

Technische Universität München

Ingenieur fakultät Bau Geo Umwelt

Lehrstuhl für Computergestützte Modellierung und Simulation

# **Data Preparation for the Evaluation of Point Cloud Segmentation by Deep Learning**

Bachelorthesis

für den Bachelor of Science Studiengang Umweltingenieurwesen

Autor: Simon-Hauke Wichmann

1. Betreuer: Prof. Dr.-Ing. André Borrmann

2. Betreuer: M. Sc. Yuandong Pan

Ausgabedatum: 12. April 2021

Abgabedatum: 12. September 2021

## Abstract

Scan-to-BIM is a process of creating Building Information Model of existing buildings and sights. The semantic information must be extracted from the scan, which is generally represented as a 3D point cloud.

The presented work evaluates the data preparation for point cloud segmentation by deep learning. In the preparation process, a point cloud is segmented and classified in order to create different data sets that serve as input for a neural network. The neural network KPConv is then used to evaluate the performance on these data sets.

The goal of this thesis is to evaluate the effects of different segmentation and classification approaches on the performance of the neural network. Varying the data set and the parameters of the neural network, multiple experiments are conducted to compare the results and thus draw conclusions on how to best prepare a point cloud for the training of a neural network for point cloud segmentation.

## Zusammenfassung

Scan-to-BIM ist ein Prozess, bei dem ein Building Information Model für Bestandbauten erstellt wird. Dabei werden die semantischen Informationen aus den gescannten Daten extrahiert, die meist als Punktwolke vorliegen.

Die hier präsentierte Arbeit untersucht die Datenaufbereitung für die Punktwolken-Segmentierung durch Deep Learning. Bei der Datenaufbereitung wird eine Punktwolke segmentiert und klassifiziert, um daraus verschiedene Datensätze zu erstellen, die daraufhin als Eingangsdaten für ein neuronales Netz dienen. Im Anschluss wurde das neuronale Netz KPConv verwendet, um die verschiedenen Datensätze auszuwerten.

Das Ziel dieser These ist es, die Auswirkungen verschiedener Herangehensweisen für die Segmentierung und Klassifizierung auf die Leistung des neuronalen Netzes zu untersuchen. Dafür wurden die verschiedenen Datensätze und Parameter in mehreren Experimenten getestet und die Ergebnisse verglichen, um daraus stichhaltige Rückschlüsse zu ziehen, wie die Datenaufbereitung für das Trainieren eines neuronalen Netzwerkes für die Punktwolken-Segmentierung am besten durchzuführen ist.

---

## Contents

List of figures	V
List of tables	VIII
List of abbreviations	IX
1 Introduction and Motivation	1
2 Theoretical Background	2
2.1 Scan-To-BIM.....	2
2.2 Machine Learning .....	5
2.3 Neural Network .....	10
2.4 Neural Networks for Point Clouds.....	18
3 Methodology	27
4 Implementation and Experiments	30
4.1 Segmentation: Data Preparation.....	30
4.2 Variations in Data Sets and Parameters .....	37
4.3 Results.....	40
5 Discussion	46
5.1 Test Set Limitation .....	46
5.2 Specialization vs. Generalization .....	48
5.3 Subsampling Size and 'Clutter' .....	50
6 Conclusion	54
Acknowledgement	55
References	56
List of Appendices	58

## List of figures

Figure 1: Creating a BIM from a 2D-Blueprint (Borrmann, König, Koch, & Beetz, 2015, p.376) .....	3
Figure 2: The Trimble TX8 in its portable case and on the tripod (B1M, 2017).....	4
Figure 3: Render of a colorized point cloud with the software CloudCompare (point cloud collected by the Chair of Computational Modeling and Simulation) .....	4
Figure 4: Classical way of programming vs. machine learning (own representation, in context of Chollet, 2018, chap.1.1).....	6
Figure 5: Divergence vs. convergence, depending on learning rate (own representation, in context of Rebala, Ravi, & Churiwala, 2019, p.33) .....	10
Figure 6: Performance of deep learning models vs. classical machine learning models (Jha & Pillai, 2021, chap.1) .....	11
Figure 7: Input layer neuron (own representation, in context of Rebala, Ravi, & Churiwala, 2019, p.106) .....	11
Figure 8: Neuron with activation function (own representation, in context of Vasilev, 2019, sec.1, graph directly from Jha & Pillai, 2021, chap.1).....	13
Figure 9: A multi-layer fully connected network (Ramsundar & Zadeh, 2018, chap.4) .....	14
Figure 10: Convolutional layer (Jha & Pillai, 2021, chap.1) .....	14
Figure 11: Deconvolutional layer (Jha & Pillai, 2021, chap.1).....	14
Figure 12: Max-Pooling layer (Jha & Pillai, 2021, chap.1) .....	15
Figure 13: Dropout layer (Ramsundar & Zadeh, 2018, chap.4) .....	16
Figure 14: Structure of a NN (Own representation, in context of Chollet, 2018, chap.3.1) .....	16
Figure 15: Loss functions (Peltarion, 2021) .....	17
Figure 16: Point cloud, voxel representation (Vasilev, 2019, sec.4) .....	18
Figure 17: PointNet architecture (Qi, Su, Mo, & Guibas, 2017) .....	19
Figure 18: T-Net-pipeline (Vasilev, 2019, sec.4).....	20
Figure 19: Accuracy of different pooling layers (Qi, Su, Mo, & Guibas, 2017) .....	20

---

Figure 20: Hierarchical feature learning architecture of PointNet++ (Qi, Yi, Su, & Guibas, 2017).....	22
Figure 21: KPConv operating on a 2D input (Hugues, et al., 2019).....	24
Figure 22: Comparison: image convolution (left) and KPConv (right) (Hugues, et al., 2019).....	24
Figure 23: Deformable kernels (Hugues, KPConv Method).....	25
Figure 24: Different number of kernels in a stable position (left), mIoU and number of kernel points (right) (Hugues, et al., 2019).....	26
Figure 25: Holdout-Method, training data set and test data set (own representation, in context of Chollet, 2018, chap.4.1).....	27
Figure 26: IoU, intersection over union (Tiu, 2019).....	29
Figure 27:Workflow of the project (own representation).....	29
Figure 28: Point Cloud (right) as part of the building at the main campus of the Technical University (satellite image from google.de/maps).....	31
Figure 29: Raw data of the point cloud (own representation).....	31
Figure 30: Dividing the point cloud (left), zoom in on the subdivision (top right), subdivision without the outside 'noise' (bottom right).....	32
Figure 31: View of the staircase, which is unique in the point cloud, therefore chosen for training purposes only (own representation with CloudCompare).....	37
Figure 32: Path to evaluate the effects of different data sets and parameters, the numbers represent the number of the experiment.....	38
Figure 33: Split of the class 'chair' (Data Set I) into the classes 'chair' and 'office chair' (Data Set II).....	38
Figure 34: Segmentation of the object 'door' (Data Set I) into the objects 'door' and 'frame' (Data Set II).....	39
Figure 35:Subsampling size (own representation, created with CloudCompare).....	40
Figure 36: mIoU of experiment 15-26-01.....	40
Figure 37: Comparison of the mIoU_100 for experiment 15-26-01 (left) and 08-08-43 (right).....	43
Figure 38: IoU and number of objects of classes.....	47
Figure 39: Prediction of a chair, wrongfully classified as floor (preds=12) and table (preds=22), experiment 12-16-37 (Data Set I) in the middle,	

experiment 15-31-06 (Data Set II) on the right (own creation with CloudCompare)..... 47

Figure 40: Floor unit wrongfully classified as cabinet (preds=1) ..... 50

Figure 41: Subsampling size, computational time (orange, in hours), performance (blue, in percent) ..... 51

Figure 42: Predictions for printer (left) and dustbin (right) with different subsampling sizes ..... 52

---

## List of tables

Table 1: Common activation functions (own assembly of Jha et al., 2021, chap.1, graphs directly from Jha & Pillai, 2021, chap.1) .....	12
Table 2: Division and subdivisions of the point cloud .....	32
Table 3: Segmentation process (own representation using the software CloudCompare).....	34
Table 4: Overview of classes.....	36
Table 5: Results of all experiments.....	41
Table 6: Results for different data sets (in percent) .....	42
Table 7: Comparison of mIoU_100 of some classes of the experiments 15-26-01 and 08-08-43 (in percent).....	44
Table 8: Comparison of different subsampling sizes (results in percent).....	45
Table 9: Results of the class 'chair', before and after the split (in percent) .....	46
Table 10: mIoU_100 (in percent), comparison of class in Data Set I and Data Set II	49
Table 11: mIoU_100 (in percent) of storage areas .....	49



## List of abbreviations

BIM	Building Information Modelling
NN	Neural Network
CNN	Convolutional Neural Network
MLP	Multi-Layer Perceptron
kNN	$K$ nearest neighbour
MSG	Multi-Scale Grouping
MRG	Multi-resolution Grouping
KPConv	Kernel Point Convolution
IoU	Intersection-Over-Union
dI	Subsampling Size
mIoU_100	Mean Intersection over Union of the last 100 epochs
Max_IoU	Maximum Intersection over Union within the last 100 epochs

## 1 Introduction and Motivation

There are two major technological, innovative trends that, combined, create a great benefit to all stages of the construction industry. The first trend is the usage of Building Information Models (BIM) that enables architects, engineers, interior designers, manufacturers, and project managers to collaborate on the most complex projects in real-time and with unprecedented precision (Borrmann, König, Koch, & Beetz, 2015, p.V). The second trend is the handling of big datasets by using machine learning algorithms and deep learning to gather and create digital models of real-world objects and processes. This revolution of data processing finds its way into all areas of industry and helps to automate various decision tasks (Chollet, 2018, sec.1.1). One of these tasks combines the two trends mentioned above: the segmentation and classification of point clouds by deep learning. By mastering this task, Building Information Models can be created more rapidly and more cost-efficient (Johnson, 2017).

This thesis presents a project in which Kernel Point Convolution (KPCConv), a convolutional neural network for point clouds, is tested on a data set that was created by segmenting and classifying a point cloud collected by the Chair of Computational Modeling and Simulation at the Technical University of Munich. By conducting various experiments, it is tried to analyse the results and make suggestions on how to improve the data preparation process.

In the first part of this thesis, the author explains the scan-to-BIM process in order to illustrate the problem at hand. Afterwards, the author attempts to cover the fundamentals of deep learning and introduces various neural networks for point clouds.

In the second part of this thesis, the author presents his own project, which includes an extensive data preparation process in which a point cloud is segmented and classified in order to train the neural network. Subsequently, the experiments are presented, and the results are discussed.

In the end, some problems are raised, and suggestions are made on how to conduct further research on the topic.

## 2 Theoretical Background

This chapter presents the process of scan-to-BIM, explains the fundamentals of machine learning and neural networks, and gives an overview of current neural networks that can process point clouds and gather valuable information from this kind of data.

### 2.1 Scan-To-BIM

According to the US National Building Information Modeling Standard Committee, Building Information Modeling (BIM) can be understood as a “digital representation of physical and functional characteristics of a facility” (National Institute of Building Sciences (ed.), 2007, p.21). This digital model of the facility not only helps in the collaboration of many companies but also eases the consistent use of the digital information, which makes error-prone work redundant and increases productivity and quality of the building process (Borrmann, König, Koch, & Beetz, 2015, p.3).

#### Application

An increasingly important aspect of BIM is the design and construction in existing contexts (Borrmann, König, Koch, & Beetz, 2015, p.371, 372). The application in this field can be divided into several categories such as demolition, redevelopment, modernization, maintenance, and repair (Borrmann, König, Koch, & Beetz, 2015, p.371, 372). These applications incorporate existing components as well as new ones, and both must be modelled in order to use BIM for planning, constructing, and managing the build (Borrmann, König, Koch, & Beetz, 2015, p.371, 372).

For the purpose of financial and legal certainty, detailed information about the existing building must be available (Borrmann, König, Koch, & Beetz, 2015, p.373, 375). It is not an uncommon practice, though, to hand in the instructions handwritten on an as-completed drawing, which presents many challenges to the company under contract (Borrmann, König, Koch, & Beetz, 2015, p.373, 375). To create a BIM of the existing building on which construction is to be done, the 2D-drawing is scanned, and

software is used to reconstruct the building as a 3D digital representation (Borrmann, König, Koch, & Beetz, 2015, p.373, 375). An example of this is shown below.

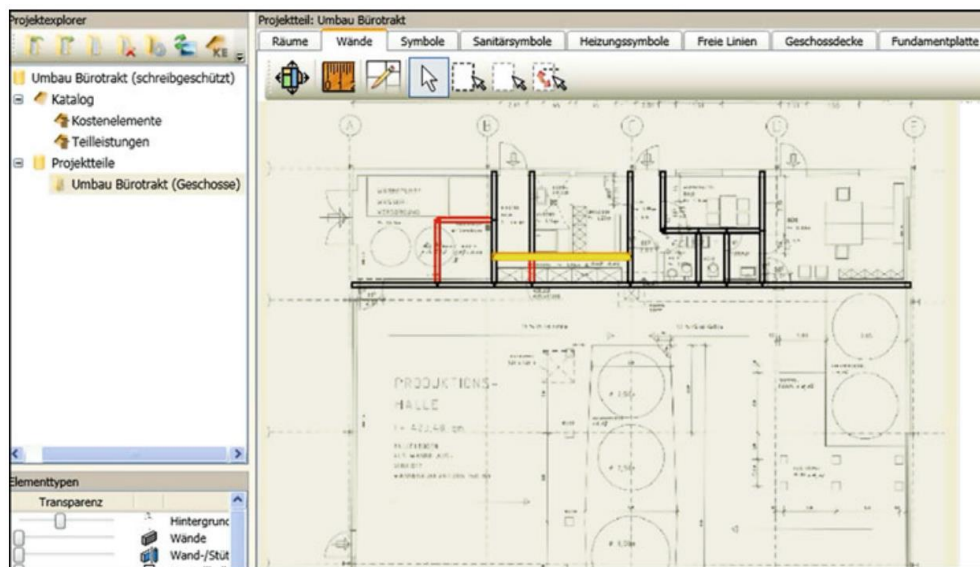


Figure 1: Creating a BIM from a 2D-Blueprint (Borrmann, König, Koch, & Beetz, 2015, p.376)

Another area in which scan-to-BIM methods are used is in the surveying and digital representation of 3D city models (Borrmann, König, Koch, & Beetz, 2015, p.177). These can not only be used for visualization and augmented reality applications but also for systems like telematics, navigation, and other complex simulations (Borrmann, König, Koch, & Beetz, 2015, p.177).

Furthermore, scan-to-BIM is not only needed to create the initial digital model but, as part of the facility management, can be used to update the data within the BIM (Borrmann, König, Koch, & Beetz, 2015, p.385).

During construction, the scan-to-BIM process can be of great benefit (Maalek, 2021, p.2). If it is done more frequently, it can not only be useful for the initial creation of the 3D representation but also for progress monitoring, dimensional quality checking, 3D BIM updating, and digital twin generation (Maalek, 2021, p.2).

### Scan-to-BIM Process

To gather the spatial data, a 3D laser scanner is set up on a tripod (B1M, 2017). The laser scanner then rotates, and the points at which the laser beam is reflected are recorded as coordinates relative to the scanner or other objects (B1M, 2017). A reasonable number of points can be called a point cloud (B1M, 2017). The surface of the objects within the scanned area can now be found in the point cloud, consisting of many individual points (B1M, 2017). At the same time as the scan, the 3D laser

scanner captures images of the scanned area (B1M, 2017). This allows the point cloud to be colorized by assigning a colour value to each point (B1M, 2017).



Figure 2: The Trimble TX8 in its portable case and on the tripod (B1M, 2017)

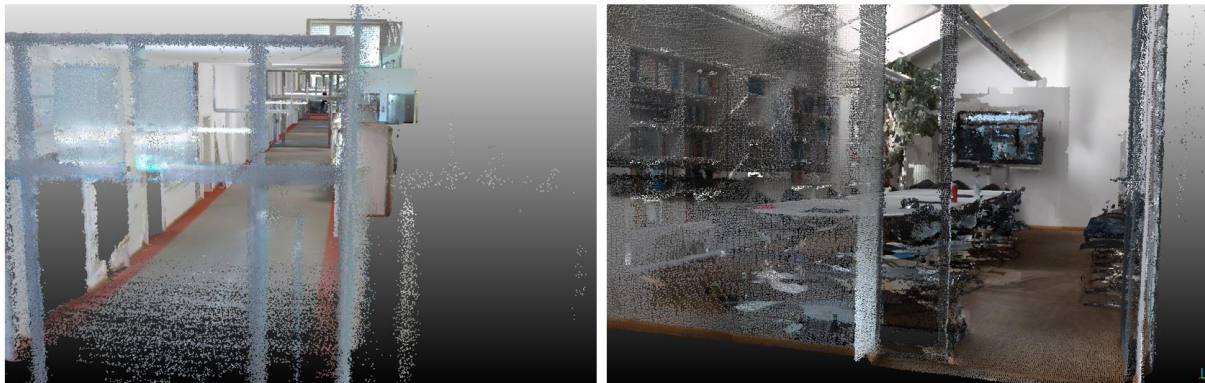


Figure 3: Render of a colorized point cloud with the software CloudCompare (point cloud collected by the Chair of Computational Modeling and Simulation)

Depending on the application purpose, the parameters for the scan can be set before scanning (B1M, 2017). One of these parameters is the *Resolution* of the scan, which determines the number of points recorded each second (B1M, 2017). This can reduce or increase the number of points in total and can affect the accuracy of the scan (B1M, 2017). Another parameter is the rotation radius, which can be set to a full circulation of 360 degrees or less (B1M, 2017). This later eases the process of only scanning a certain area that was not fully captured in a previous scan (B1M, 2017).

