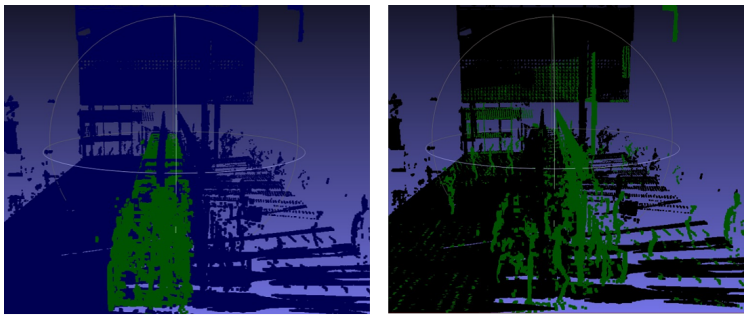


Figure 41: First try on Amersfoort data using PointCNN. Entrance gates is the only labeled object in this scene. As we can see, the prediction detect the other side of the entrance gate. After this experiment, we decided to label entrance gate for both side.

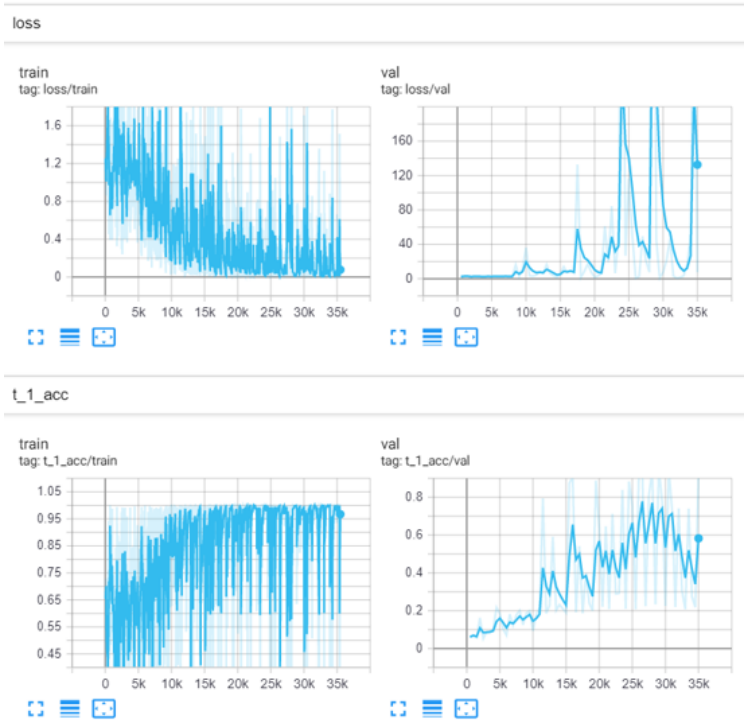


Figure 42: Rough loss and accuracy during training and validation process in the initial running.

Class Accuracy	Unclassified	Garbage can	Couch	Entrance gate	Floor	SOS pole	Info boards
PointNet++	0.94	0.00	0.07	0.00	0.00	0.00	0.00
PointCNN	0.83	0.00	0.00	0.00	0.00	0.00	0.00

Table 7: Initial result of Amersfoort data set using PointNet++ and PointCNN. All classes barely have true predictions, except unclassified.

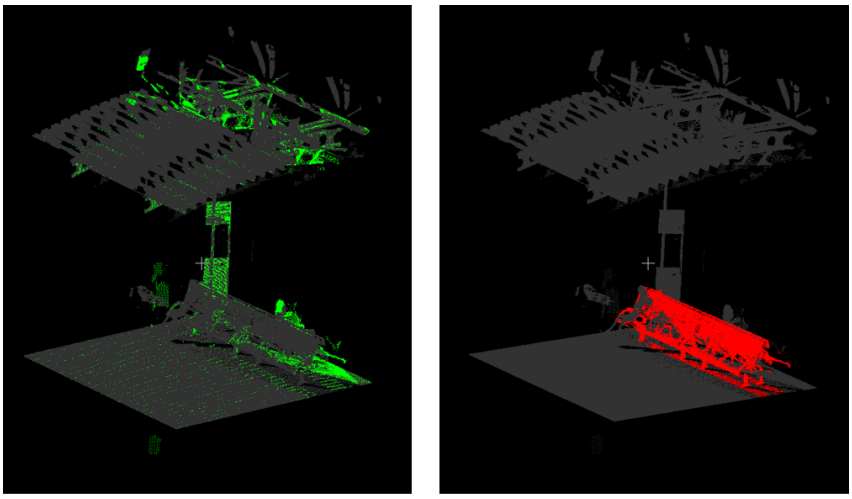


Figure 43: Visualisation of the initial result on the couch. No couch detected in the prediction model. Some parts are predicted as garbage can.

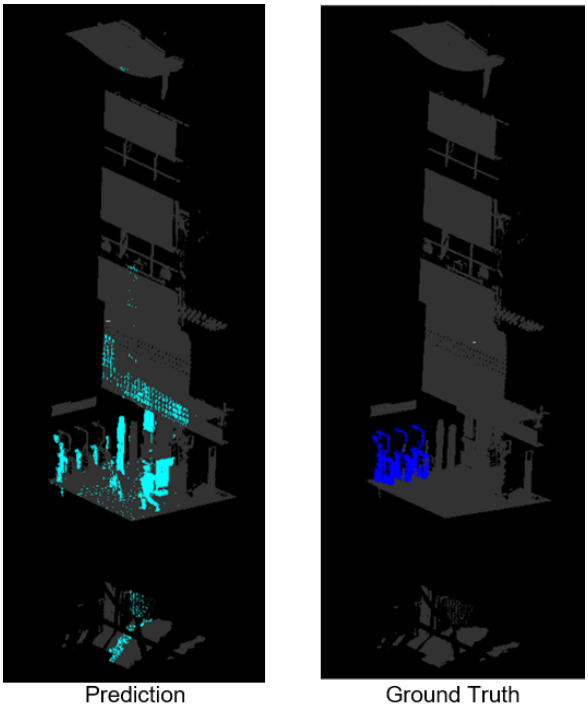


Figure 44: Visualisation of the initial result on the entrance gate, but no entrance gate classified correctly in the prediction.

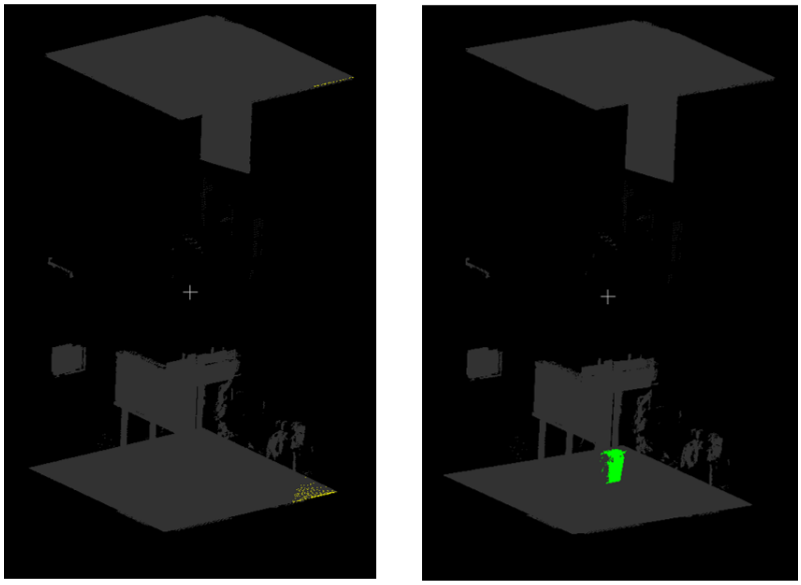


Figure 45: Visualisation of the initial result on the garbage can. Prediction result depicts some yellow eggs on the floor and ceiling.