The scanning process can be done by one person; thus, compared to conventional surveying, the scanning part is not very laborious (B1M, 2017). Furthermore, the scanning process is less time-consuming and therefore has a monetary advantage over other surveying methods (B1M, 2017). After the data is collected, it is calibrated by software and is then ready to be shared into the project's data environment (B1M, 2017).

The challenge is to convert the point cloud into semantic information (Maalek, 2021, p.2). This process is very time-consuming if it is done manually (Macher, Landes, & Grussenmeyer, 2017, p.3). Therefore, the automation of this process is one of great interest and a prominent field of study. To further explain the details of this study, the next two sections explain the machine learning process and the use of neural networks to solve the challenge of point cloud segmentation and object classification.

## 2.2 Machine Learning

In general, one can say that machine learning is one of the building blocks that make up modern artificial intelligence research, and deep learning is a part of machine learning (Chollet, 2018, chap.1.1). With that said, the author first gives an overview of machine learning before explaining deep learning and neural networks in the subsequent section.

### Introduction to Machine Learning

“Machine Learning is about extracting knowledge from data” (Müller & Guido, 2016, chap.1). Prior to machine learning, the task of classification was accomplished more or less efficiently by using “handcoded rules of ‘if’ and ‘else’ decisions” (Müller & Guido, 2016, chap.1.1) in order to make the software more intelligent. This way of programming might be useful for domains for which the following prerequisites are true:

1. Humans have a good understanding of the problem at hand and
2. The problem to solve or the task to perform does not change over time (Müller & Guido, 2016, chap.1.1).

If one of these prerequisites does not apply, the problem becomes unsolvable, or the attempts of solving it become unsustainable because “[c]hanging the task even slightly might require a rewrite of the whole system” (Müller & Guido, 2016, chap.1.1). Instead of having to understand the process and then develop the software that encompasses all the known rules, software engineers can focus on the machine learning algorithm and then let the computer assess the data and find the rules behind it (Chollet, 2018, chap.1.1). The downside for this process is that instead of knowledge about the problem, now, software engineers need large amounts of data to feed to the algorithm (Chollet, 2018, chap.1.1).



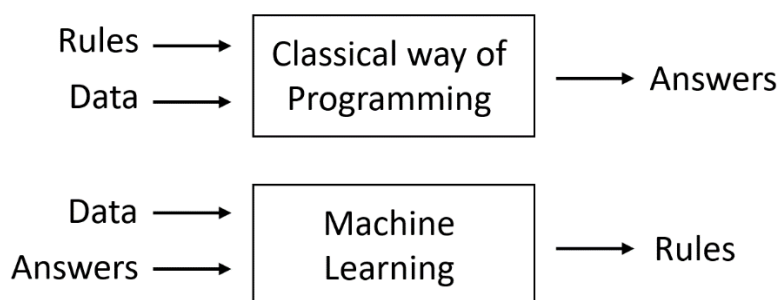


Figure 4: Classical way of programming vs. machine learning (own representation, in context of Chollet, 2018, chap.1.1)

### Machine Learning Categories

Machine Learning algorithms can be divided into two categories:

1. Supervised learning and
2. Unsupervised learning (Joshi, 2020, p.33).

Supervised learning requires a set of labelled data or samples (Joshi, 2020, p.33). This is feasible if the output is predefined (Joshi, 2020, p.33). In classification, this would include a list of examples and the corresponding classes (Joshi, 2020, p.33). Unlike unsupervised learning, the number of classes of this classification process is known beforehand (Joshi, 2020, p.33). After the learning process is completed, the model is then fed with a new dataset and can make a prediction into which class each element should be categorized (Joshi, 2020, p.33).

Unsupervised learning does not require a labelled dataset (Joshi, 2020, p.33). This kind of algorithm is often used to find some form of structure in the training data. An example of this kind of algorithm would be clustering (Joshi, 2020, p.33).

There are other machine learning algorithms, like semi-supervised learning, which uses a partially labelled dataset to complete the clustering with the corresponding names for each cluster, and reinforcement learning which is specifically designed for an ever-changing environment (Rebala et al., 2019, p.22). Since this thesis will focus on supervised learning models, the other models mentioned above will not be discussed.

There are numerous models used to solve classification and regression problems; some of them are linear regression, logistic regression, and random forest using decision trees (Rebala et al., 2019, p.25, 77).

To outline the general idea of these algorithms, the following presents an overview of the linear regression model.

### Linear Regression

The data generally consists of input data and some distinctive output (Rebala, Ravi, & Churiwala, 2019, p.26). The kind of input data for each record can be called feature, and the output is the value to be predicted (Rebala, Ravi, & Churiwala, 2019, p.26). The goal of the learning process is to find the relationship between the input data and the output, which is expressed by the coefficient for each type of input data (Rebala, Ravi, & Churiwala, 2019, p.26). A multi-dimensional linear model would therefore look like the following (Rebala, Ravi, & Churiwala, 2019, p.26):

$$y = \theta y = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \theta_3 * x_3 + \dots \quad (2.2.1)$$

$y =$  value to be predicted

$x_i =$  variable for  $i$ th feature that impacts the value of  $y$

$\theta_i =$  coefficient that determines the contribution of  $x_i$  to the value of  $y$

Choosing the linear regression model, one assumes that the relationship between the data can be represented by a linear function (Rebala, Ravi, & Churiwala, 2019, p.28). To solve the problem of finding the coefficients that result in the closest approximation of the data representation, there are two methods available: the Normal Method and the Gradient Descent Method (Rebala et al., 2019, p.28).

### Normal Method

Using the Normal Method, the hypothesis can be represented by a matrix equation (Rebala, Ravi, & Churiwala, 2019, p.28):

$$X * \theta = Y \quad (2.2.2)$$

$X =$  matrix of dimensions  $(m, n + 1)$ ;  $m =$  number of data points;  $n + 1 =$  first column with explicit feature and  $n$  features

$\theta =$  column matrix of dimension  $(n + 1, 1)$ , having values  $\theta_0$  through  $\theta_n$

$Y =$  column matrix of dimension  $(m, 1)$ ; having values  $Y_1$  through  $Y_m$

It is going to be very rare, in most cases even impossible, to find an exact solution for every coefficient  $\theta_i$ , hence it is aimed for values so that  $X * \theta$  is as close to  $Y$  as pos-



sible (Rebala, Ravi, & Churiwala, 2019, p.29). To accomplish that, the value of  $Y_p$  is introduced, which represents the predicted value  $X * \theta = Y_p$  (Rebala, Ravi, & Churiwala, 2019, p.29). The error  $(Y - Y_p)$  over all data entries is to be minimized, and since it is unknown if the error is going to be positive or negative over all the data entries, the loss function is defined as follows (Rebala, Ravi, & Churiwala, 2019, p.29):

$$J(\theta) = (Y - X\theta)^T(Y - X\theta) \quad (2.2.3)$$

Note that by multiplying the transposed column matrix with itself, the error is squared, and therefore it does not matter if the error is positive or negative (Rebala, Ravi, & Churiwala, 2019, p.29). In the next step, one uses differential calculus to minimize the loss function (Rebala, Ravi, & Churiwala, 2019, p.29). For that, one needs to find the value at which the slope of the curve is 0 (Rebala, Ravi, & Churiwala, 2019, p.29). If one derives the loss function and equates it to 0, the result is the following (Rebala, Ravi, & Churiwala, 2019, p.26):

$$2X^T X\theta - 2X^T Y = 0 \quad (2.2.4)$$

$$X^T X\theta = X^T Y \quad (2.2.5)$$

$$\theta = (X^T X)^{-1} X^T Y \quad (2.2.6)$$

With these equations, it is possible to determine the values for the column matrix  $\theta$  and therefore enables the program to predict a new outcome  $Y$  for a new data entry (Rebala, Ravi, & Churiwala, 2019, p.29). The problem with this method is that not every matrix is invertible, and even if it is, the computation of the matrix inversion  $X^T X$  might be very computationally intense (Rebala, Ravi, & Churiwala, 2019, p.29). To avoid that, one can also use the Gradient Descent Method (Rebala, Ravi, & Churiwala, 2019, p.29).

### Gradient Descent Method

When using the Gradient Descent Method, one calculates the error over all the data entries as follows (Rebala, Ravi, & Churiwala, 2019, p.30):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^i) - y^i)^2 \quad (2.2.7)$$

$h(x^i)$  = the predicted value for the  $i$ th dataset

$m$  = number of data points in the dataset

$J(\theta)$  = is the total error term of the model for all the datasets

The squaring of the error for each dataset not only ensures that the term of the sum is positive at all times, but it also gives the loss function some important properties that one can take advantage of (Rebala, Ravi, & Churiwala, 2019, p.31). The curve has only one minimum value and the slope of the curve increases, the further one is from the local minima (Rebala, Ravi, & Churiwala, 2019, p.31). Moreover, the value of the slope gives the direction in which the minimum is to be found (Rebala, Ravi, & Churiwala, 2019, p.31). This means that for any given coefficient, one might find a better fitting one if one determines the slope of the loss function for that coefficient (Rebala, Ravi, & Churiwala, 2019, p.31). Since the loss function depends on multiple variables, the partial derivative is set to (Rebala, Ravi, & Churiwala, 2019, p.31):

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h(x^i) - y^i) * x_j^i \quad (\text{for } j = 0, 1, 2, \dots, n) \quad (2.2.8)$$

Subscript  $j$  refers to the values corresponding to  $j$ th parameter

Subscript  $i$  refers to the values corresponding to  $i$ th dataset

$x_j^i$  refers to the value corresponding to  $j$ th parameter in  $i$ th dataset

After setting an initial value for each coefficient  $\theta_j$  and determining the slope of the loss function, the coefficient can be updated by the following correction (Rebala, Ravi, & Churiwala, 2019, p.32):

$$\theta_j = \theta_j - \alpha * \frac{\partial J}{\partial \theta_j} \quad (2.2.9)$$

The  $\alpha$  in this correction is also called the learning rate and determines how fast the correction is undertaken (Rebala, Ravi, & Churiwala, 2019, p.33). With that said, it should also be mentioned that  $\alpha$  is not to be chosen too large; otherwise, it could lead to overshooting, which results in a divergence instead of converging to the optimal value (Rebala, Ravi, & Churiwala, 2019, p.33).

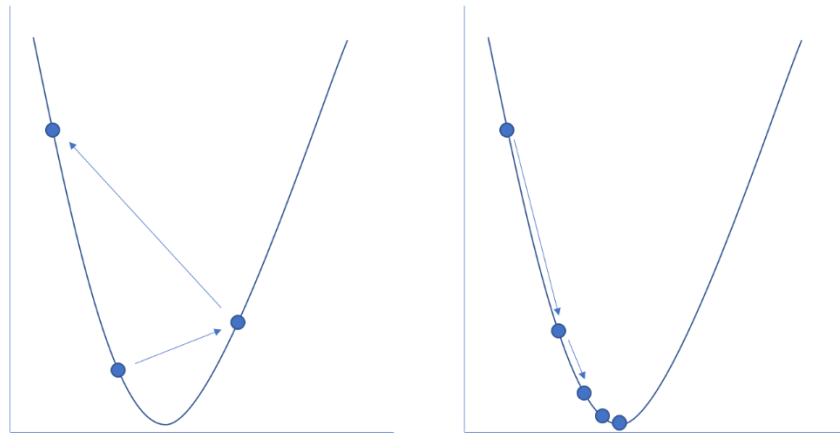


Figure 5: Divergence vs. convergence, depending on learning rate (own representation, in context of Rebala, Ravi, & Churiwala, 2019, p.33)

There are no algorithms that can determine the best value for the learning rate; trial and error is still the most used heuristic to accomplish that (Rebala, Ravi, & Churiwala, 2019, p.33).

Having explained machine learning, the next section introduces neural networks, which are a part of machine learning and use the same basic principles.

### 2.3 Neural Network

Even though it is often claimed that Neural Networks (NNs) mimic the human brain, it is important to mention that NNs are merely inspired by how the human brain works (Chollet, 2018, chap. 1.1). They are rather mathematical frameworks on how to represent data (Chollet, 2018, chap. 1.1).

NNs are used for complex applications for which machine learning does not provide a good enough accuracy (Rebala, Ravi, & Churiwala, 2019, p.103). Moreover, with a large amount of data, NNs can perform better than classical machine learning models, as seen in Figure 6 (Jha & Pillai, 2021, chap.1). Another reason for the recently gained attention of NNs is the increased computing power, following Moore's law which states that the processing power of computers is doubling every two years (Jha & Pillai, 2021, chap.1). Furthermore, machine learning is heavily dependent on feature engineering, while NNs perform well on complex problems without any hand-engineered features and thus outperform conventional machine learning algorithms (Jha & Pillai, 2021, chap.1).

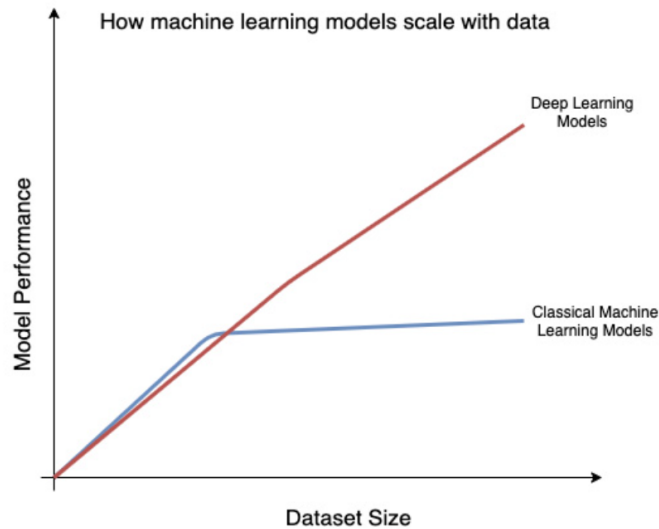


Figure 6: Performance of deep learning models vs. classical machine learning models (Jha & Pillai, 2021, chap.1)

Every NN is made up of a combination of different layers, which then makes up the architecture of this NN (Jha & Pillai, 2021, chap.1). If a NN has more than two of these layers, it is a deep learning network, also called a deep learning model (Jha & Pillai, 2021, chap.1).

### Neurons and Activation Functions

The building blocks which make up a layer is also called a neuron (Rebala, Ravi, & Churiwala, 2019, p.106, 107). These neurons can also be represented as a graph with nodes and edges (Rebala, Ravi, & Churiwala, 2019, p.106, 107). The mathematical expression  $y = m_1x_1 + m_2x_2$  can be described as shown in Figure 7,  $m_i$  being the weights and  $x_i$  being the input data values (Rebala, Ravi, & Churiwala, 2019, p.106, 107).  $y$  represents the predicted value of the model (Rebala, Ravi, & Churiwala, 2019, p.106, 107). This kind of neuron usually makes up the first layer of the NN (Rebala, Ravi, & Churiwala, 2019, p.106, 107).