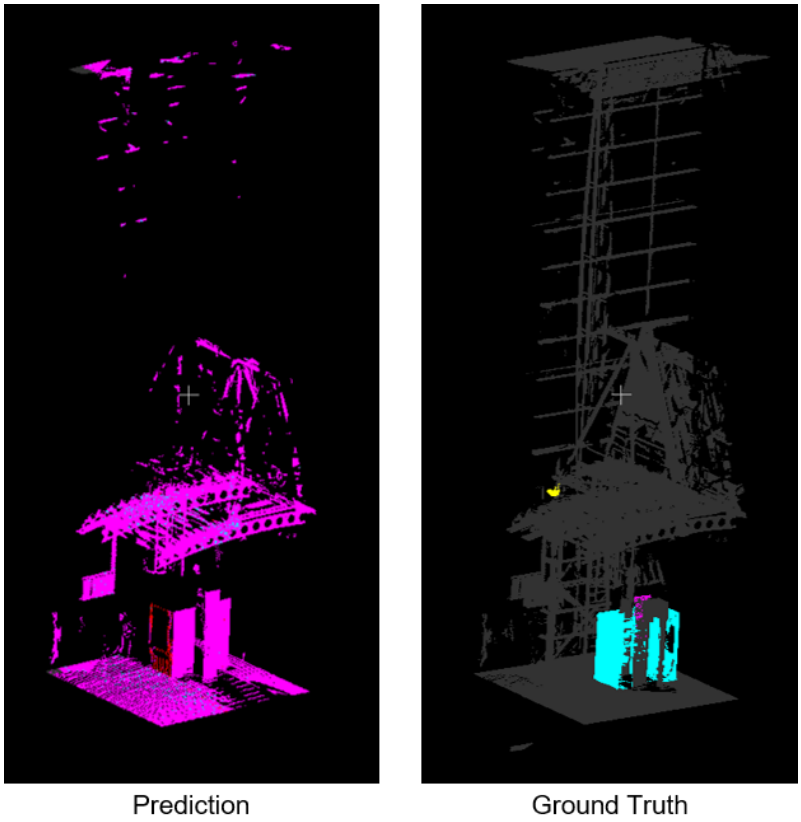


Figure 46: Visualisation of the initial result on the vending machine. Most points are predicted as ticket machines.

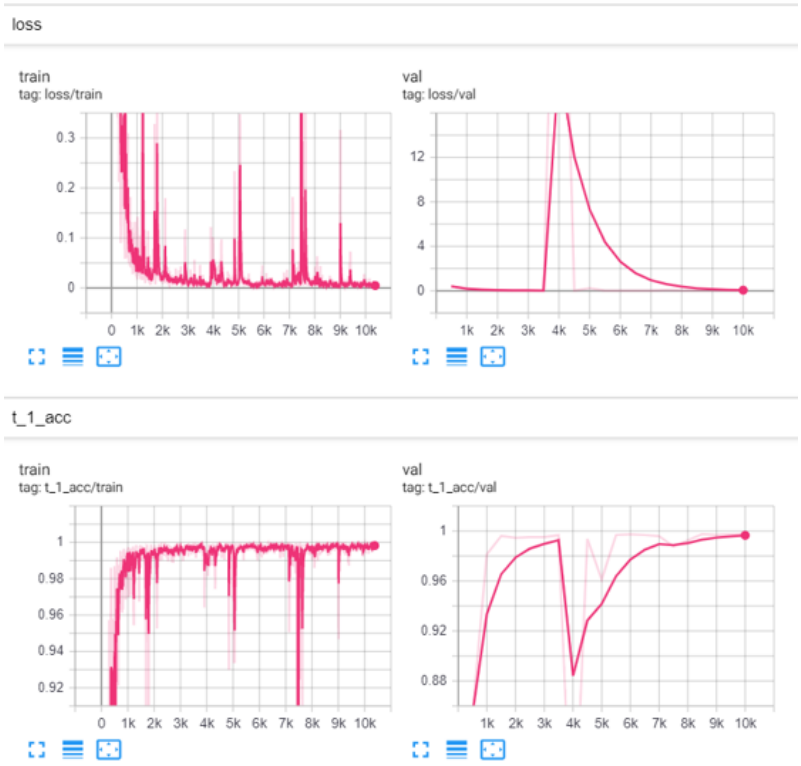


Figure 47: Loss and accuracy during training and validation process using only one small scene having entrance gate, floor, SOS pole, and advertisement board on the ground.

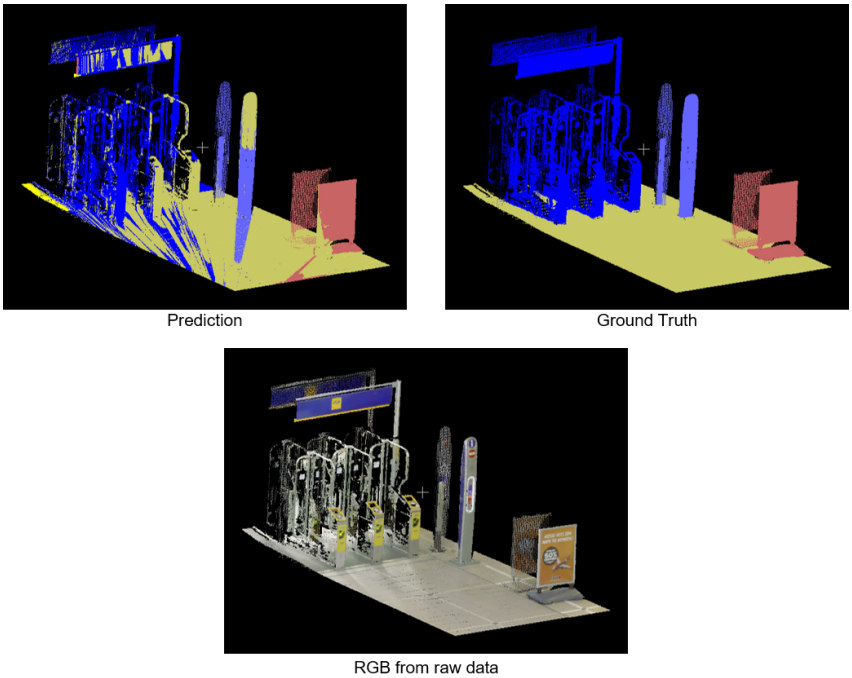


Figure 48: Visualisation of the first trial. Training only on one scene with additional class: floor, SOS pole, and advertisement board on the ground. Notice that the shadow shown in the raw data are represented in the prediction result.

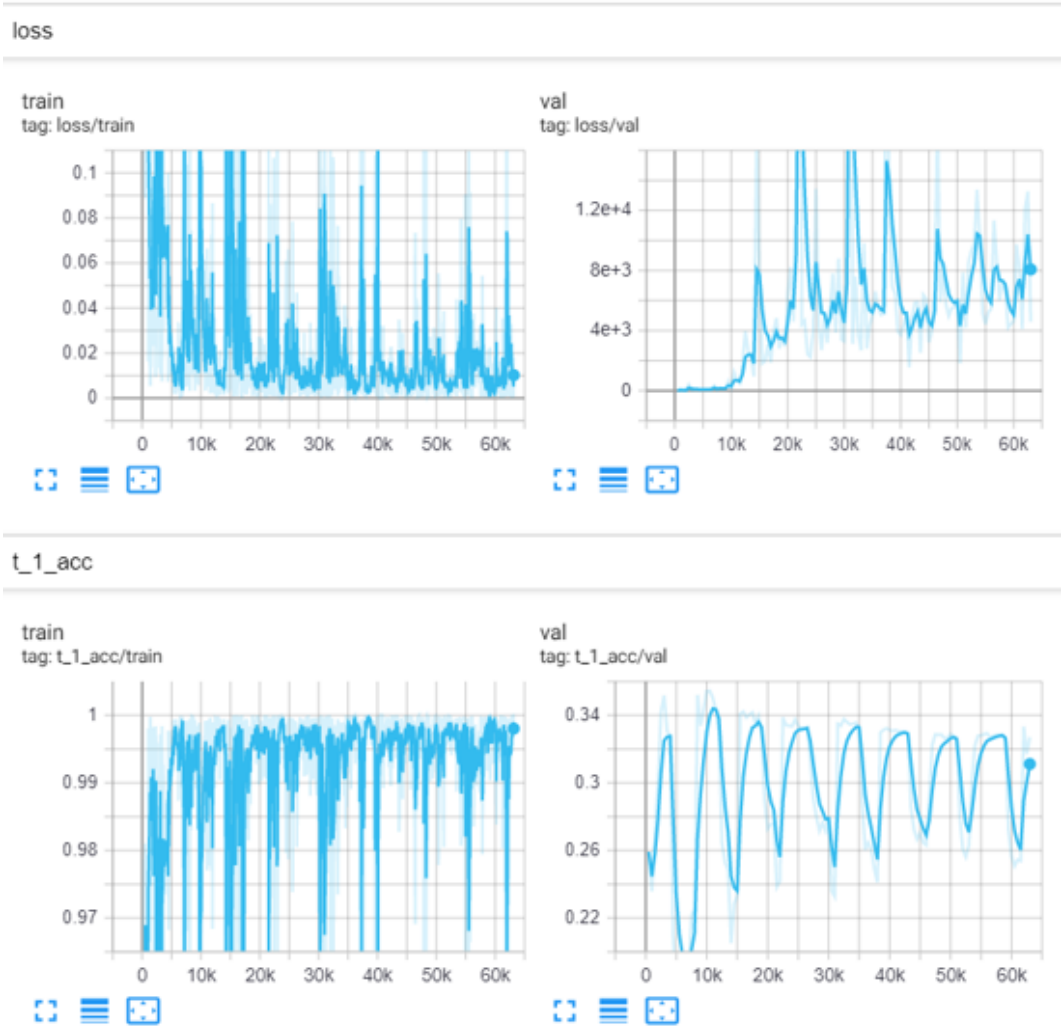


Figure 49: Loss and accuracy during training and validation process on the large scenes with a full data set.

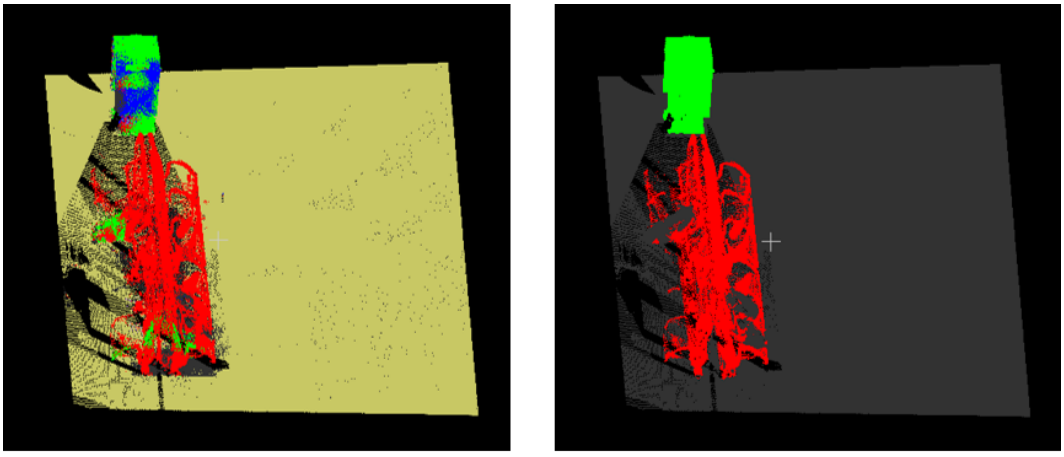


Figure 50: Visualisation of the prediction of other results in the large scenes. Some parts of garbage can are predicted as entrance gate and some parts of couch are detected as garbage can. There is color difference in the floor due to unlabeled floor in the ground truth.

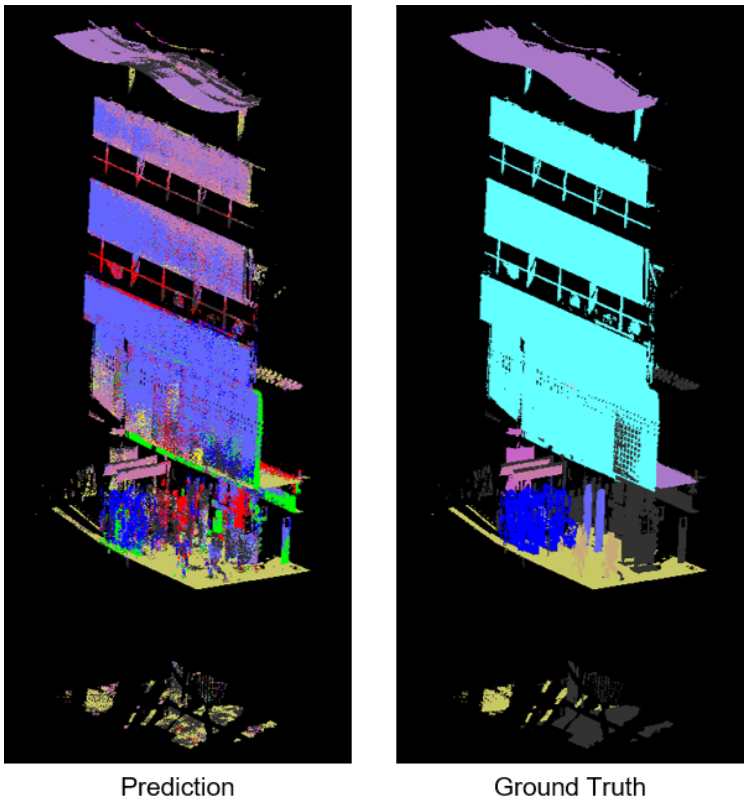


Figure 51: Visualisation of the predictions of other results in the large scenes. As we can see, prediction for ceiling is quite well and also floor. The model could predict parts of entrance gate correctly. However, most walls are predicted as ceiling and other objects are classified randomly.