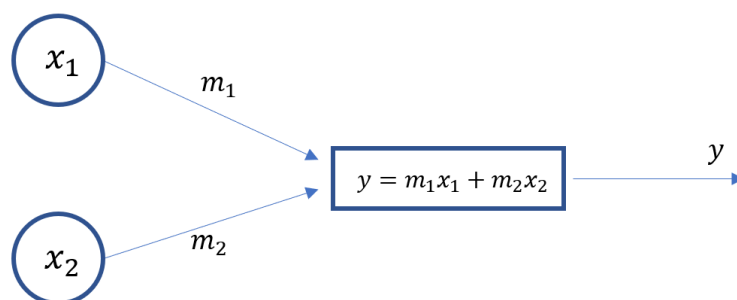
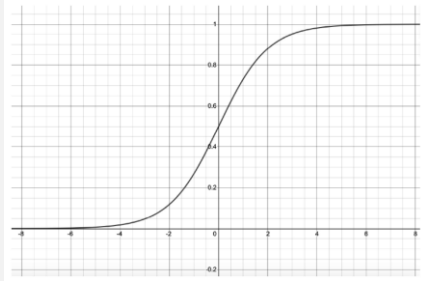
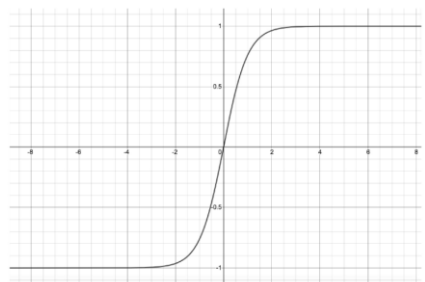
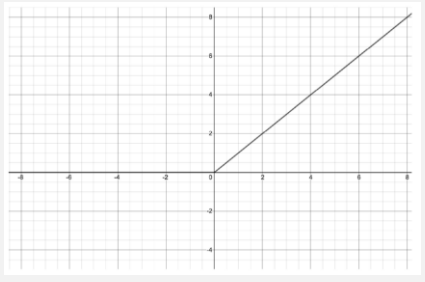
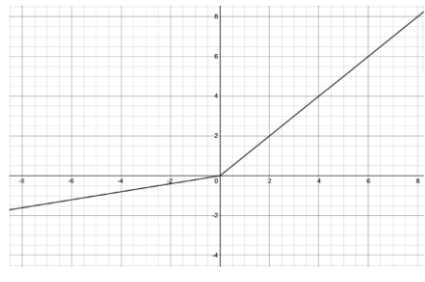


Figure 7: Input layer neuron (own representation, in context of Rebala, Ravi, & Churiwala, 2019, p.106)

To introduce a non-linearity into the model, the perceptron not only adds up the product of the input data values and the weights of the edges, but instead, a function is implemented to produce a new value (Rebala, Ravi, & Churiwala, 2019, p.107). This function is also called the activation function, and there are several types of non-linear activation functions commonly used (Rebala, Ravi, & Churiwala, 2019, p.107). Some of them are listed in the table below.

*Table 1: Common activation functions (own assembly of Jha et al., 2021, chap.1, graphs directly from Jha & Pillai, 2021, chap.1)*

Activation Function	Graph
<p><b>Sigmoid</b></p> $y = f(x) = \frac{1}{1 + e^{-x}}$	
<p><b>TanH</b></p> $y = f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
<p><b>Rectified Linear Units (ReLUs)</b></p> $y = f(x) = \max(0, x)$	
<p><b>Leaky ReLU</b></p> $y = f(x) = \max(kx, x)$	

The next layer and the following ones consist of neurons that calculate the output corresponding to their activation function (Rebala, Ravi, & Churiwala, 2019, p.106, 107). The argument for the activation function is the sum of all the outputs of the previous neurons  $x_i$ , multiplied with their corresponding weight  $w_i$ , and the bias  $b$  (Vasilev, 2019, sec.1). One of these neurons is illustrated in the figure below. In this case, the Sigmoid function is chosen as the activation function.

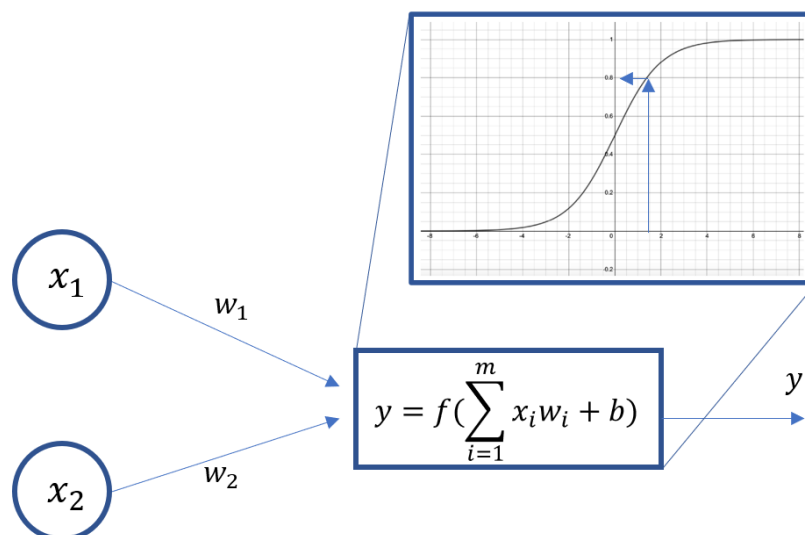


Figure 8: Neuron with activation function (own representation, in context of Vasilev, 2019, sec.1, graph directly from Jha & Pillai, 2021, chap.1)

## Layers

A layer is a module that handles data in the form of a tensor. It takes one tensor as an input and outputs another tensor (Chollet, 2018, chap.3.1). The weights of each layer are learned by using gradient descent (Chollet, 2018, chap.3.1). These weights hold therefore the knowledge of the NN (Chollet, 2018, chap.3.1).

The architecture of a NN is determined by the kind of layers and the combination of such layers (Chollet, 2018, chap.3.1). The layers can be combined into useful data transformation pipelines (Chollet, 2018, chap.3.1). It is important to note that these layers must be compatible, which means that the output of one layer must be a tensor in a shape that is accepted as input by the next layer (Chollet, 2018, chap.3.1). This part gives an overview of the commonly used layers.

Within *Fully Connected Layers*, all neurons that make up that layer are connected to all the neurons within the next layer (Jha & Pillai, 2021, chap.1). Fully connected layers are part of most deep learning classifiers (Jha & Pillai, 2021, chap.1).

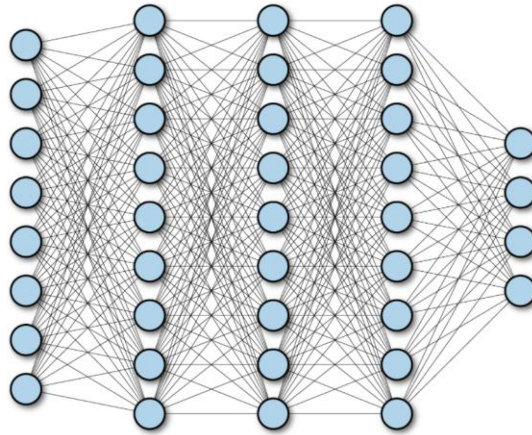


Figure 9: A multi-layer fully connected network (Ramsundar & Zadeh, 2018, chap.4)

Convolutional Neural Networks (CNN) are successfully applied for most computer vision problems (Jha & Pillai, 2021, chap.1). CNNs use *Convolutional Layers* to convolve a filter over the input (Jha & Pillai, 2021, chap.1). The following figure shows a simple example of a convolutional computation:

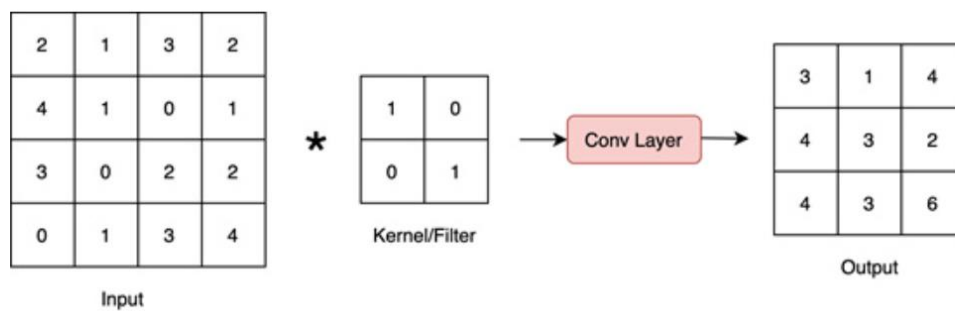


Figure 10: Convolutional layer (Jha & Pillai, 2021, chap.1)

*Deconvolutional Layers* constitute the opposite of *Convolutional Layers* in that they reverse the convolutional process (Jha & Pillai, 2021, chap.1). These kinds of layers enhance the input data and are used for networks that generate or reconstruct images (Jha & Pillai, 2021, chap.1). An example can be seen in the following diagram:

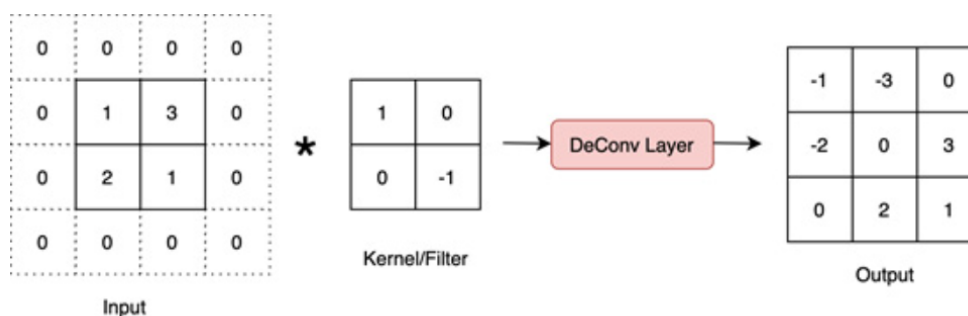


Figure 11: Deconvolutional layer (Jha & Pillai, 2021, chap.1)

When remembering past inputs while dealing with present ones is important, *Recurrent Layers* can be of great value (Jha & Pillai, 2021, chap.1). This is usually the case when the data is sequential (Jha & Pillai, 2021, chap.1). A *Recurrent Layer* imparts memorizing capabilities to the network (Jha & Pillai, 2021, chap.1).

A *Pooling Layer* is similar to the convolutional layer in that it downsizes the input data. E.g., the Max-Pooling operation takes the highest value from an array (usually 2x2) and that way, reduces the size of the input by a factor of 2 (Chollet, 2018, chap.5.1). There are other kinds of *Pooling Layer* such as Min-Pooling and the Mean-Pooling, but the most used *Pooling Layer* is the Max-Pooling (Chollet, 2018, chap.5.1). An example can be seen below:

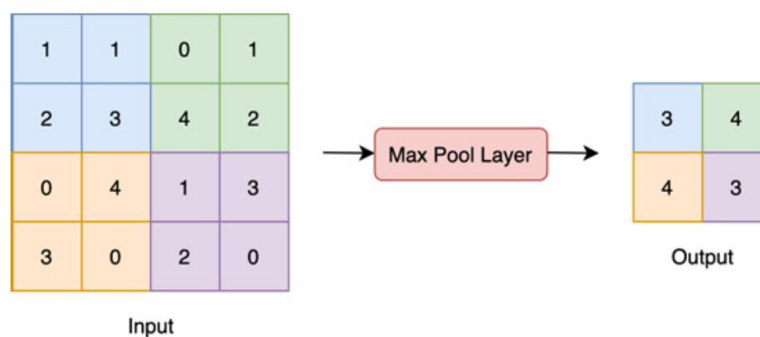


Figure 12: Max-Pooling layer (Jha & Pillai, 2021, chap.1)

*Dropout Layers* play an important role in the training process because they prevent co-adaptation (Ramsundar & Zadeh, 2018, chap.4). Co-adaptation appears when a neuron has learned a useful representation and other neurons deeper in the network rely on this particular neuron for information (Ramsundar & Zadeh, 2018, chap.4). This often results in poor predictions on new data because instead of learning a general rule, it relies on the feature learned by the neuron, which could also come from a quirk in the dataset (Ramsundar & Zadeh, 2018, chap.4). Looking at the learning process, co-adaptation is comparable to memorizing the dataset, which is an inefficient way of training the network and results in mediocre performance on new data (Jha & Pillai, 2021, chap.1).

Within the *Dropout Layer* some neurons are randomly dropped out, which means that the corresponding weight and therefore the contribution to the corresponding activation function is dropped to zero (Ramsundar & Zadeh, 2018, chap.4). The two graphs below show a fully connected layer (a) and a dropout layer (b) (Ramsundar & Zadeh, 2018, chap.4). It is important to note that the *Dropout Layer* is only applied in the



training process and should not be activated when making a prediction (Ramsundar & Zadeh, 2018, chap.4).

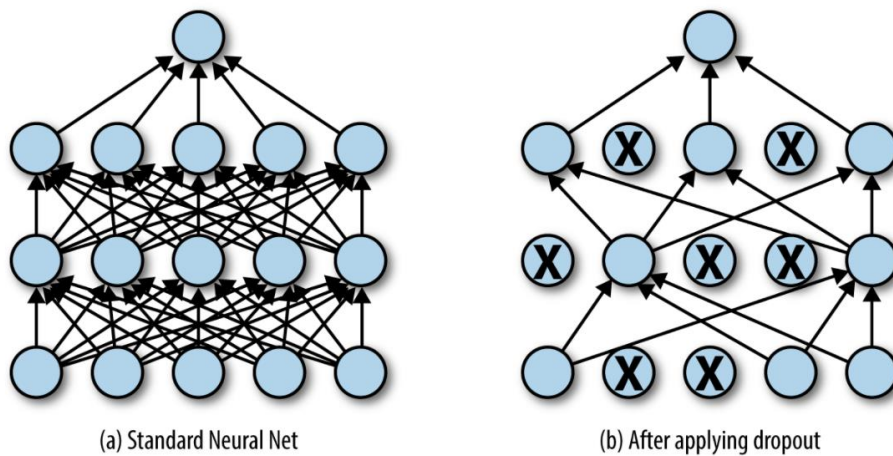


Figure 13: Dropout layer (Ramsundar & Zadeh, 2018, chap.4)

### Structure of NN

The general structure of a NN can be seen in the following diagram:

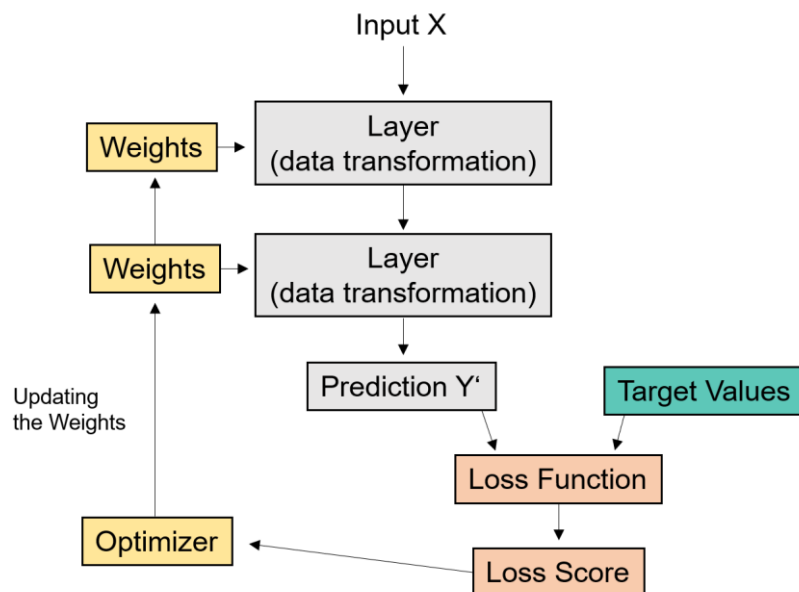


Figure 14: Structure of a NN (Own representation, in context of Chollet, 2018, chap.3.1)

The different layers can be combined as necessary (Chollet, 2018, chap.3.1). There are no set rules, and often this is done by trial and error, but the combination of the layers highly depends on the kind of NN that is to be built (Chollet, 2018, chap.3.1). With the input data, a prediction can be calculated and compared with the target value (Chollet, 2018, chap.3.1). These two values are fed into the loss function, which computes the loss score, which is an indicator of how good the prediction is (Chollet,

2018, chap.3.1). The optimizer then updates the weights of the layers and the NN moves on to the next dataset (Chollet, 2018, chap.3.1).