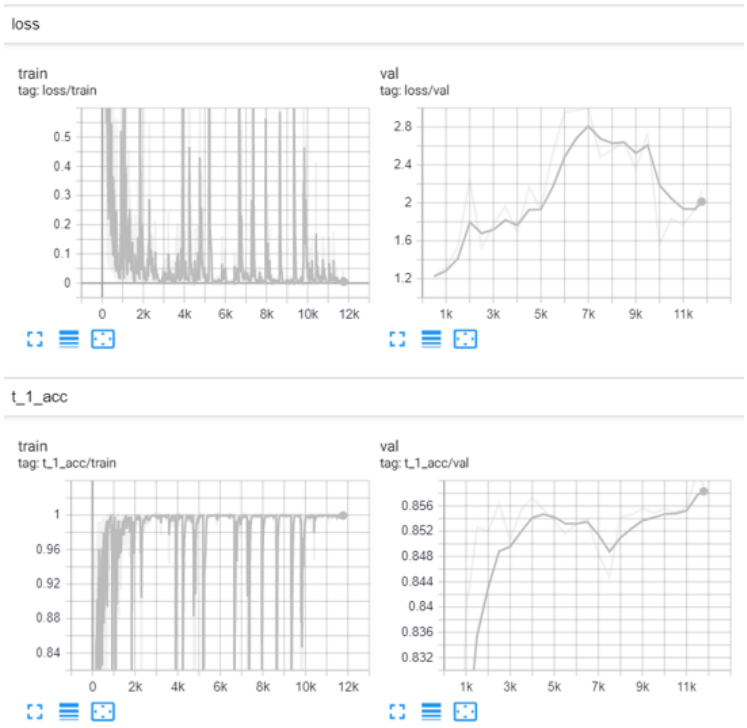


Figure 52: Loss and accuracy during training and validation process focused only on the yellow egg.

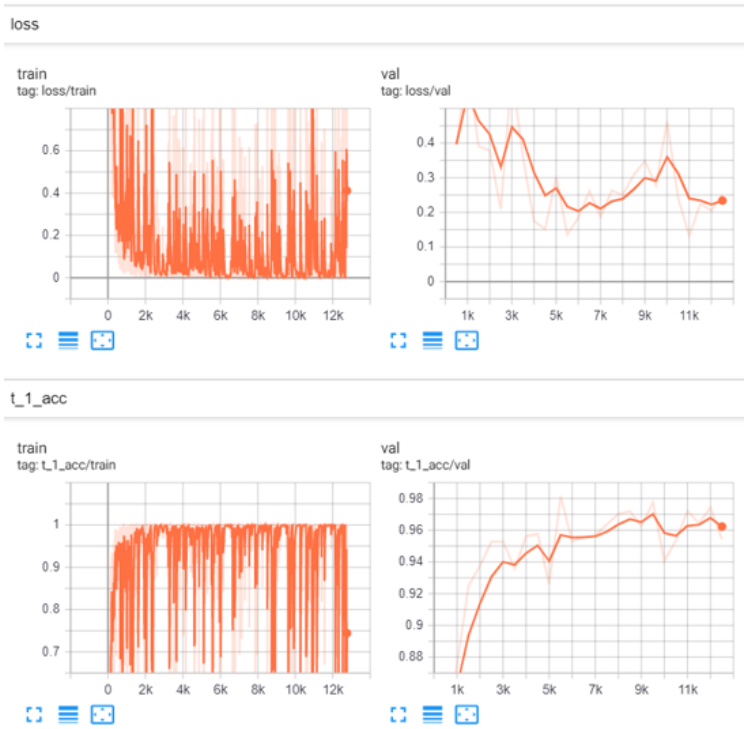


Figure 53: Loss and accuracy during training and validation process focused only on the yellow egg.

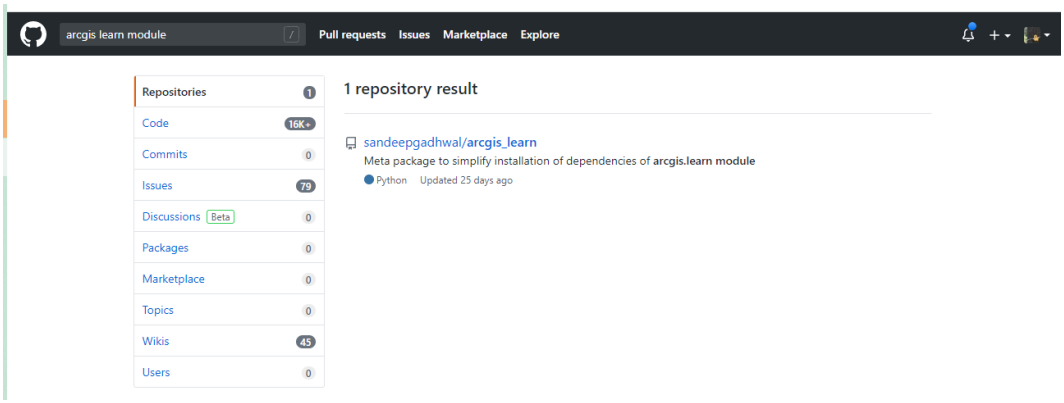


Figure 54: The open-source projects using arcgis.learn module in GitHub

B.4 ArcGIS Pro Integration Problems

- ArcGIS API for Python: the $arcgis.learn$ module provides functions that support machine learning and deep learning with spatial data. Currently, the learning module focuses on remote sensing satellite imagery. But they do include PointCNN for point cloud segmentation. However, various required python libraries and dependencies must be installed to compile the PointCNN framework. It also requires Visual C++14 and CUDA Toolkit.

  Python environment:

```
conda create -n pytorcharcgis conda activate pytorcharcgis
    conda install -c conda-forge requests-toolbelt requests_ntlm
     conda install -c esri -c fastai -c pytorch arcgis fastai
    =1.0.54 pytorch=1.1.0 cudatoolkit=10 scikit-image pillow
    =6.2.1 fastprogress=0.1.21 --no-pin

conda install -c esri -c plotly -c owlas laspy=1.6.0 plotly
    =4.5.0 plotly-orca psutil h5py=2.10.0 transforms3d pip
    install --no-binary torch-cluster==1.4.5 torch-cluster
    ==1.4.5 --no-cache-dir pip install --no-binary torch-sparse
    ==0.4.3 torch-sparse==0.4.3 --no-cache-dir pip install --no-
    binary torch-scatter==1.4.0 torch-scatter==1.4.0 --no-cache-
    dir pip install --no-binary torch-geometric==1.3.2 torch-
    geometric==1.3.2 --no-cache-dir

```

  In order to avoid some underlying conflicts, xiaoai Li reinstalled the operation system of her PC. Then she followed all the instructions. But the compiling of PointCNN framework still failed.

- ArcGIS Pro version conflicts: another option of using ArcGIS API for Python is to copy the python environment of the installed ArcGIS Pro. At the start, the ArcGIS Pro with the old version (2.0) was used. However, the ArcGIS Pro with version 2.6 is already in beta stage. The Python environment from ArcGIS Pro 2.0 was not updated which results in a lot of conflicts.
- Few related study materials: the module of $arcgis.learn$ is relatively newly added, especially for the PointCNN framework. There are few tutorials about how to use this learn module. In GitHub there is nearly no open-source project using $arcgis.learn$ module (see Figure 54).
- Operation system conflicts: PointCNN framework in ArcGIS API for Python has some unfixed problems for use. But we have already had a workable PointCNN neural network which is available to use. The workable PointCNN runs on Linux system including some Linux-specific Cuda and cuDNN configuration. To use this neural network, Xiaoai Li installed the Linex system on her PC. However, developing an add-on in ArcGIS Pro becomes impossible since ArcGIS Pro only has a Windows version.

B.5 Third party data

Another setback during the project has been the third party to some extent. There have been a lot of delays when it came to this data, from the expected sprint 3 to sprint 6 which essentially is a third of the project later. This delay made our planning quite hard to do as this delay kept happening gradually. The initial thought had been that we would be able to work with data provided by Schiphol in sprint 3 so we planned on this. This was then pushed to a week later, then it became unsure if this could be an option, so other options were considered, this then also took another week to become clear until the NS was finally chosen. This is obviously part of working with external companies and we fully understand this but it was still a setback in our project which caused quite a delay. Another downside of this third party data was afterwards that the data was not labelled when we eventually got it. This made the experience we got and the workflow we had to take closer than would usually be the case when working with a company, but this caused us to have to pick up the labelling ourselves, which is something we put in our won't haves for the project initially as we believed this would take way to much time to do properly. In the end, it can indeed be seen in our results that this is the case if no special precautions are taken. This is also a nice conclusion to have at the end of the project but it was still something that we had rather not seen happen in the project as the cleaning of the data so late in the project gave us a significant amount of extra pressure we had to deal with.

B.6 Travel Issues

Due to the current worldwide situation caused by the Coronavirus pandemic, one member of the team (Mihai-Alexandru Erbașu) lost a few working days. While trying to get home, his flight was cancelled and rescheduled a few times, sometimes with less than 12 hours notice, time which could have been spent on the project. Also, besides the days spent on preparing for the flight (even though it was cancelled in the end), another full working day was lost on the travelling day itself.

C Data Preparation

Convert .e57 to .las

```
e572las -v -i Amersfoort.e57 -o Amersfoort.las
```

Split the data into several blocks.

```
las2las -i amersfoort.las -keep_xy -5 20 25 30 -o merged_1.las
```

-keep_xy -5 20 25 30 (res is in merged_1)	-keep_xy 25 20 55 30 (res is in merged_14)
-keep_xy -5 10 25 20 (res is in merged_2)	-keep_xy 25 10 55 20 (res is in merged_15)
-keep_xy -5 0 25 10 (res is in merged_3) empty result	-keep_xy 25 0 55 10 (res is in merged_16)
-keep_xy -5 -10 25 0 (res is in merged_4)	-keep_xy 25 -10 55 0 (res is in merged_17)
-keep_xy -5 -20 25 -10 (res is in merged_5)	-keep_xy 25 -20 55 -10 (res is in merged_18)
-keep_xy -5 -30 25 -20 (res is in merged_6)	-keep_xy 25 -30 55 -20 (res is in merged_19)
-keep_xy -5 -40 25 -30 (res is in merged_7)	-keep_xy 25 -40 55 -30 (res is in merged_20)
-keep_xy -5 -50 25 -40 (res is in merged_8)	-keep_xy 25 -50 55 -40 (res is in merged_21)
-keep_xy -5 -60 25 -50 (res is in merged_9)	-keep_xy 25 -60 55 -50 (res is in merged_22)
-keep_xy -5 -70 25 -60 (res is in merged_10)	-keep_xy 25 -70 55 -60 (res is in merged_23)
-keep_xy -5 -80 25 -70 (res is in merged_11)	-keep_xy 25 -80 55 -70 (res is in merged_24)
-keep_xy -5 -90 25 -80 (res is in merged_12)	-keep_xy 25 -90 55 -80 (res is in merged_25)
-keep_xy -5 -100 25 -90 (res is in merged_13)	-keep_xy 25 -100 55 -90 (res is in merged_26)

Table 8: Area division based on xy values.

After all the data has been cut into several areas as shown in Table 8 it was then cut in 5 by 5 meters blocks. The result was a scene cut in a lot of small cubes which can be seen in Figure 55.

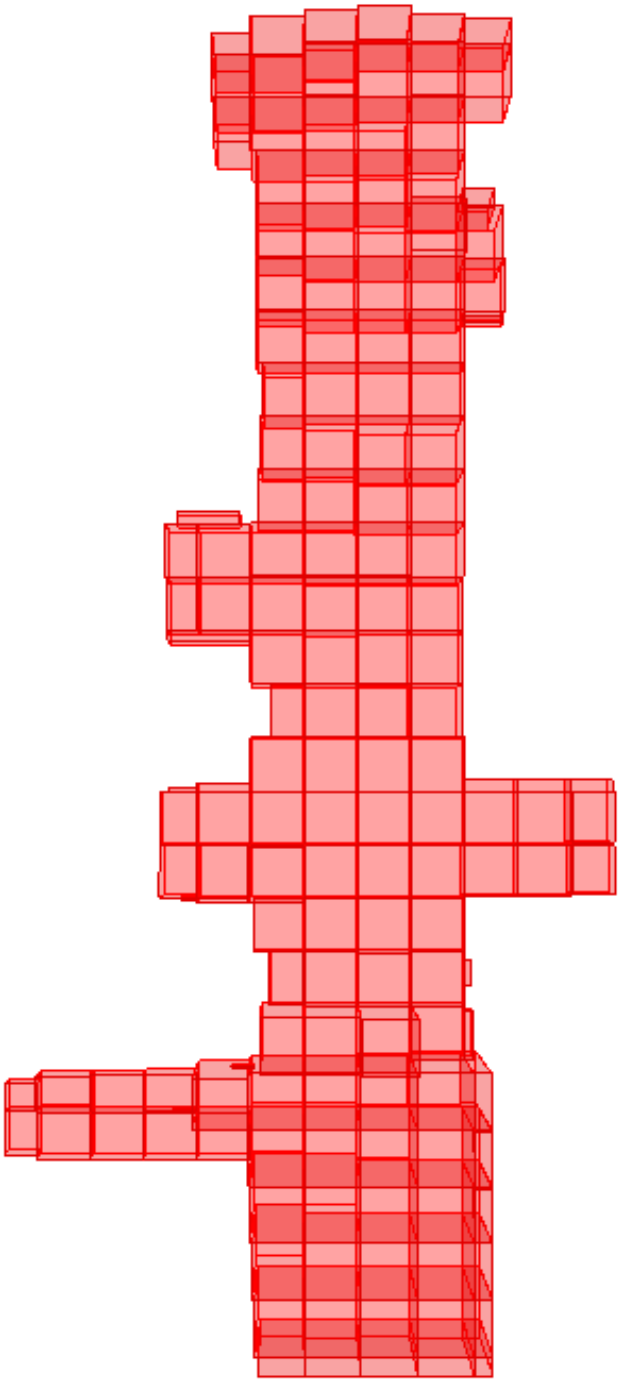


Figure 55: An overview from the top of how the station has been cut up into pieces for easier data processing

Index	Class	Scan num	Partition
0	unclassified		
1	yellow egg	15	17-6; 24-2; 26-1; 26-2; 26-3
2	garbage can		14-6; 15-1; 18-2; 18-5; 20-2; 22-8; 23-2; 23-6
3	couch		18-2; 18-5; 20-5; 23-2; 23-6;
4	ticket machine	03 11 15 16	14-6; 14-7 17-2; 17-6 24-2 26-2
5	vending machine	15	24-2
6	entrance gate		12-1; 12-2; 17-1; 17-2; 17-5; 17-6; 25-1; 25-2; 25-3; 25-5; 25-6; 25-7

Figure 56: List of identified objects per class per scene.