The choice of the loss function is a very important aspect of the training process and depends on the problem being solved by the NN (Chollet, 2018, chap.3.1). The following shows some commonly used loss functions:

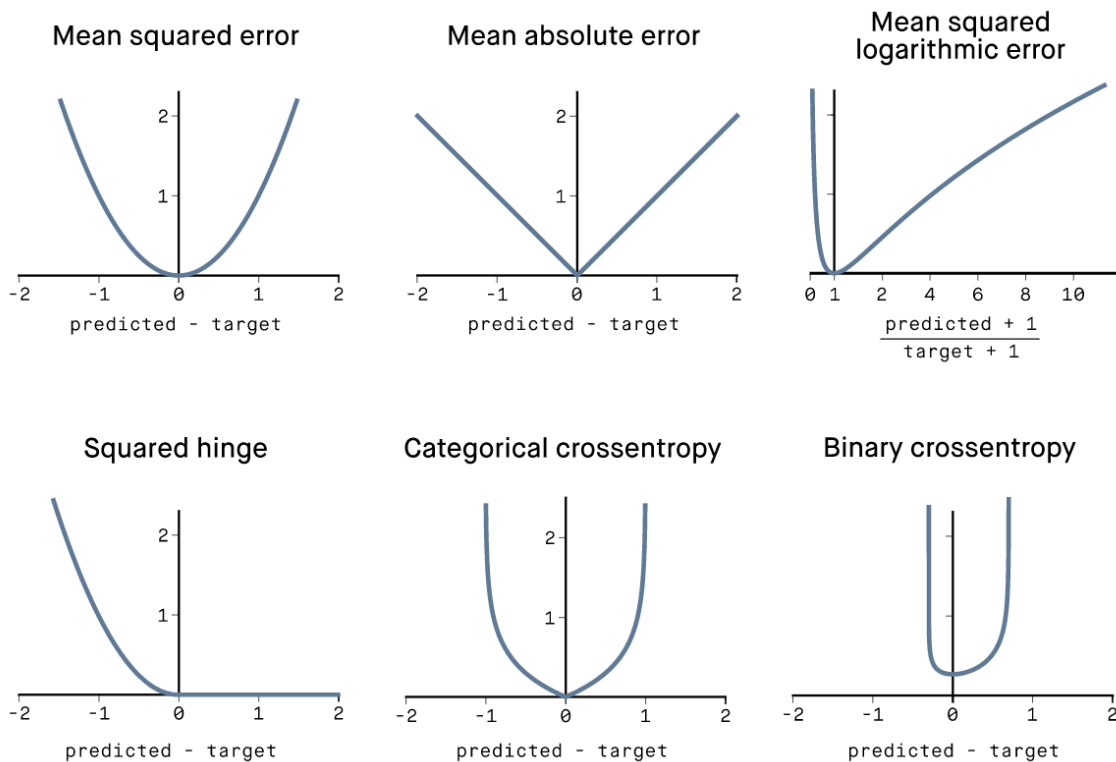


Figure 15: Loss functions (Peltarion, 2021)

Binary cross-entropy is a loss function that is used for classification with only two classes (Chollet, 2018, chap.3.1). For multivariate classification, categorical cross-entropy is used (Chollet, 2018, chap.3.1). The mean squared error is mainly used for regression tasks (Chollet, 2018, chap.3.1). There are more loss functions being developed for very specific deep learning problems (Chollet, 2018, chap.3.1). There also has been an attempt to make the loss function adaptive by including another parameter called 'robustness' to the loss function. It has been shown that this can increase the results for computer vision problems (Barron & Google Research, 2019).

## 2.4 Neural Networks for Point Clouds

After covering the theory of machine learning and NNs, the following explains the uniqueness of point cloud segmentation and some of the NNs recently developed for that task.

### Properties of Point Sets

Point Clouds have specific properties that need to be addressed before discussing the architecture of the NNs further.

The point cloud is a type of geometric data structure that inhabits certain properties. For one, point clouds are unordered, which means that the network consuming point clouds must be invariant to permutations of the input data (Qi, Su, Mo, & Guibas, 2017). Secondly, points within a point cloud cannot be processed individually because they are not single items but only part of certain objects (Qi, Su, Mo, & Guibas, 2017). These objects can only be detected by capturing local structures, which means that neighbouring points must be taken into consideration (Qi, Su, Mo, & Guibas, 2017). Finally, transformations like rotation and translation of points should not have any impact on the category or the segmentation of the point cloud (Qi, Su, Mo, & Guibas, 2017).

### PointNet

PointNet is a CNN that solves the problem of the increasingly large footprint in case one chooses to represent the geometric data as voxels (Vasilev, 2019, sec.4). This idea stems from the image recognition and classification problem, in which the image is represented by a 2D tensor with three slices, one for each channel (Vasilev, 2019, sec.4). Before PointNet, researchers tried to represent a point cloud the same way by using voxel representation (Vasilev, 2019, sec.4). The following picture shows an example of how a point cloud could be represented as voxels within a grid:

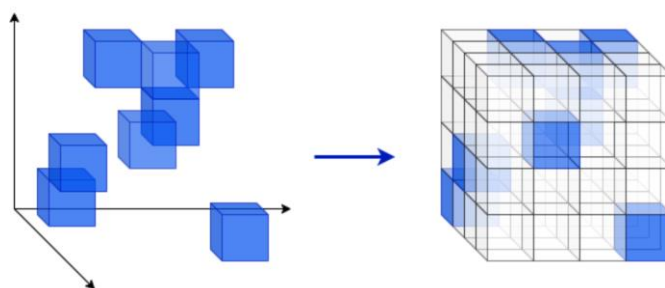


Figure 16: Point cloud, voxel representation (Vasilev, 2019, sec.4)

The problem with this representation comes into effect if the point cloud is sparse, which means that the footprint of the representation is much larger than the information that needs to be represented, which in the case shown above only consists of eight points (Vasilev, 2019, sec.4). To avoid this, PointNet represents the input data as vectors, which eliminates the representation of 'empty space' and only represents the points that were recorded (Vasilev, 2019, sec.4). The input data is represented as a  $n \times 3$  tensor with an arbitrary number of points (Vasilev, 2019, sec.4). This is possible because all the weights of the network are shared among all points (Vasilev, 2019, sec.4).

The following diagram shows the architecture of the PointNet network:

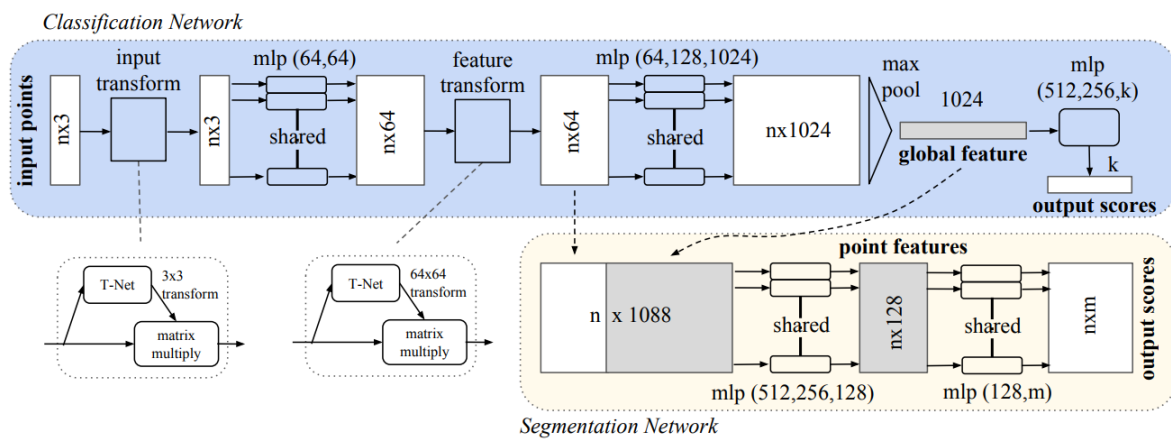


Figure 17: PointNet architecture (Qi, Su, Mo, & Guibas, 2017)

PointNet is a multi-layer perceptron (MLP), a feed-forward network that consists of fully connected layers and max-pooling layers (Vasilev, 2019, sec.4). The input passes through the input transform *T-Net*, where the input is unsampled to  $n \times 1,024$  with 64-, then 128-, then 1024-unit fully connected layers (Vasilev, 2019, sec.4). After that, a max-pooling operation is applied, which outputs a  $1 \times 1,024$  vector, which is then fed to another MLP with a 512- and a 256-unit fully connected layer (Vasilev, 2019, sec.4). By these, the data is downsampled to a  $1 \times 256$  vector and then multiplied by the  $256 \times 9$  matrix of the global shared learnable weights (Vasilev, 2019, sec.4). This results in a  $3 \times 3$  matrix which is multiplied with the original input point to produce the final output of a  $n \times 3$  tensor (Vasilev, 2019, sec.4). The feature transform uses a very similar *T-Net*, only that the input and output of this transformation is a  $n \times 64$  tensor, which results in a  $64 \times 64$  matrix (Vasilev, 2019, sec.4). the following diagram shows the T-Net-pipeline:

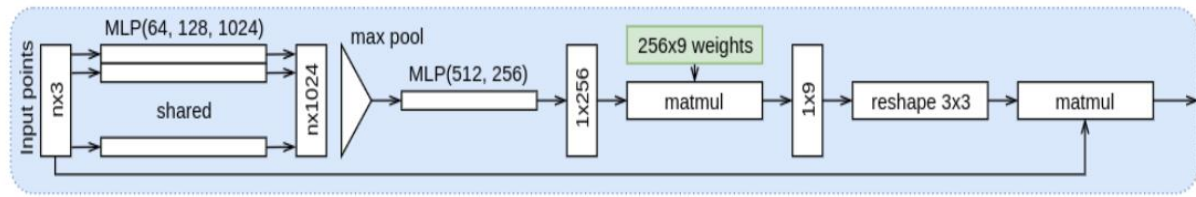


Figure 18: T-Net-pipeline (Vasilev, 2019, sec.4)

After the feature transform, the data is then gradually upsampled with 64, then 128, and finally 1,024 fully connected layers (Qi, Su, Mo, & Guibas, 2017). This output has the form of a  $n \times 1,064$  tensor and is then fed to a max-pooling layer, which results in a 1,024-dimensional output vector that represents the aggregated information among all  $n$  points (Qi, Su, Mo, & Guibas, 2017). The max-pooling is a symmetric operation and makes sure that permutation has no effect on the output, and compared to other symmetric functions, like average pooling and sum, the max-pooling showed the highest accuracy (Qi, Su, Mo, & Guibas, 2017).

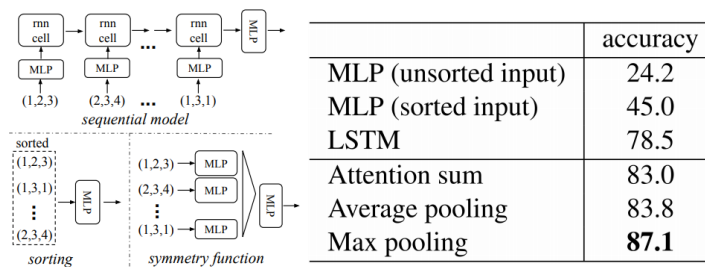


Figure 19: Accuracy of different pooling layers (Qi, Su, Mo, & Guibas, 2017)

PointNet then differentiates between two tasks: segmentation and classification (Qi, Su, Mo, & Guibas, 2017). The overall goal is to assign each point of the point cloud to a class (e.g., chair, lamp, table) (Qi, Su, Mo, & Guibas, 2017).

For the classification, the output vector of the max-pooling serves as an input of multiple fully connected layers resulting in a  $k$ -dimensional vector (Vasilev, 2019, sec.4). A  $k$ -way softmax function calculates the values for each class (Vasilev, 2019, sec.4). To test the accuracy of the classification of PointNet, Qi et al. (2017) used the ModelNet40, a dataset that represents 12,311 CAD models that are divided into 40 object categories (Qi, Su, Mo, & Guibas, 2017). The training dataset includes 9,843 models and the accuracy can be tested against a test set of 2,468 models (Qi, Su, Mo, & Guibas, 2017). PointNet performs well in classification tasks, reaching higher mIoU on points than two traditional models (Qi, Su, Mo, & Guibas, 2017).

The segmentation, on the other hand, requires local and global knowledge of the point cloud (Vasilev, 2019, sec.4). As can be seen in the diagram (Figure 17), this

knowledge comes from the intermediate  $n \times 64$  tensor and the global feature, which is then concatenated into a  $n \times 1,088$  tensor (Vasilev, 2019, sec.4). By a series of fully connected layers, the vector of each point is downsampled to a  $n \times 128$  tensor (Vasilev, 2019, sec.4). Finally, the last fully connected layer has  $m$  units, one for each class and a softmax activation (Vasilev, 2019, sec.4). In order to test the accuracy of the segmentation of PointNet, Qu et al. (2017) used the ShapeNet part data set, which includes 16,881 shapes of 16 categories (Qi, Su, Mo, & Guibas, 2017). The accuracy of the segmentation also proved to be above average (2.3% mean IoU improvement) (Qi, Su, Mo, & Guibas, 2017).

Another advantage that must be considered is the fact that PointNet performs well even if the density of the points (mentioned in chap. 2.1 as resolution) decreases (Qi, Su, Mo, & Guibas, 2017). Tested against two data sets, the accuracy only drops by 2.4% and 3.8% when the number of points is reduced by 50% (Qi, Su, Mo, & Guibas, 2017). The same robustness has been found considering outliers: The accuracy remains over 80% even if 20% of the points are outliers (Qi, Su, Mo, & Guibas, 2017). Qi et al. (2017) also proved that PointNet is very efficient considering the computational cost, which makes it scalable and therefore applicable for larger data sets (Qi, Su, Mo, & Guibas, 2017).

### PointNet++

The problem with PointNet is that local structures induced by the metric are not captured by the network (Qi, Yi, Su, & Guibas, 2017). To solve this problem, the authors of PointNet introduced PointNet++, which is a hierarchical neural network that extracts local features that are then grouped into higher-level features (Qi, Yi, Su, & Guibas, 2017). By repeating this process, the features of the whole point cloud are captured (Qi, Yi, Su, & Guibas, 2017).



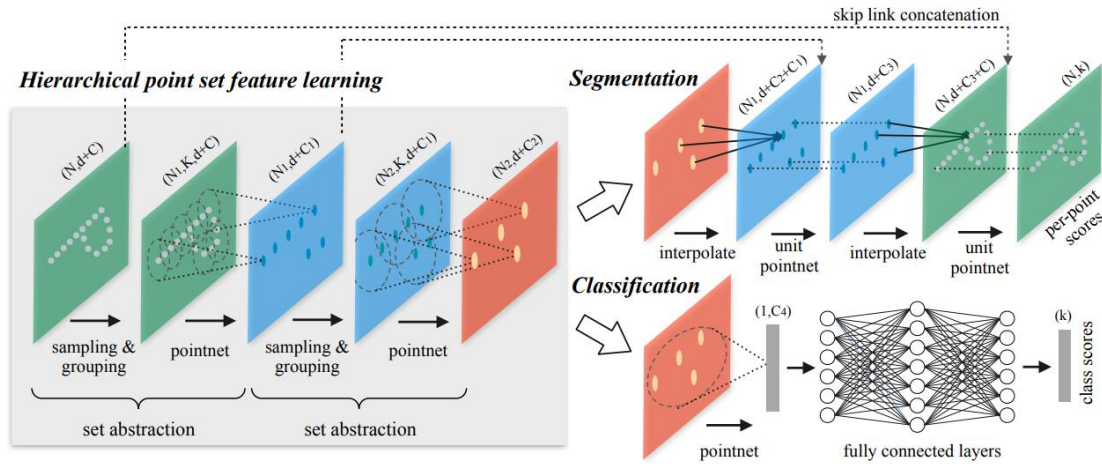


Figure 20: Hierarchical feature learning architecture of PointNet++ (Qi, Yi, Su, & Guibas, 2017)

The diagram above shows the hierarchical feature learning architecture of the PointNet++ network (Qi, Yi, Su, & Guibas, 2017). On the left, it can be seen that the process consists of a number of set abstractions. Each abstraction reduces the number of elements and is made of three key layers: *Sampling Layer*, *Grouping Layer*, *PointNet Layer* (Qi, Yi, Su, & Guibas, 2017).

The *Sampling Layer* selects a subset of points from the input data using iterative farthest point sampling (FPS) (Qi, Yi, Su, & Guibas, 2017). Given the same number of centroids, FPS has better coverage of the entire point set than random sampling (Qi, Yi, Su, & Guibas, 2017).

The *Grouping Layer* takes a point set of the size  $N \times (d + N)$  together with the coordinates of a set of centroids of size  $N \times d$  and the output, which are groups of point sets, which is of the size  $N' \times K \times (d + C)$  (Qi, Yi, Su, & Guibas, 2017).  $K$  represents the number of points in the neighbourhood of centroid points (Qi, Yi, Su, & Guibas, 2017). There are two methods that can find the points within a neighbourhood:

1. The ball query selects all the points that are within a certain radius of the query point (Qi, Yi, Su, & Guibas, 2017).
2. The  $K$  nearest neighbour (kNN) is a range query that selects a certain number of points (Qi, Yi, Su, & Guibas, 2017).

Since the ball query is more generalizable than the kNN, it is the preferred method of selecting the neighbouring points (Qi, Yi, Su, & Guibas, 2017).