Two parts below are intentionally developed according to input format for PointCNN data preparation, but we make it compatible with PointNet++ and DGCNN as well.

```
# Convert .las to text file format having xyz, rgb, and label
    information.
las2txt -i amersfoort.las -o amersfoort.txt -sep space -parse xyzRGBc
    -scale_rgb_down
```

```
# Separate text files into .txt (xyzrgb) and .labels (class codes).
#!/bin/sh
ls *.las | sed -e 's/.las//g' > flist

for f in `cat flist`
do
    echo Processing $f ...
    /mnt/c/programs/LAStools/bin/las2txt.exe -i $f.las -o $f.txt -sep
    space -parse xyzRGBc -scale_RGB_down
    awk 'NF-=1' $f.txt > $f\_xyzrgb.txt
    awk '{print $(NF)}' $f.txt > $f\_xyzrgb.labels
    rm $f.txt
    echo $f DONE
done
mv *.txt ../prepare_xyzrgb_labels
mv *.labels ../prepare_xyzrgb_labels
```

D Data Labeling

First you install ArcGIS obviously and start it. Then we follow the following steps:

1. Make a new project (using a local scene for instance (note different scenes or maps have different properties)).
2. In the catalogue on the right navigate to the "Folders" folder and right-click, then navigate to the folder containing the wanted data and add it to the project.
3. In this folder, you can then make a LAS Dataset if this is wanted to combine multiple LAS Files into one. This can also give them a joined Coordinate Reference system (both for a horizontal and vertical projection (in our case these were RD New and NAP))
4. Then you can right-click on the dataset you wanted and add it to the current map or scene.
5. Now the data will be displayed on the middle screen which allows you to interact with the scene itself.
6. To make classification of points easier you can then click on the left on the 3d las layer which gives you the LAS Dataset Layer options in the ribbon, here you can go to classification and click on "Create" in the profile viewing section.
7. Once this option has been selected you can then start reclassifying the points by zooming in on the points and selecting them using the select tool in the classification tab.

E Setting Up the Environment for Neural Networks

E.1 How to setup PointNet++

Requirements:

```
ubuntu 16.04
python 3.6.7
pytorch 1.1.0
plyfile
```

Data Preparation

For PointNet++, make the following changes to take the new dataset:

1. In file S3DISDataLoader.py change the class number: search and replace every 13 with 7; replace every 14 with 8.
2. In file train_semseg.py change the class name list: used to be 13 classes; change into 7 classes, optionally change the names.
3. In file train_semseg.py change the –npoint argument: make sure the number is less or equal with the minimal number of points in one scene.
4. Change the npy filenames with area index, e.g., Area_1_somename, so that the code knows whether it should be used for training or testing.
5. In file train_semseg.py change the –test_area argument: make it compatible with the area partition of our dataset.

Code:

```
import os
import sys
import numpy as np

base_dir = "/media/zhaiyu/7CF2DC06F2DBC296/Datasets/Amersfoort"
filenames_train = os.listdir('/media/zhaiyu/7CF2DC06F2DBC296/Datasets/Amersfoort/train')
filenames_train_data = [filename for filename in filenames_train if filename.endswith('.txt')]
filenames_train_label = [filename[:-11] + ".labels" for filename in filenames_train_data]
filenames_output = [filename[:-4] + '.npy' for filename in filenames_train if filename.endswith('.txt')]

# only for training dataset
for filename_data, filename_label, filename_output in zip(filenames_train_data, filenames_train_label, filenames_output):
    print("processing " + filename_data)
    if os.path.exists(os.path.join(base_dir, 'data_pointnet2', filename_output)):
        print("skipping existing file")
        continue
    train_label = np.loadtxt(os.path.join(base_dir, "train", filename_label)).reshape([-1, 1])
    train_data = np.loadtxt(os.path.join(base_dir, "train", filename_data))
    # merged_15_4 has only 1 point
    if len(train_label) < 2:
        print("skipping the file with #point<2")
        continue
    npy_data = np.concatenate([train_data, train_label], axis=1)
    np.save(os.path.join(base_dir, 'data_pointnet2', filename_output), npy_data)
```

Run

```
## Check model in ./models
## E.g. pointnet2_ssg
python train_semseg.py --model pointnet2_sem_seg --test_area 5 --log_dir pointnet2_sem_seg
python test_semseg.py --log_dir pointnet2_sem_seg --test_area 5 --visual

Visualization results will save in log/sem_seg/pointnet2_sem_seg/visual/ and you can visualize these .obj file by MeshLab.
Visualization Using show3d_balls.py

## build C++ code for visualization
cd visualizer
bash build.sh
## run one example
python show3d_balls.py
```

E.2 How to setup Superpoint Graph

I followed the installation guide they provided on their GitHub when installing the network. This guide is found at https://github.com/mlpc-team/superpoint_graph in the readme section. My pc is running windows as well btw so this section is based on the windows system.

First I went and installed Conda via Miniconda, which is a free minimal Conda installer found at https://conda.io/en/latest/miniconda.html, as this was needed in a later step anyway. After installing this please make sure that the miniconda3 folder and the
Scripts folder are added to your PATH found in your environment variables. (After adding these to your environment variables you might have to restart your computer).

For the conda environment, I made a new environment at the location I wanted by running the command

```
conda create --prefix=yourEnvName python=x.x
```

(with yourEnvName as the name you want to use and x.x as the correct version of python, in my case these were sp_graph_env and 3.7)

I then ran the following command to activate the correct environment to work in.

source active sp_graph_env/

and then I used cd sp_graph_env/ to enter this folder.

Although I did make a fork the installation of the cut-pursuit submodule did not go as expected on the forked repo and failed so, therefore, I advise to use the cloning approach they advise on their own repo. That means running the command:

```
git clone --recurse-submodules https://github.com/loicland/
    superpoint_graph
Here the --recurse-submodules command makes sure cut-pursuit is
    installed correctly, making sure you can skip step 4 of their own
    installation guide.
```

The installation guide then says to run the command

```
pip install git+https://github.com/pytorch/tnt.git@master
```

To install PyTorch and torchnet which need to be installed as a requirement. My computer, however, had a problem with this as it did not have the module named tools.nnwrap. So I installed the framework in another way. I ran the command

```
conda install pytorch torchvision cudatoolkit=10.0 -c pytorch
(cudatoolkit=10.0 as we discussed this during the meeting on
    15-05-2020)
Not the newest install
pip install torch===1.5.0 torchvision===0.6.0 -f https://download.
    pytorch.org/whl/torch_stable.html
```

Which I found on the PyTorch website https://pytorch.org/get-started/locally/ when I filled in my system specs and my package manager as pip. This takes a while to install.

For torchnet I went to https://pypi.org/project/torchnet/#modal-close and under download files I downloaded the "torchnet-0.0.4.tar.gz" file and then I ran the command

```
pip install file:///C:\Users\\TUDelftSID\\Downloads\\torchnet-0.0.4.
    tar.gz
```

Where the last part is an absolute path to the tar.gz file on my pc.