The *PointNet Layer* is applied to learn local patterns (Qi, Yi, Su, & Guibas, 2017). These relationships between points can be learned using relative coordinates and

point features (Qi, Yi, Su, & Guibas, 2017). The layer takes in  $N'$  local regions of points with the data size  $N' \times K \times (d + C)$  and the output data has the size  $N' \times (d + C')$  (Qi et al., 2017).

The specialty of PointNet++ is the density adaptive PointNet layers that can learn to combine features from regions of different scales (Qi, Yi, Su, & Guibas, 2017). This is done by two types of adaptive layers:

1. *Multi-Scale Grouping* (MSG) applies grouping layers of different scales, followed by PointNets, to extract the features of each scale (Qi, Yi, Su, & Guibas, 2017). Afterwards, these features are combined into multiscale features (Qi, Yi, Su, & Guibas, 2017). By using random input dropout, a portion of the input data is not submitted to the learning process, and that way, the network is presented with input data of various sparsity and varying uniformity (Qi, Yi, Su, & Guibas, 2017).
2. *Multi-resolution Grouping* (MRG) is a computationally expensive and time-consuming operation because PointNet is applied at large-scale neighbourhoods for every centroid point (Qi, Yi, Su, & Guibas, 2017). The authors of PointNet++ suggest that the features of a region of some level are a concatenation of two vectors (Qi, Yi, Su, & Guibas, 2017). The first vector aggregates features of a subregion from the lower level; the other is obtained by processing all raw points in a local region by a single PointNet (Qi, Yi, Su, & Guibas, 2017). That way, the first layer provides more information if the density is high (Qi, Yi, Su, & Guibas, 2017). If the density is low and the point cloud is sparse, the second vector is weighted higher (Qi, Yi, Su, & Guibas, 2017).

The PointNet++ network shows even higher accuracies than the PointNet network (Qi, Yi, Su, & Guibas, 2017). Moreover, it is very robust even if the point cloud is sparse or incorporates sampling deficiencies (Qi, Yi, Su, & Guibas, 2017).

### KPConv

Another NN that operates directly on point clouds is the Kernel Point Convolution (KPConv), and it does so without any intermediate representation (Hugues, et al., 2019). The general idea is that a set of customizable 3D filters is defined by convolution and these filters are applied on a local level of the point cloud (Hugues, et al., 2019). Inspired by image-based convolution, KPConv uses a set of kernel points to define the area where each kernel weigh is applied (Hugues, et al., 2019). There is



no constrain on the number of kernels which makes the design more flexible (Hugues, et al., 2019). The illustration below shows the operation on 2D points. The input points (grey) are convolved by a set of kernel points (black) with the help of the filter weights on each point (Hugues, et al., 2019).

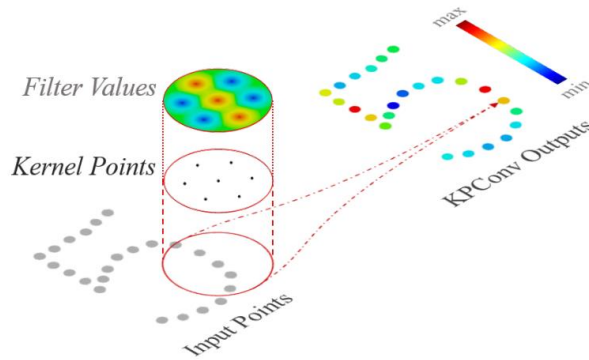


Figure 21: KPCConv operating on a 2D input (Hugues, et al., 2019)

When dealing with image convolution, the kernel is aligned with the image, which leads to a weight matrix  $W_k$  that is then multiplied with each pixel feature vector (Hugues, et al., 2019). Due to its unstructured form and variance in density and point number, the points in the point cloud are not aligned with the kernel used by KPCConv (Hugues, et al., 2019). Therefore, the relative position of each point and the kernel points allows for a correlation coefficient, which forms the weight matrices that are multiplied with the point features (Hugues, et al., 2019). The two different practices can be seen below (Hugues, et al., 2019).

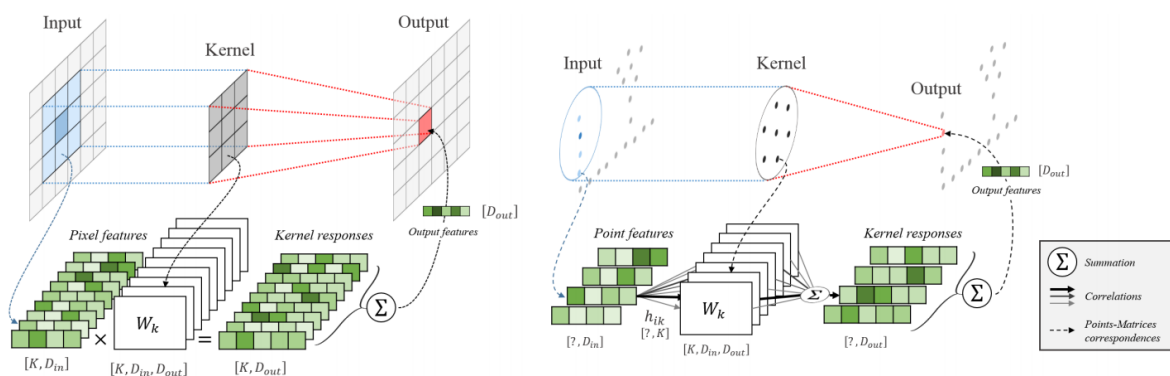


Figure 22: Comparison: image convolution (left) and KPCConv (right) (Hugues, et al., 2019)

An important part of the KPCConv design is the subsampling strategy which ensures a consistent density of the input points for each layer by using the grid subsampling (Hugues, et al., 2019). With this strategy, all the empty grid cells can be neglected, which not only alleviates the computational cost but establishes robustness against varying densities of the point cloud (Hugues, et al., 2019).

After the subsampling, the number of points is progressively reduced by using either a max-pooling layer or a convolution layer (Hugues, et al., 2019). By doubling the cell size for every layer, the receptive field of the KPConv increases (Hugues, et al., 2019). The inputs for the following KPConv layer are the points and their features and the matrix with the neighbourhood indices (Hugues, et al., 2019). The size of the neighbourhood matrix is defined by the largest number of neighbours in the point cloud (Hugues, et al., 2019). This creates so-called shadow neighbours, which are unused elements that are overlooked by the convolution computation (Hugues, et al., 2019).

The network architecture differs depending on the task at hand (Hugues, et al., 2019). The classification network (KP-CNN) consists of five layers, each with two convolutional blocks (Hugues, et al., 2019). Similar to an image convolution, the last layer aggregates the features by using global average pooling and fully connected and softmax layers (Hugues, et al., 2019). The segmentation network is called KP-FCNN and is a fully convolutional network (Hugues, et al., 2019). KP-FCNN uses the same encoder as KP-CNN (Hugues, et al., 2019). The decoder uses nearest upsampling to aggregate the final features (Hugues, et al., 2019). These features are passed between intermediate layers of the encoder and the decoder by using skip links (Hugues, et al., 2019).

Hugues et al. (2019) also propose a deformable version of the convolution operator in order to learn local shifts by deforming the convolution kernels (Hugues, et al., 2019). This makes the convolution kernels fit the point cloud geometry better and can lead to higher performance (Hugues, et al., 2019). An example of this can be seen below.

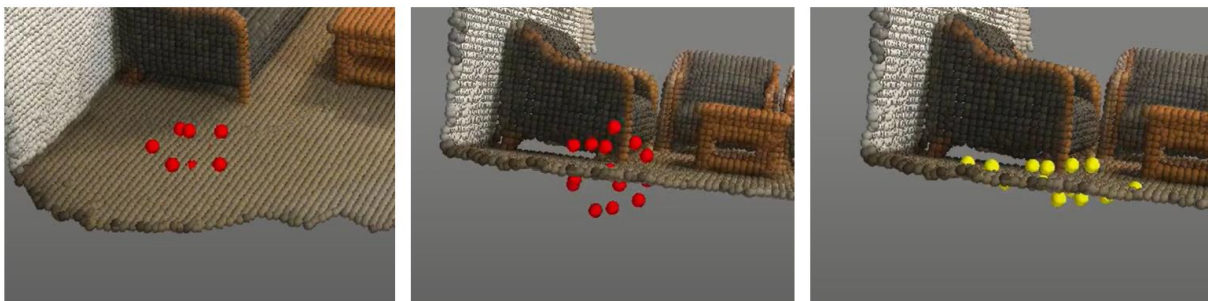


Figure 23: Deformable kernels (Hugues, KPConv Method)

The convolutional kernels can also vary in number (Hugues, et al., 2019). An example of how this might look can be seen below,  $K$  being the number of kernels

(Hugues, et al., 2019). Although it might be assumed that a higher number of kernels allows the spatial structures to be more complex, Hugues et al. (2019) have shown that the improvement of the mIoU eventually reaches a limit (Hugues, et al., 2019).

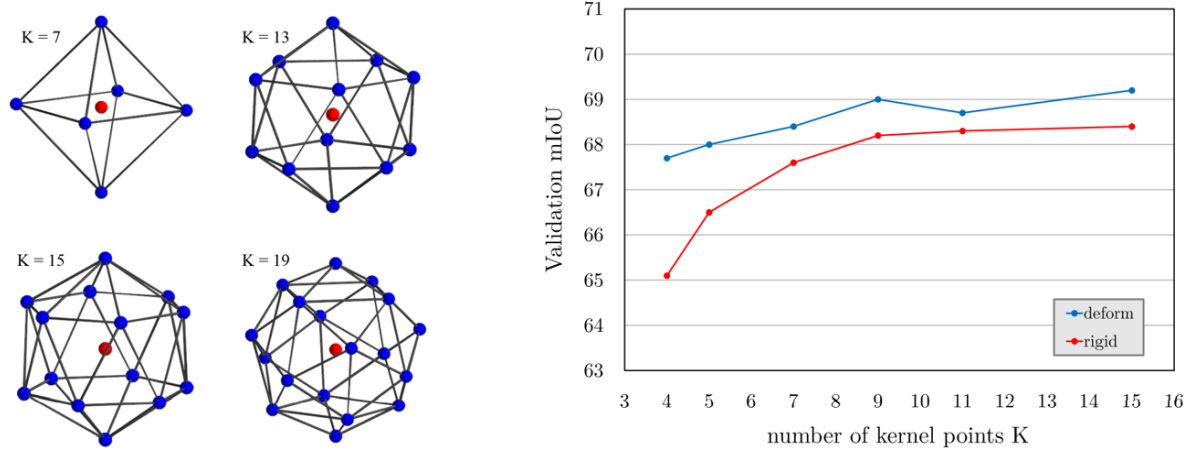


Figure 24: Different number of kernels in a stable position (left), mIoU and number of kernel points (right) (Hugues, et al., 2019)

### 3 Methodology

This chapter explains the methodology chosen for this project. It covers the methods behind the data preparation and the metric by which the neural network and the data preparation was evaluated. At the end of this chapter, the workflow of the project is illustrated.

Neural networks for point cloud segmentation are assessed by training and testing them with common datasets (Hugues, et al., 2019). There are many datasets available that can be used, not only to gain feedback on the overall performance but also to compare the accuracy of the model with different networks that operated on the same data set (Hugues, et al., 2019). Some of the experiments that have been done use different data sets to evaluate the segmentation and the classification part of their network (Hugues, et al., 2019).

For this project, the author used a point cloud that was collected by the Chair of Computational Modeling and Simulation at the Technical University of Munich.

#### Data Preparation

To train and test the neural network, the point cloud had to be dissected and labelled. This process of data preparation can be very time-consuming, but it is an important step to validate the effectiveness of neural networks. For this project, the holdout-method is used, which splits the labelled data set into two categories, the training set and the test set (Chollet, 2018, chap.4.1).

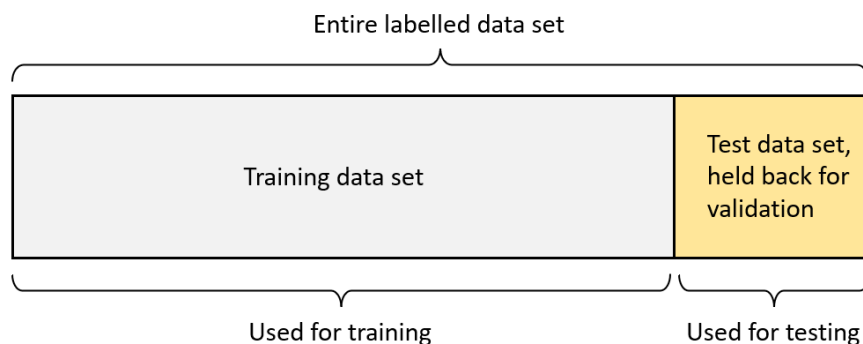


Figure 25: Holdout-Method, training data set and test data set (own representation, in context of Chollet, 2018, chap.4.1)

After the data preparation is completed, the training set is used to train the neural network. In this phase of the process, part of the training set becomes the validation data set which helps to find a suitable configuration within the given parameters (Chollet, 2018, chap.4.1). After validation, the validation data can be used for training purposes (Chollet, 2018, chap.4.1). When the training is done, the test set is then used to evaluate the neural network (Chollet, 2018, chap.4.1). It is important to note that the test set cannot take any part in the training process (Chollet, 2018, chap.4.1). Any adjustments to the model caused by the performance of the test data set can impair the assessment of the generalization ability of the neural network (Chollet, 2018, chap.4.1).

The disadvantage of the holdout-method comes to effect when the data set is small, and the training and validation data set does not contain enough samples for each class to be representative (Chollet, 2018, chap.4.1). If this is the case, a cross-validation would deliver different results (Chollet, 2018, chap.4.1).

### Evaluation Metric

To evaluate the results of segmentation and classification of the neural network, it is common to use the metric of the Intersection-Over-Union (IoU), which is superior to the regular accuracy (like pixel accuracy in image classification) for it evades the problem of class imbalance (Tiu, 2019). This problem occurs if the number of points is subject to fluctuation among the classes, which leads to an overemphasis on larger classes (Tiu, 2019). The accuracy of smaller classes is largely overlooked (Tiu, 2019). In comparison, the IoU weights each class by the number of points in the ground truth (Tiu, 2019). The IoU is the overlap of the predicted segmentation and the ground truth divided by the union between the predicted segmentation and the ground truth (Tiu, 2019).

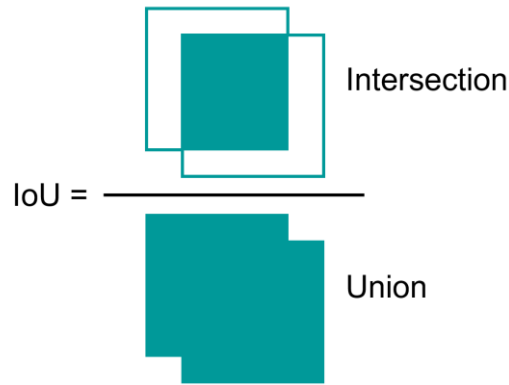


Figure 26: IoU, intersection over union (Tiu, 2019)

### Workflow

The chart below shows the workflow of this project. The segmentation and classification of the point cloud was comparatively time-consuming and made up a big part of the work. After this part was completed, the data set could then be tested. After the first results, a few changes were made. In order to draw sound conclusions, these changes had to be done one at a time. After comparison with the previous results, one could determine how to conduct the next experiment. The following diagram shows the overall workflow:

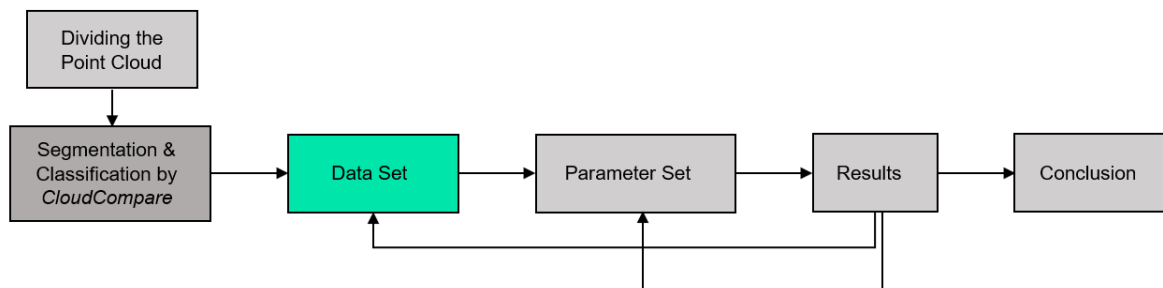


Figure 27: Workflow of the project (own representation)

Due to limited computational power, the experiments had to be carefully chosen. To accurately assess the work done by segmenting and classifying the point cloud, the work on the data set continued after the first experiments had been conducted. By changing either the parameter or the data set, it could be determined why the performance had changed.

## 4 Implementation and Experiments

In this chapter, the author outlines the implementation of the project. The first part explains how the data was prepared. Afterwards, the variations of each data set and the parameters are discussed, and the results are presented.

### 4.1 Segmentation: Data Preparation

As mentioned before, the point cloud needed to be segmented and labelled manually, which requires a software that can visualize the point cloud and allows for various operations. Additionally, it must be decided into how many classes of objects the point cloud is to be divided. This is only possible by looking at the point cloud in detail.

For the data preparation, the author used version 2.12 of the software CloudCompare, which is a 3D point cloud (and triangular mesh) processing software (CloudCompare, n.d.). The point cloud can be visualized and operated on by using various tools and standard plugins (CloudCompare, n.d.).

#### The Point Cloud

The point cloud used for this project derives from a scan of a part of the building of the Technical University at the main campus in Munich. The point cloud resembles part of the 3rd level of that building and includes offices, hallways, and a part of the staircase. Below one can see a satellite image of the Technical University (left) and the point cloud top-down (right).



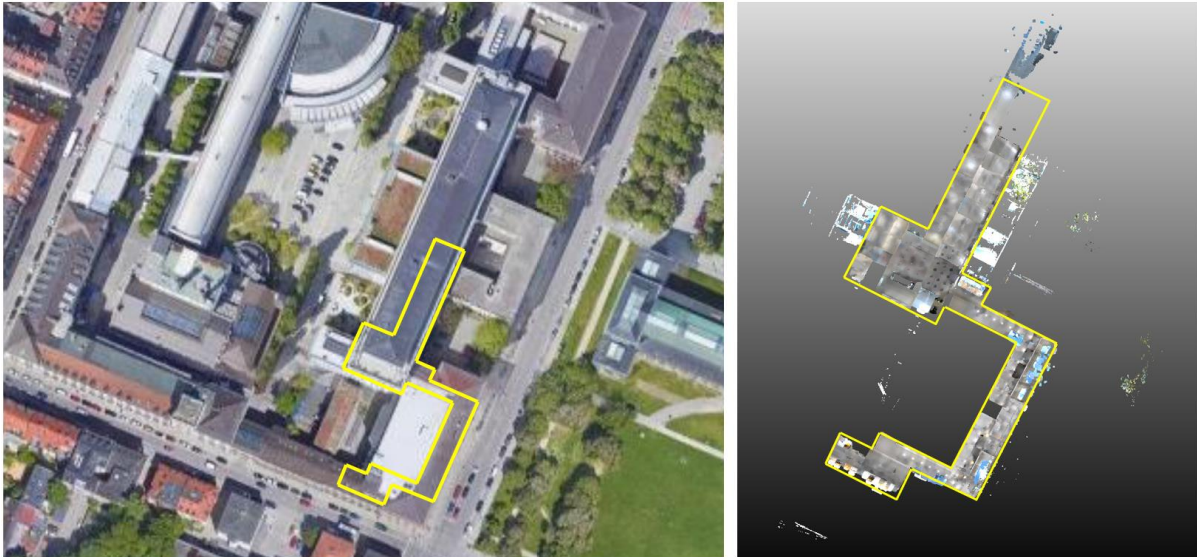


Figure 28: Point Cloud (right) as part of the building at the main campus of the Technical University (satellite image from google.de/maps)

The point cloud consists of 57,355,397 individual points, each with the XYZ-coordinates of the Euclidean space, the RGB colour values, the original cloud index, and the surface normals. An example of values of points can be seen below:

-8.73503113	-50.00270462	-0.71579826	17	17	18	0.000000	0.001209	-0.276796	0.960928
-8.67429447	-50.00173187	-0.70823085	22	19	17	0.000000	-0.104858	-0.178876	0.978268
-8.70870876	-49.99611282	-0.71060616	25	25	25	0.000000	-0.042513	-0.227139	0.972934
-8.66099167	-49.99893570	-0.70666522	41	31	27	0.000000	-0.120490	-0.172661	0.977584
-8.39732265	-49.81448746	-0.66850442	47	44	50	0.000000	-0.321039	-0.063688	0.944922
-8.46201447	-49.81408327	-0.68667030	13	14	10	0.000000	0.132045	0.018002	0.000041
XYZ-Coordinates	RGB-Colour Values	Original Cloud Index	Normals						
-8.40497398	-49.80456924	-0.67138886	42	43	45	0.000000	-0.315742	-0.055416	0.947226
-8.84619617	-49.57472610	-1.11421728	17	16	17	0.000000	0.024778	-0.206482	0.978136
-8.36463737	-49.59646606	-0.45558479	32	35	39	0.000000	-0.376247	-0.885904	0.271317

Figure 29: Raw data of the point cloud (own representation)

In order to segment the point cloud, a general division of the whole was made to handle the large amount of data more efficiently. For that, the point cloud was divided into smaller parts, generally office blocks and hallways. There were some points that could be regarded as 'noise' because they were outside the building and therefore not within the space of interest. With a few exceptions, each room was then separated from another, and hallways were divided into several parts. The individual parts can be seen below:



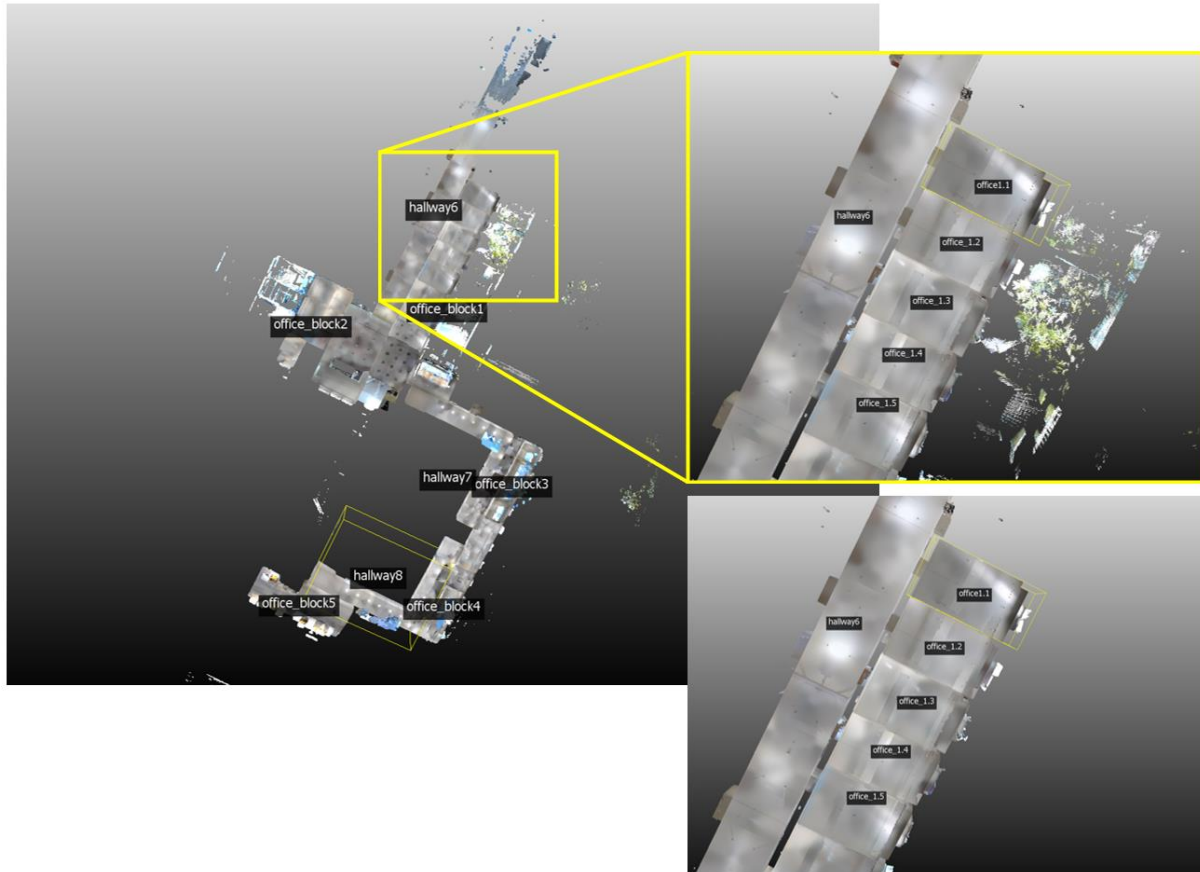


Figure 30: Dividing the point cloud (left), zoom in on the subdivision (top right), subdivision without the outside 'noise' (bottom right)

The office blocks and the hallways each had a different size and were therefore divided into different numbers of rooms and parts. The division by rooms was chosen to make it easier to cut the least number of objects so that when bringing the point cloud together, the objects that were cut in two could be brought together more easily. That way, all the objects could still be considered as whole and submitted to the network for training or testing purposes. The table below shows the divisions and subdivisions.

Table 2: Division and subdivisions of the point cloud

1 <sup>st</sup> Division	2 <sup>nd</sup> Division	Folders
Noise	Noise	Noise
Office Blocks	Office_Block1	Office1.1 Office1.2 Office1.3 Office1.4 Office1.5 Office1.6 Office1.7

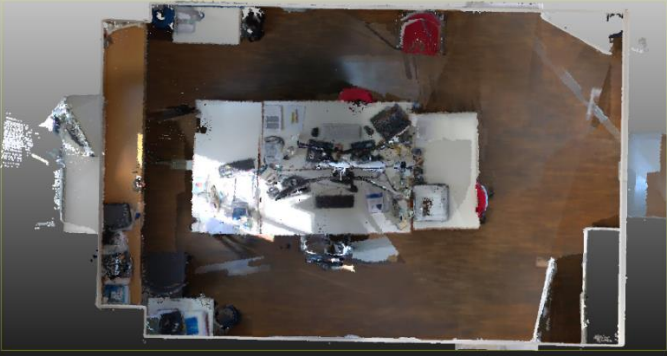
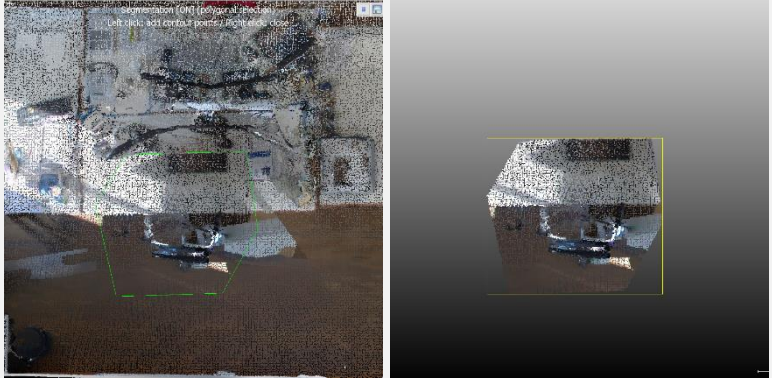
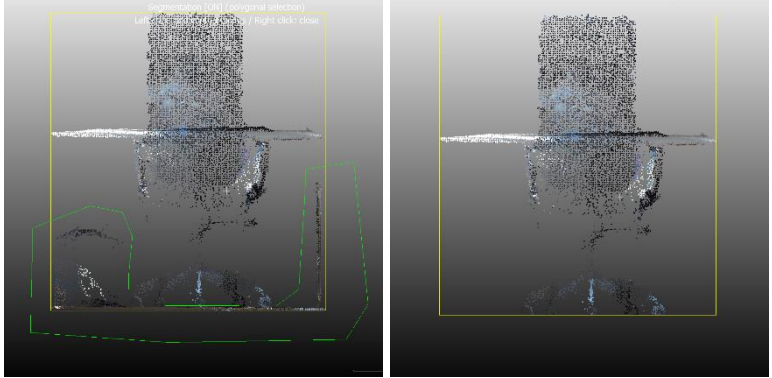
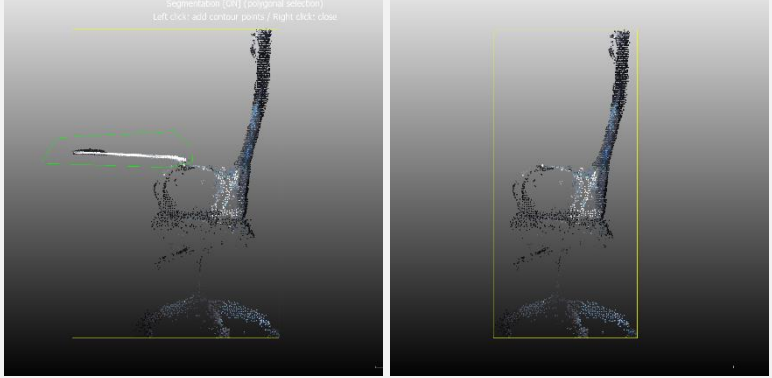
	Office_Block2	Office2.1 Office2.2 Office2.3 Office2.4
	Office_Block3	Office3.1 Office3.2 Office3.3 Office3.4
	Office_Block4	Office4.1 Office4.2 Office4.3
	Office_Block5	Office5.1 Office5.2 Office5.3
Hallways	Hallway1	Hallway1.1 Hallway1.2 Hallway1.3 Hallway1.4
	Hallway2	Hallway2.1 Hallway2.2 Hallway2.3
	Hallway3	Hallway3.1 Hallway3.2

After dividing the point cloud into smaller parts, the segmentation process was undertaken.

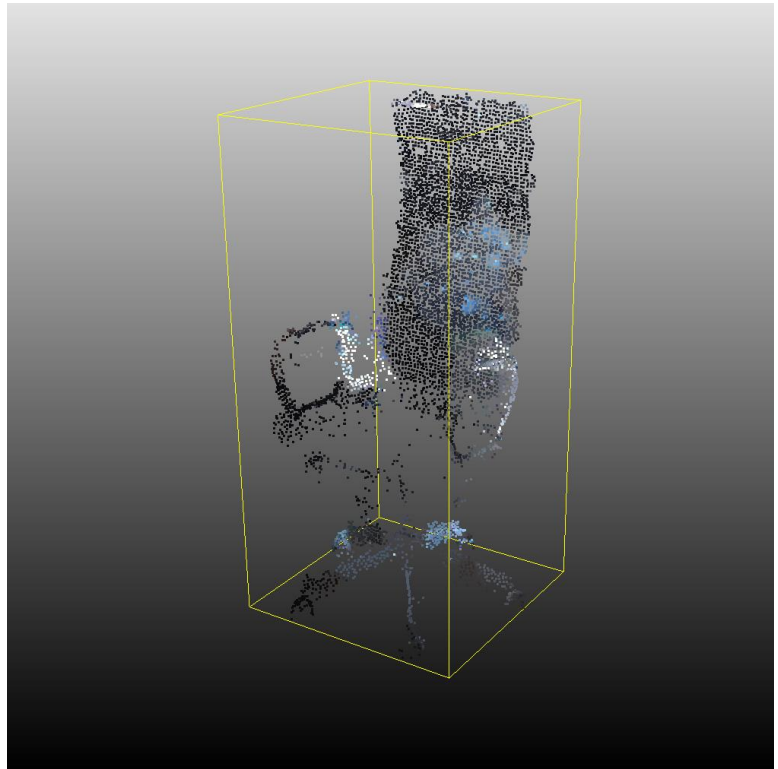
#### Point Cloud Segmentation with CloudCompare

To dissect the point cloud into its elementary parts and objects that could then be classified, the segmentation tool was used. The view of the point cloud could be adjusted so that the screen serves as a 2D projection on which the segmentation tool could be applied. The segmented part could then be singled out, or points that did not belong to a specific object could be cut off from the point cloud representing an object. An example of the process can be seen below.

Table 3: Segmentation process (own representation using the software CloudCompare)

<p>Top view of an office without ceiling and overhead lights</p>	
<p>Zoom in on an object which is to be segmented; here it is an office chair; first segmentation is done</p>	
<p>After segmentation from the top view, points that do not belong to this object are segmented out from another angle.</p>	
<p>Finally, a third angle is used to separate parts of the desk from the object.</p>	

The result is the office chair, represented by all the points belonging to this object.



This process is repeated until all the points are segmented and can be identified as belonging to a certain class. As mentioned before, the objects that were cut by division can be merged and assigned to one of the offices or hallways.