It then asks to download some additional python packages with the command

```
pip install future python-igraph tqdm transforms3d pynvrtc fastrlock
    cupy h5py sklearn plyfile scipy
In my case, however, cupy didn't want to install directly so I removed
     this one from the install command and manually installed it doing
     the following.
```

Then I ran the command:

```
conda install -c anaconda cupy
```

Then some more packages needed to be installed via conda with the command

```
conda install -c anaconda boost; conda install -c omnia eigen3; conda
    install eigen; conda install -c r libiconv
```

And then you run the following command to make the cmake dependencies. Make sure the -DPYTHON_LIBRARY points to your local python library file, -DPYTHON_INCLUDE_DIR points to your local python version and -DEIGEN3_INCLUDE_DIR to your eigen3 clone.

```
cd partition/ply_c; cmake . -DPYTHON_LIBRARY=/usr/lib/x86_64-linux-gnu
    /libpython3.6m.so -DPYTHON_INCLUDE_DIR=/usr/include/python3.6m -
    DBOOST_INCLUDEDIR=/usr/include -DEIGEN3_INCLUDE_DIR=/usr/include/
    eigen3; make; cd ..; cd cut-pursuit; mkdir build; cd build; cmake
    .. -DPYTHON_LIBRARY=/usr/lib/x86_64-linux-gnu/libpython3.6m.so -
    DPYTHON_INCLUDE_DIR=/usr/include/python3.6m -DBOOST_INCLUDEDIR=/
    usr/include -DEIGEN3_INCLUDE_DIR=/usr/include/eigen3; make;
```

Then to use their own pretrained network use the following code for which you manually pass the –S3DIS_PATH which is the path to the data you want to pass to the network and the –odir which should be where you want to store the output from the network.

```
for FOLD in 1 2 3 4 5 6; do \
CUDA_VISIBLE_DEVICES=0 python learning/main.py --dataset s3dis --
    S3DIS_PATH /content/gdrive/My\ Drive/mlpc-team/Neural\ Networks/
    superpoint_graph --cvfold $FOLD --epochs -1 --lr_steps '[275,320]'
     \
--test_nth_epoch 50 --model_config 'gru_10_0,f_13' --ptn_nfeat_stn 14
    --nworkers 2 --pc_attribs xyzrgbelpsvXYZ --odir "results/s3dis/
    pretrained/cv${FOLD}" --resume RESUME; \
done
```

E.3 How to setup PointCNN

Environment for PointCNN is based on Github: https://github.com/yangyanli/PointCNN. We adapted some parts of codes for the compatibility within our NS data sets. All modified code for NS is available in https://github.com/mlpc-team/PointCNN-NS.

Requirements / packages:

```
tensorflow-gpu 1.8 with CUDA 10.x
matplotlib
plyfile
scipy
tqdm
transforms3d
```

Folder structure in PointCNN (based on semantic3d data structure):

```
.
├── PointCNN
    └── data
        ├── amersfoort
            ├── filelists
            ├── test
                ├── amersfoort_area3_xyzrgb.txt
                ├── amersfoort_area3_xyzrgb.labels
                ├── amersfoort_area4_xyzrgb.txt
                └── amersfoort_area4_xyzrgb.labels
            ├── train
                ├── amersfoort_area1_xyzrgb.txt
                ├── amersfoort_area1_xyzrgb.labels
                ├── amersfoort_area2_xyzrgb.txt
                └── amersfoort_area2_xyzrgb.labels
            ├── val
                ├── amersfoort_area3_xyzrgb.txt
                └── amersfoort_area3_xyzrgb.labels
            ├── test_files.txt
            ├── train_data_files.txt
            ├── val_data_files.txt
        └── other data sets
```

Data preparation for NS data (adapted from semantic3d data preparation)

```
#1 Input .txt and .labels containing xyzrgb and class codes
    respectively. Distribute the data into training, validation, and
    testing sets according to folder structures.
#2 Go to /data_preparation and run prepare_ns_data_xyzrgb.py to
    convert input files into HDF format (.h5) within six data
    dimension (xyzrgb). If user wish to use XYZ only or add intensity,
     then run prepare_ns_data_xyz.py or prepare_ns_data_xyzirgb.py
#3 Run prepare_ns_filelists.py to create a list of files having
    distributed data.
```

User may adjust parameter setting for training. The file located under pointcnn_seg/ns_x8_2048_fps.py. The parameter used in this study follows the semantic3d setting, but adapted at some parts as follow:

```
num_class = 18
batch_size = 4
num_epochs = 1024
decay_steps = 5000
data_dim = 6
```

Setup PointCNN for NS:

```
# Compile sampling; make sure that CUDA version and tensorflow path
    directory alredy satisfy tf_sampling_compile.sh. This step is only
     taken once for initial setup. Afterwards, directly go to the next
     step.
cd PointCNN/sampling
bash tf_sampling_compile.sh
# Prepare dataset, put all input according to folder structure in
    PointCNN/data/
cd PointCNN/data_conversions
python prepare_ns_data.py --folder ../../data/amersfoort/
mkdir filelists
python prepare_ns_filelists.py --folder ../../data/amersfoort/
#Run train and val
cd ../
python train_val_seg.py --filelist data/amersfoort/train_data_files.
    txt
                        --filelist_val data/amersfoort/val_data_files.
    txt
# Run test
python test_general_seg.py --filelist data/amersfoort/test_files.txt
                           --load_ckpt models/seg/pointcnn_seg_ns/
    ckpts/iter-
# Run evaluation
cd evaluation/
# Merge files
python ns_merge.py --datafolder ../data/amersfoort/test
# Calculate accuracy and IoU
python eval_ns.py --data ../data/amersfoort/test
# Additional: convert numpy format (.npy) to text file
python ns_npy2obj.py --data ../data/amersfoort/test
```

Code: ns_npy2obj.py

```
#!/usr/bin/python3

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import os
import numpy as np


DEFAULT_DATA_DIR = '../data/amersfoort_xyzrgb_v1/test/'

p = argparse.ArgumentParser()
p.add_argument(
    "-d", "--data", dest='data_dir',
    default=DEFAULT_DATA_DIR,
    help="Path to S3DIS data (default is %s)" % DEFAULT_DATA_DIR)

args = p.parse_args()

label2color = [[50,50,50],    #clutter
               [255,255,0],   #yellow egg
               [0,255,0],     #garbage can
               [255,0,0],     #couch
               [255,0,255],   #ticket machine
               [0,255,255],   #vending machine
               [0,0,255],     #entrance gate
               [200,200,100],#floor
               [100,100,255],#other machine
               [200,100,100],#adv board
```

```
                [200,100,200],#pillars
                [100,255,255],#walls
                [100,255,100],#digital info monitor
                [170,120,200],#station ceiling
                [170,200,120],#escalator
                [120,170,200],#lamps
                [120,200,170],#stairs
                [200,120,170],#info signs in the air
                [200,170,120]]#people

path = args.data_dir

categories_list = [filename[:-4] for filename in os.listdir(args.
    data_dir) if filename.endswith('.txt')]
print(categories_list)

for category in categories_list:
  path_pred_label = os.path.join(args.data_dir, category+'_pred.npy')
  path_gt_label = os.path.join(args.data_dir, category+'.labels')
  path_scene = os.path.join(args.data_dir, category+'.txt')

  pred = np.loadtxt(path_pred_label)
  gt = np.loadtxt(path_gt_label)
  scene = np.loadtxt(path_scene)

  fout = open(os.path.join(args.data_dir, category+'_pred.txt'), 'w')
  fout_gt = open(os.path.join(args.data_dir, category+'_gt.txt'), 'w')

  for j in range(scene.shape[0]):
    color = label2color[int(pred[j])]
    color_gt = label2color[int(gt[j])]
    fout.write('%f %f %f %d %d %d\n' % (scene[j, 0], scene[j, 1],
    scene[j, 2], color[0], color[1], color[2]))
    fout_gt.write('%f %f %f %d %d %d\n' % (scene[j, 0], scene[j, 1],
    scene[j, 2], color_gt[0], color_gt[1], color_gt[2]))

  fout.close()
  fout_gt.close()
```

E.4 How to setup DGCNN (in Google Colab)

1. Connect to GPU in Collab
2. Change runtime to GPU
3. Unzip the zipped archive (in sample_data, for example)
4. Create data folder in dgcnn
5. Put the lower Standford 3d dataset in data
6. After it loads for a bit, we get an error: what we should to do solve it, is copy the contents of indoor3d_sem_seg_hdf5_data into a new folder, which we will call indoor3d_sem_seg_hdf5_data_test. Run it again one more time
7. The results will now be saved to the folder checkpoints.