### Classification

By singling out certain objects, the points can then be assigned to a class. It is important to know what kind of classes there are, and the following gives an overview of all of the classes that have been chosen for this point cloud:

Table 4: Overview of classes

Building parts	Furniture	Items	Other
Ceiling	Bookshelf	Dustbin	Noise
Door <sup>1</sup>	Cabinet	Fan	Clutter
Frame <sup>2</sup>	Closet	Monitor	
Elevator	Desk Area <sup>3</sup>	Plant	
Exit Sign	Floor Unit	Fire Extinguisher	
Floor	Showcase	Printer	
Hallway Door	Standing Lamp	Overhead Projector	
Overhead Light	Table	Partition	
Smoke Detector	Bench	Sign	
Wall	Couch	Desk Lamp	
Window	Chair		
Heating	Office chair <sup>4</sup>		
Beam			
Wall Lamp			
Fixed Light			
Handrail			
Steps			
Skylight			
Hallway Window			
Air System			

Testing the hypothesis that the data preparation has an impact on the performance of the neural network, the data has been split into two separate data sets, Data Set I and Data Set II. Details will be explained and shown in section 4.2.

### Training Set and Test Set

It is imperative that, in order to draw any conclusions from the results, the data must be looked at very carefully. In doing so, one finds various irregularities within the point cloud. To work around this issue, the training and test data was carefully chosen to avoid the testing on parts of the point cloud that are unique and were therefore not subject to training. One example of this is the staircase, which appears only once

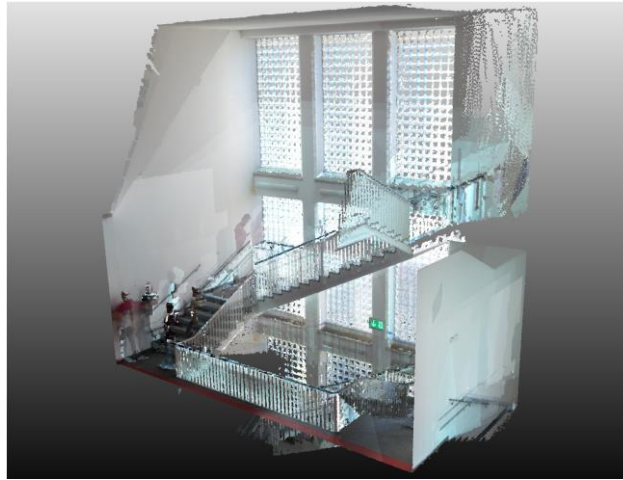
<sup>1</sup> Includes frame for Data Set I

<sup>2</sup> Used in Data Set II to specify the class door

<sup>3</sup> Used for Data Set I to differentiate between tables and desks

<sup>4</sup> Used in Data Set II to differentiate between normal chairs and office chairs

in the point cloud. This was intentionally chosen for training purposes only. It was ensured that the test sets were of regular size and not overly large or small. This would change the proportion of the test set and training set unnecessarily.



*Figure 31: View of the staircase, which is unique in the point cloud, therefore chosen for training purposes only (own representation with CloudCompare)*

The test set contains Office1.4, Office3.4, Office5.3, Hallway1.2, and Hallway2.2. This gives the test data enough diversity over all different kinds of offices and hallways without using unique parts of the point cloud for testing purposes.

Due to the limited number of parts within the point clouds, the test set chosen for the first five experiments does not contain objects of all the classes. The following classes are not included in the first test set: elevator, hallway door, plant, bench, fixed light, handrail, couch, overhead projector, partition, sign, desk lamp, hallway window, air system. If these classes are not predicted in the test set, the mIoU is set to the overall mIoU to not distort the overall performance.

## 4.2 Variations in Data Sets and Parameters

For the evaluation, this project used the neural network KPConv, which was described in section 2.4. An example of the various parameters chosen for the experiments can be seen in Appendix C.

In seeking to establish general rules on what makes the neural network perform better, multiple experiments have been conducted; each consisted of the training and testing of the full data set. The variations can be categorized into variation within the data set and variation of parameters of the point cloud. Following each experiment was a thorough examination of the results. The following diagram shows the detailed path the author took to evaluate the data sets and the various parameters:

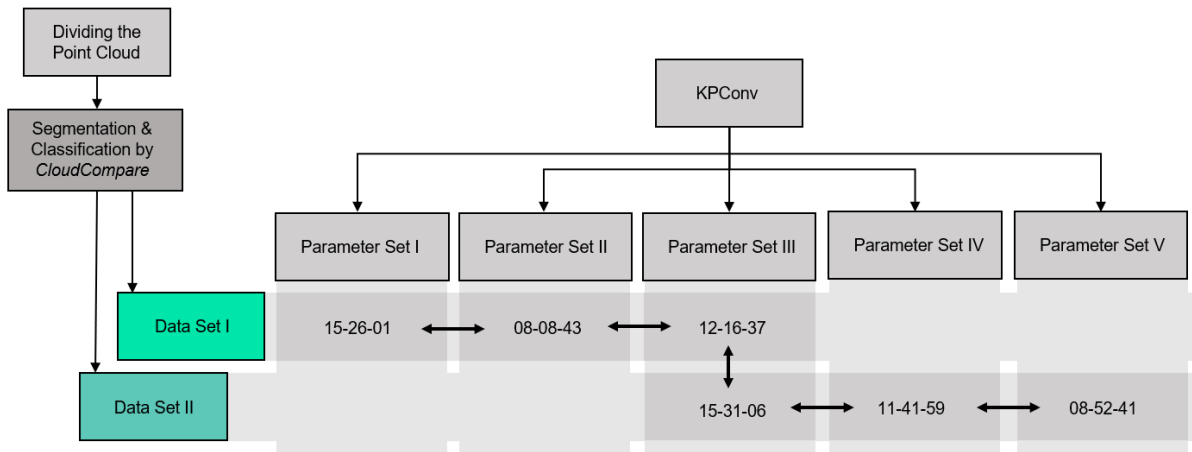


Figure 32: Path to evaluate the effects of different data sets and parameters, the numbers represent the number of the experiment

## Data Sets

Data Set I and Data Set II share many of the classes, as can be seen in Table 4, but there are some minor changes that were applied to Data Set II in order to test various hypotheses. These changes contain the following:

1. The class 'chair' was divided into a class 'chair' and 'office chair'. The purpose of this split is to analyse the performance after differentiating between two similar but also quite distinctive objects. With more attention to these details, the overall performance of the classes could be different from the performance of only one class. The following figure shows the distinction between the class 'chair' and the class 'office chair':

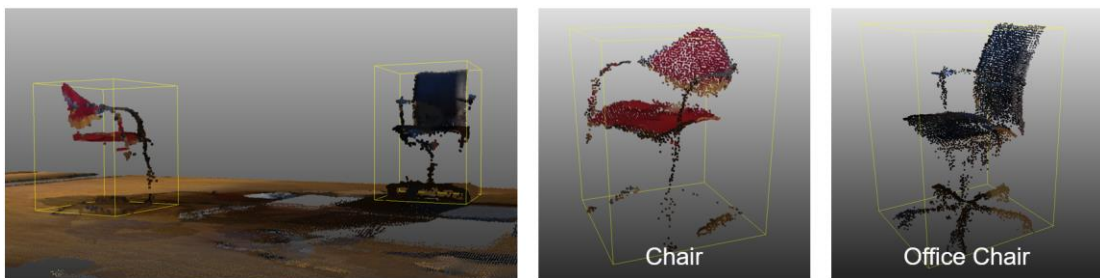


Figure 33: Split of the class 'chair' (Data Set I) into the classes 'chair' and 'office chair' (Data Set II)

2. Another change was splitting an object into two different objects; in this case, the author chose to split the object 'door' into the objects 'door' and 'frame' and classified them accordingly. This can be distinguished from the above because, unlike the chairs, the objects of this class were not only classified differently but each object was again segmented. The aim of this change is to reduce the complexity of the object itself and analyse the results to see if this



path should be pursued further. The segmentation can be seen in the following:



Figure 34: Segmentation of the object 'door' (Data Set I) into the objects 'door' and 'frame' (Data Set II)

3. The third change was similar to the first one, only reversed. In order to see if two very similar classes could be more easily recognized by the neural network if the classes are to be combined, the class 'table' and the class 'desk area' were merged into the class 'table'. This was done because the distinction between the two was not always clear, and combining them might reduce the wrongful classification of one as the other. This might lead to an increase in performance within the class and helps to understand the results behind change number one.
4. Other changes result from a previously inconsistent classification within the Data Set I. To differentiate the class 'monitor' even more, two objects were resorted into the class 'clutter' because they were of a different kind (bigger size, mounted to the wall). Also, there were some objects of the class 'skylight' which were wrongly classified as 'window', which might have led to an inconsistency in the training and testing process. By making these amends, using the Data Set II should lead to better performance for these particular classes.

## Parameters

The first change that distinguishes the first experiment from the second consists of a difference in the training strategy. While the first experiments weighted the classes considering the number of points, the second experiment considered them equal, disregarding the number of points in each class.

The second change in parameter was the reduction of the subsampling size ( $d_l$ ), which is one of the key parameters of the KPConv. The subsampling size was reduced from 0.05 m to first 0.03 m and then to 0.02 m. The following picture shows



the points chosen for the subsampling in one part of the point cloud. These are the subsampling points for the experiment 12-16-37, and the distance between the points is 0.05 m.

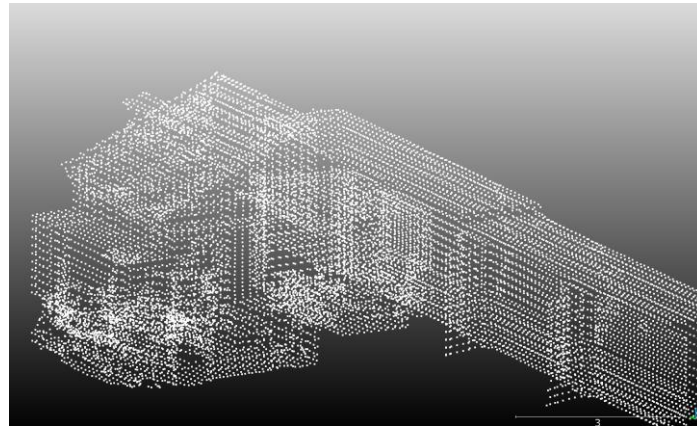


Figure 35: Subsampling size (own representation, created with CloudCompare)

After having explained the variations on the data set and the parameters, the next section will show the results of each of the experiments.

### 4.3 Results

#### Value Figure

The results of each experiment are the IoUs computed for each class after every epoch. The number of epochs was set to 500 for each of the experiments. The number of classes was 42 for Data Set I and 43 for Data Set II. The performance over all classes can be expressed by calculating the mean over all classes. The classes that do not appear in the Test Set are automatically set to the mean value so that these do not have any negative impact on the result. In the following Figure one can see the performance of the first experiment over all 500 epochs:

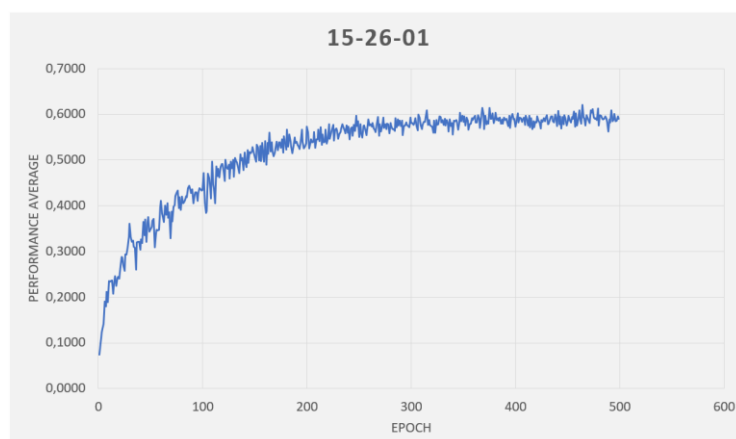


Figure 36: mIoU of experiment 15-26-01

The performance increases the most during the first 100 epochs and moves asymptotically towards a value close to 60%. It must be noted that even though the change in performance is not very great with an increasing number of epochs, the variance of each performance is still substantial. In order to evaluate the variations in data sets and parameters, the author chose the mean value over the last 100 epochs, which is called *mIoU\_100*. That way, the performance chosen to evaluate the variations is not coincidentally smaller or larger than the one just before. The *max\_IoU* is considered the maximum performance over all epochs and classes.

The following Table shows the experiments that were done in chronological order and the data sets and parameters that were used. It also shows the *mIoU\_100*, the *max\_IoU*, and computing time for the experiment.

*Table 5: Results of all experiments*

<b>Experiment</b>	<b>Data Set</b>	<b>dl (m)</b>	<b>mIoU_100</b>	<b>max_IoU</b>	<b>Comp. Time (h)</b>
15-26-01	Data Set I	0.05	59%	62%	12.4
08-08-43	Data Set I	0.05	62%	66%	10.5
12-16-37	Data Set I	0.03	64%	66%	21.4
15-31-06	Data Set II	0.03	69%	72%	21.3
11-41-59	Data Set II	0.02	72%	74%	45.1
08-52-41	Data Set II	0.05	65%	68%	10.6

Even though the performance of each experiment seems to improve progressively, one must distinguish between the performance of the various classes to draw any conclusion on what might have caused the improvement. The performance of classes was also measured by taking the *mIoU\_100* for each class.

### Results for different Data Sets

Below one can see the results for the classes that have been undergone various changes while making Data Set I and Data Set II. The other parameter stayed the same over the two experiments.

Table 6: Results for different data sets (in percent)

Class	12-16-37	15-31-06
Chair	89	78
Office chair	-	91
Door	77	66
Frame	-	67
Desk area	74	-
Table	26	80
Monitor	90	92
Window	29	90
Skylight	13	84

The different changes discussed in the previous section has led to the following results:

1. The separation of the class 'chair' into the class 'chair' and 'office chair' had only minor impact on the results. It also appears not to be very conclusive. Having a mIoU\_100 of 89% using Data Set I, the performance increased slightly for the class 'office chair' (91%). For the other chairs though, the performance decreased to 78%.
2. Segmenting the doors and the frames seems to have a negative impact on the result. By doing so, the performance decreased from a mIoU\_100 of 77% to 66% for the class 'door' and 67% for the class 'frame'.
3. Similar findings can be done comparing the performance of the class 'table' which was combining the class 'desk area' and 'table' from Data Set I. Merging the two classes into one increased performance for both classes from 74% for desk areas and only 26% for tables to 80% for the combined class.
4. The most substantial difference in performance can be seen for the classes 'window' and 'skylight'. The performance went from under 30% to over 80%. Also, the class 'monitor' appears to be recognized even better if the objects are more uniform. The performance increased slightly to 92%.

## Results for different Parameters

The first two experiments compare two different training strategies. In the first experiment, all classes are trained on regarding their size, which is the number of points. In the second experiment, all classes are considered equally, no matter the number of points within the class.

Treating all classes equally seems to have a positive impact on the performance, especially for classes of smaller objects like exit signs and smoke detectors. The performance went from almost 1% to 62% for exit signs and from 1% to 23% for smoke detectors. Moreover, objects of classes like 'fan', 'monitor', and 'wall lamp' were recognized much better due to the change. All of these can be described as smaller objects. A direct comparison between the results can be seen in Table 7.

By considering all classes equally, the learning curve appears to be steeper. This can be seen in the following figure:

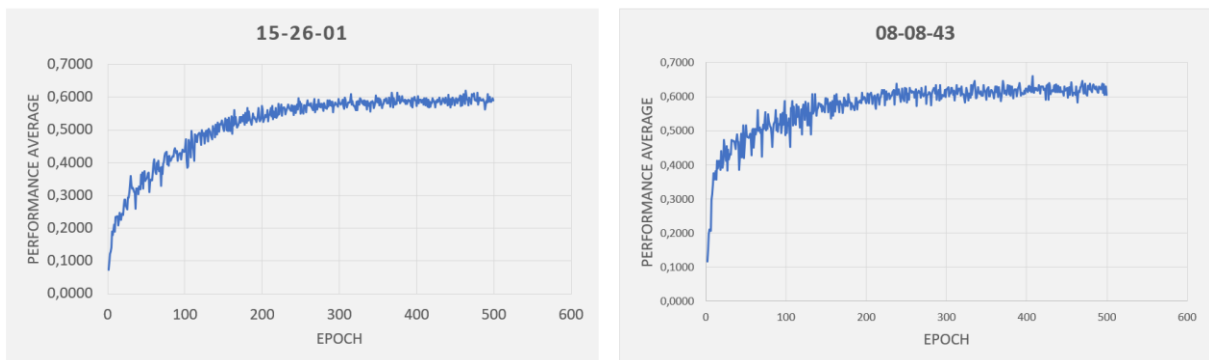


Figure 37: Comparison of the mIoU\_100 for experiment 15-26-01 (left) and 08-08-43 (right)

*Table 7: Comparison of mIoU\_100 of some classes of the experiments 15-26-01 and 08-08-43 (in percent)*

<b>Class</b>	<b>15-26-01</b>	<b>08-08-43</b>
Chair	88	91
Door	75	64
Exit sign	1	62
Fan	55	64
Monitor	85	88
Smoke detector	1	23
Standing Lamp	69	75
Heating	55	57
Wall lamp	74	83
Fire extinguisher	77	80

Experiment 12-16-37 is similar to experiment 08-08-43; the only difference is that the subsampling size was set to 0.03 m instead of 0.05 m. Even though the overall performance increased slightly from 62% to 64%, the computation time more than doubled. The same is true when comparing the experiments 15-31-06 and 11-41-59: The performance increases by 3%, from 69% to 72%, but the computational time doubled yet again, from 21.3 hours to 45.1 hours (see Table 8).