F Web application

The code is hosted in GitHub and the **code structure** in depth of 4 is as follows:

```
.
├── babel.config.js
├── package.json
├── public
│   ├── favicon.ico
│   └── index.html
├── README.md
├── server
│   ├── app.py
│   ├── db.sqlite
│   └── scipoc
│       ├── data_conversions
│       │   ├── prepare_s3dis_data.py
│       │   ├── prepare_s3dis_filelists.py
│       │   └── prepare_s3dis_label.py
│       ├── data_utils.py
│       ├── evaluation
│       │   ├── eval_s3dis.py
│       │   └── s3dis_merge.py
│       ├── __init__.py
│       ├── pointcnn.py
│       ├── pointcnn_seg
│       │   ├── s3dis_x8_2048_fps.py
│       │   ├── test_s3dis.sh
│       │   └── train_val_s3dis.sh
│       ├── pointcnn_seg.py
│       ├── pointfly.py
│       ├── s3dis_npy2obj.py
│       ├── sampling
│       │   ├── LICENSE
│       │   ├── tf_sampling_compile.sh
│       │   ├── tf_sampling.cpp
│       │   ├── tf_sampling_g.cu
│       │   ├── tf_sampling_g.cu.o
│       │   ├── tf_sampling.py
│       │   └── tf_sampling_so.so
│       └── train_val_seg.py
├── src
│   ├── App.vue
│   ├── assets
│   │   └── cover.jpg
│   ├── components
│   │   ├── Segmentation.vue
│   │   ├── Upload.vue
│   │   └── ViewPointCloud.vue
│   ├── main.js
│   ├── router
│   │   └── index.js
│   └── views
│       ├── Home.vue
│       ├── PointCloudView.vue
│       ├── SegmentationView.vue
│       └── UploadView.vue
└── vue.config.js
```

The required **Python** environment:

```
name: scipoc
channels:
  - anaconda
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1=main
  - _tflow_select=2.1.0=gpu
  - absl-py=0.9.0=py36_0
  - astor=0.8.0=py36_0
  - blas=1.0=mkl
  - c-ares=1.15.0=h7b6447c_1001
  - ca-certificates=2020.4.5.1=hecc5488_0
  - certifi=2020.4.5.1=py36h9f0ad1d_0
  - click=7.1.2=py_0
  - cudatoolkit=9.0=h13b8566_0
  - cudnn=7.6.5=cuda9.0_0
  - cupti=9.0.176=0
  - cycler=0.10.0=py36_0
  - dbus=1.13.14=hb2f20db_0
  - expat=2.2.6=he6710b0_0
  - flask=1.1.2=py_0
  - flask-cors=3.0.8=py_0
  - fontconfig=2.13.0=h9420a91_0
  - freetype=2.9.1=h8a8886c_1
  - gast=0.3.3=py_0
  - glib=2.63.1=h3eb4bd4_1
  - grpcio=1.27.2=py36hf8bcb03_0
  - gst-plugins-base=1.14.0=hbbd80ab_1
  - gstreamer=1.14.0=hb31296c_0
  - icu=58.2=he6710b0_3
  - intel-openmp=2020.1=217
  - itsdangerous=1.1.0=py36_0
  - jinja2=2.11.2=py_0
  - jpeg=9b=h024ee3a_2
  - kiwisolver=1.2.0=py36hfd86e86_0
  - ld_impl_linux-64=2.33.1=h53a641e_7
  - libedit=3.1.20181209=hc058e9b_0
  - libffi=3.3=he6710b0_1
  - libgcc-ng=9.1.0=hdf63c60_0
  - libgfortran-ng=7.3.0=hdf63c60_0
  - libpng=1.6.37=hbc83047_0
  - libprotobuf=3.11.4=hd408876_0
  - libstdcxx-ng=9.1.0=hdf63c60_0
  - libuuid=1.0.3=h1bed415_2
  - libxcb=1.13=h1bed415_1
  - libxml2=2.9.9=hea5a465_1
  - markdown=3.1.1=py36_0
  - markupsafe=1.1.1=py36h7b6447c_0
  - matplotlib=3.1.3=py36_0
  - matplotlib-base=3.1.3=py36hef1b27d_0
  - mkl=2020.1=217
  - mkl-service=2.3.0=py36he904b0f_0
  - mkl_fft=1.0.15=py36ha843d7b_0
  - mkl_random=1.1.1=py36h0573a6f_0
  - ncurses=6.2=he6710b0_1
  - numpy=1.18.1=py36h4f9e942_0
  - numpy-base=1.18.1=py36hde5b4d6_1
  - openssl=1.1.1g=h516909a_0
  - pcre=8.43=he6710b0_0
  - pip=20.0.2=py36_3
  - plyfile=0.7.2=pyh9f0ad1d_0
  - protobuf=3.11.4=py36he6710b0_0
```

```
  - pyparsing=2.4.7=py_0
  - pyqt=5.9.2=py36h05f1152_2
  - python=3.6.10=h7579374_2
  - python-dateutil=2.8.1=py_0
  - python_abi=3.6=1_cp36m
  - qt=5.9.7=h5867ecd_1
  - readline=8.0=h7b6447c_0
  - setuptools=47.1.1=py36_0
  - sip=4.19.8=py36hf484d3e_0
  - six=1.14.0=py36_0
  - sqlite=3.31.1=h62c20be_1
  - tensorboard=1.9.0=py36hf484d3e_0
  - tensorflow=1.9.0=gpu_py36h02c5d5e_1
  - tensorflow-base=1.9.0=gpu_py36h6ecc378_0
  - tensorflow-gpu=1.9.0=hf154084_0
  - termcolor=1.1.0=py36_1
  - tk=8.6.8=hbc83047_0
  - tornado=6.0.4=py36h7b6447c_1
  - transforms3d=0.3.1=py_0
  - werkzeug=1.0.1=py_0
  - wheel=0.34.2=py36_0
  - xz=5.2.5=h7b6447c_0
  - zlib=1.2.11=h7b6447c_3
prefix: /home/xiaoaili/anaconda3/envs/scipoc
```

The **JavaScript** dependencies:

```
{
  "name": "scipoc-vue-app",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "axios": "^0.19.2",
    "bootstrap": "^4.5.0",
    "bootstrap-vue": "^2.15.0",
    "core-js": "^3.6.5",
    "esri-loader": "^2.14.0",
    "jquery": "^3.5.1",
    "three": "^0.117.1",
    "three-orbitcontrols": "^2.110.3",
    "tween": "^0.9.0",
    "vue": "^2.6.11",
    "vue-router": "^3.2.0"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "~4.4.0",
    "@vue/cli-plugin-eslint": "~4.4.0",
    "@vue/cli-plugin-router": "~4.4.0",
    "@vue/cli-service": "~4.4.0",
    "@vue/eslint-config-airbnb": "^5.0.2",
    "babel-eslint": "^10.1.0",
    "eslint": "^6.7.2",
    "eslint-plugin-import": "^2.20.2",
    "eslint-plugin-vue": "^6.2.2",
    "vue-template-compiler": "^2.6.11"
  },
  "eslintConfig": {
    "root": true,
    "env": {
      "node": true
    },
    "extends": [
      "plugin:vue/essential",
      "@vue/airbnb"
    ],
    "parserOptions": {
      "parser": "babel-eslint"
    },
    "rules": {}
  },
  "browserslist": [
    "> 1%",
    "last 2 versions",
    "not dead"
  ]
}
```