A direct comparison of some classes of four experiments can be seen in the following table:

Table 8: Comparison of different subsampling sizes (results in percent)

	Data Set I		Data Set II	
Class	08-08-43	12-16-37	15-31-06	11-41-59
Subsampling size (m)	0.05	0.03	0.03	0.02
Comp. Time (h)	10.5	21.4	21.3	45.1
Cabinet	48	49	48	58
Dustbin	62	55	67	55
Exit sign	62	76	48	83
Fan	64	62	57	46
Floor	98	97	98	98
Floor unit	16	18	23	47
Monitor	88	90	92	93
Smoke detector	23	26	29	37
Standing Lamp	75	78	74	76
Heating	57	67	69	71
Wall lamp	83	85	86	88
Fire extinguisher	80	87	90	90
Printer	10	10	22	12

It appears that for some classes, e.g., 'exit sign', the reduction of the subsampling size increased performance on that class dramatically, while other classes, e.g., 'floor', an improvement cannot be observed. For some classes, the performance even decreased by lowering the subsampling size. This is the case for the classes 'dustbin' and 'printer'.

## 5 Discussion

This chapter outlines the author's assessment of the results. Comparing the results and looking at the point cloud more closely, the author tries to explain the results.

### 5.1 Test Set Limitation

One of the phenomena that needed to be explained was that with a specification of the class 'chair', the performance of the new class 'chair' decreased by more than 10%. This is particularly surprising because the performance of the new class 'office chair' increased by 2%. After a careful analysis, this seemed to be caused by a low number of objects in the test data.

In the test set of Data Set I, there were 13 chairs that needed to be segmented and classified. These included office chairs and regular chairs. The network reached a mIoU<sub>100</sub> of nearly 90%. After splitting the class into the two classes mentioned above, there were only three regular chairs in the test set of Data Set II, but ten office chairs. The class with a higher number of objects within the test set appears to have a higher performance.

*Table 9: Results of the class 'chair', before and after the split (in percent)*

	12-16-37 (Data Set I)		15-31-06 (Data Set II)	
Class	Number of objects	mIoU (%)	Number of objects	mIoU (%)
Chair	13	89	3	78
Office chair	-	-	10	91

This is not a coincidence. It can be observed that several classes that have a low number of objects within the test data also have a low performance. Simultaneously, classes that have a higher number of objects within the test data are more often classified correctly and inherit a higher mIoU<sub>100</sub>. The following graphs show some classes to emphasise the correlation between the number of objects and the overall performance of that class. Larger objects are not included in these graphs; instead, all the classes depict objects of smaller size: chair (office chair), dustbin, fan, monitor, overhead lights, showcase, smoke detector, standing lamp, table, heating, beam, and wall lamp.

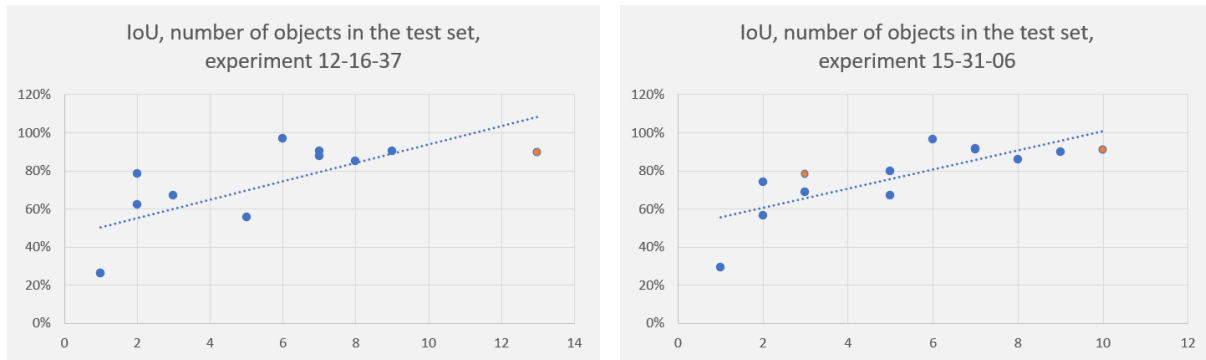


Figure 38: IoU and number of objects of classes

On the left, one can see the results for the above-mentioned classes for the experiment 12-16-37, which was done with the Data Set I, and on the right, one can see the results for the same classes for experiment 15-31-06, which was done using the Data Set II. The orange data points represent the class ‘chair’ (on the left) and the classes ‘chair’ and ‘office chair’ (on the right).

The reason behind this correlation is that wrongfully classified points carry greater weight if the overall number of objects is small. This can be shown not only by the high performance of classes with a low number of objects in the test set (e.g., fire extinguisher) but also by looking at the predictions more closely. For example, in the figure below, one can see that the lower part of a chair is classified as ‘table’ or ‘floor’. The difference is that before the split, this chair represents only one out of 13 other chairs, and after the split, it accounts for one out of three. Therefore, the performance of chairs decreases from 89% to 78%. On the other hand, if an object occurs only once but is segmented and classified correctly, the performance can still be fairly high: The class ‘fire extinguisher’ appears in the test set only once, but since it is well segmented and classified, the performance reaches 90%.

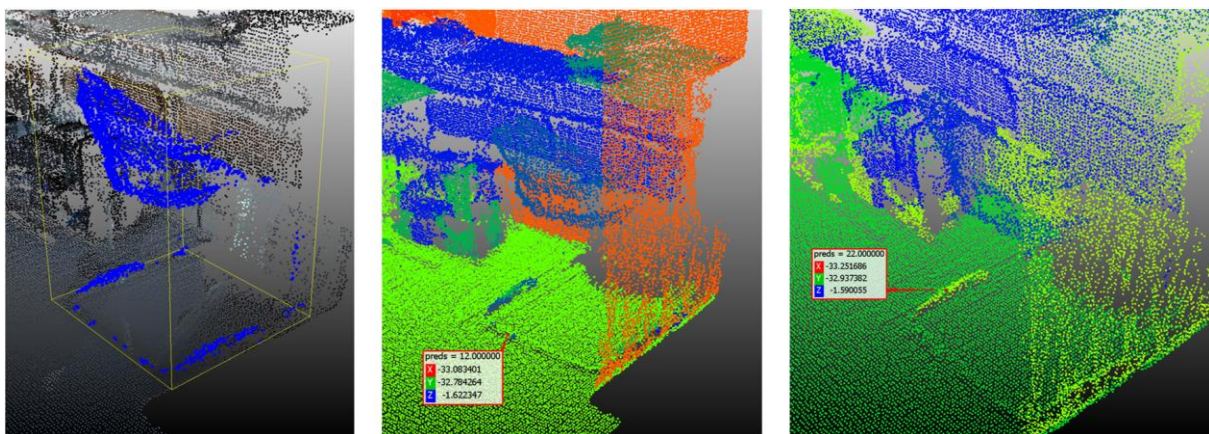


Figure 39: Prediction of a chair, wrongfully classified as floor (preds=12) and table (preds=22), experiment 12-16-37 (Data Set I) in the middle, experiment 15-31-06 (Data Set II) on the right (own creation with CloudCompare)



This assumption is also validated by the experiment 08-52-41, for which the test set was expanded, and the number of normal chairs increased from three to nine. Even with an increased subsampling size, the performance of the class 'chair' improved to 90%.

The number of objects of one class within the test set should be above six, preferably even higher, for the class to be properly represented. If this is not the case, a small number of points that were wrongfully classified could carry greater weight and therefore misrepresent the performance on that class.

The point cloud was divided into 31 subdivisions, and for the first five experiments, five of them were assigned as the test set. For the last experiment, six of the subdivisions were assigned as test set, which accounts for about 20% of the data set. With every assignment of test set and training set, one should be aware of the count of objects for each class to ensure sufficient representation.

## 5.2 Specialization vs. Generalization

Unlike the author's prediction, a specialization of classes did not necessarily lead to increased performance. As mentioned before, splitting the class 'chair' into the classes 'office chair' and 'chair' increased the performance of the one (office chair) but decreased the performance of the other (chair). In this case, the results indicate that the count of objects in the test set is too small. In other instances, it appears to be not that easily explained.

The general idea is that the more uniform the objects of a class are, the better the performance on that class. An indication for that would be the class 'monitor' in which a slight improvement could be observed after taking some objects out of this class that did not fit a narrower description. In the process of creating Data Set II, some monitors were taken out of the class 'monitor' that did not exhibit the standard size and were mounted to the wall instead of resting on a desk or cabinet. By making these changes, the performance of the class 'monitor' could be improved from 90% to 92%.

In the case of furniture, however, specialization proved to be more complicated. In Data Set I, the class 'table' and the class 'desk area' were used to differentiate between desks and tables. In Data Set II, these two classes were combined to the class 'table', and the performance improved.

Table 10: mIoU\_100 (in percent), comparison of class in Data Set I and Data Set II

Class	12-16-37	15-31-06
Door	77	66
Frame	-	67
Desk area	74	-
Table	26	80
Monitor	90	92

Pointing to the same conclusion, the split of the class 'door' into 'frame' and 'door' led to a decrease in performance. This cannot be explained by a lower count of objects within the test set because each former door was segmented into one frame and one door. That way, the number of objects in the test set is the same in Data Set I and Data Set II.

The performance of classes of storage areas like 'bookshelf', 'closet', 'cabinet' and 'floor unit' seems to be consistently mediocre, ranging between 16% and 61%, as can be seen in the following table:

Table 11: mIoU\_100 (in percent) of storage areas

Class	08-08-43	12-16-37	15-31-06
Bookshelf	61	56	53
Closet	29	37	35
Cabinet	48	49	48
Floor unit	16	18	23

While closets were frequently classified as wall<sup>5</sup>, some of the floor units were classified as cabinets. The following figure shows an example of this:

---

<sup>5</sup> See Appendix B

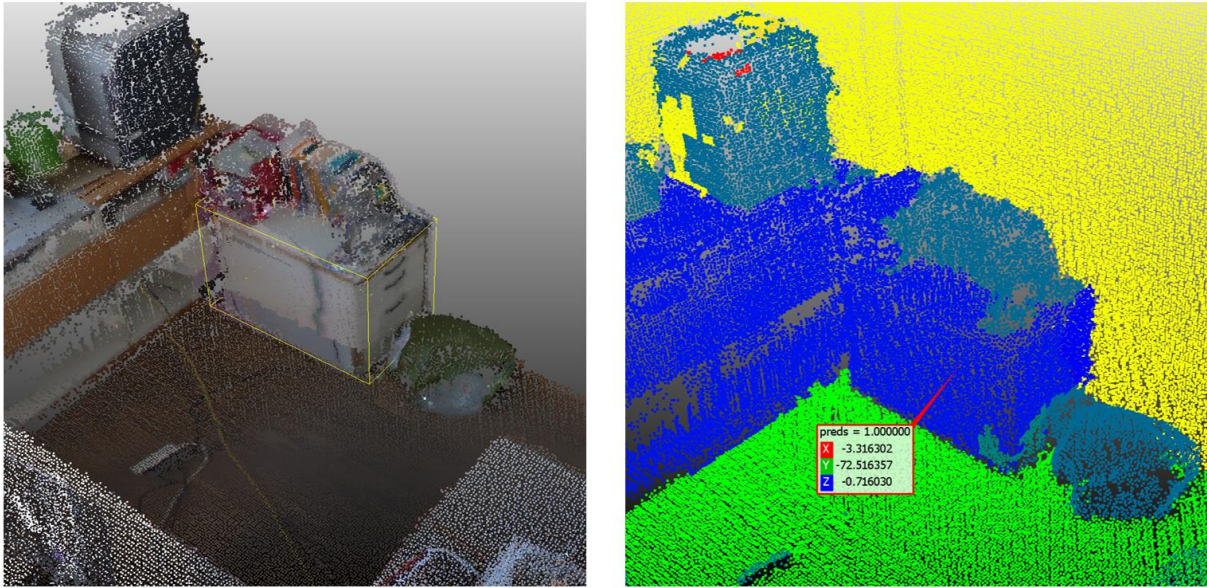


Figure 40: Floor unit wrongfully classified as cabinet (preds=1)

These observations raise the question if it is necessary to differentiate between a cabinet and a floor unit. Might it just be enough to know that these points belong to a small piece of furniture that can be used as storage? The same question can be asked for the differentiation of frame and door.

In the author's view, the answer to this question highly depends on the application. The degree of specialization of classes must be chosen with the end goal in mind, which is to capture all relevant information of the building. If the differentiation of cabinets and floor units is important, the rules to differentiate must be clearly defined. If this is not the case, it might be more sensible to combine the classes. In some cases, this differentiation might not be relevant.

One could also argue that all relevant information about size and dimensions can be done in the post-processing of the segmentation and classification process or even within the BIM itself. In that case, it might be more cost-efficient to generalize some of the classes and differentiate between objects afterwards. This does not apply to the wrongful classification of closets as walls, which would present a challenge if it were to be segmented again within the BIM.

### 5.3 Subsampling Size and 'Clutter'

By decreasing the subsampling size from 0.05 m to 0.03 m and then again to 0.02 m, it can be shown that each reduction improves the performance of the neural network. It can be argued, though, that the computational time increases so dramatically that

the further reduction is questionable. The following table shows the increase in performance and the increase in computation time.

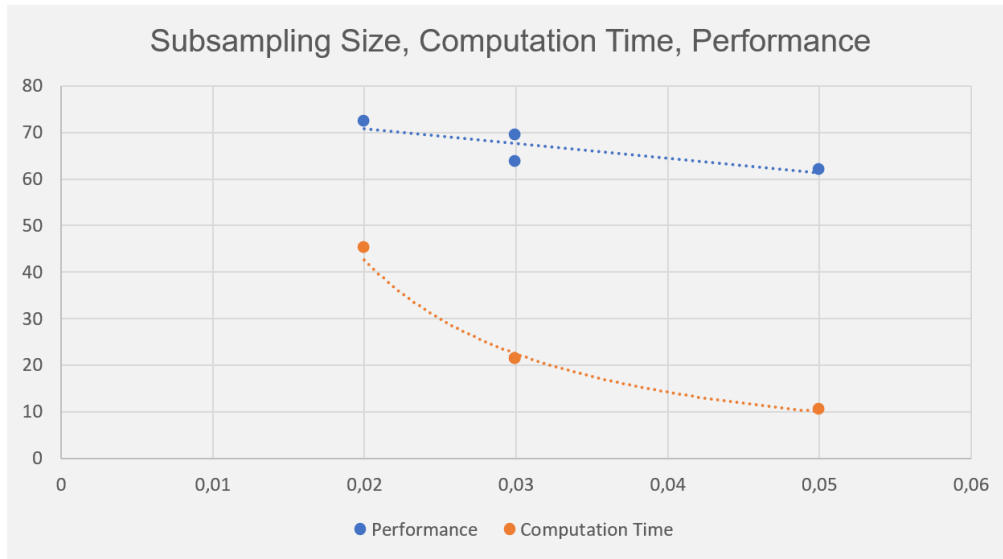


Figure 41: Subsampling size, computational time (orange, in hours), performance (blue, in percent)

While the increase of the performance is merely linear, the time it takes to compute the weights of the neural network increases exponentially. This becomes unsustainable for very small subsampling sizes.

Another interesting fact is that for some classes, the performance decreases with a reduction of the subsampling size. This can be seen in Table 8 for the classes ‘dustbin’ and ‘printer’. This cannot be simply explained by the size of the objects, because the increased performance of other small classes, namely ‘exit sign’, ‘smoke detector’ and ‘wall lamp’ show, that the performance of classes of smaller objects can also increase by lowering the subsampling size.

One explanation might be that both, dustbin and printer, are more likely to be classified as clutter. The following picture shows the prediction of the neural network on some points that are supposed to constitute a printer (left). Instead, a large portion of the points that belong to the class ‘printer’ are wrongfully classified as ‘clutter’ (preds=5.0). This portion increases when the subsampling size is smaller. The picture at the top shows the prediction after 500 epochs of the experiment 15-31-06 (dl=0.03 m), the picture at the bottom depicts the prediction after 500 epochs of the experiment 11-41-59 (dl=0.02 m).



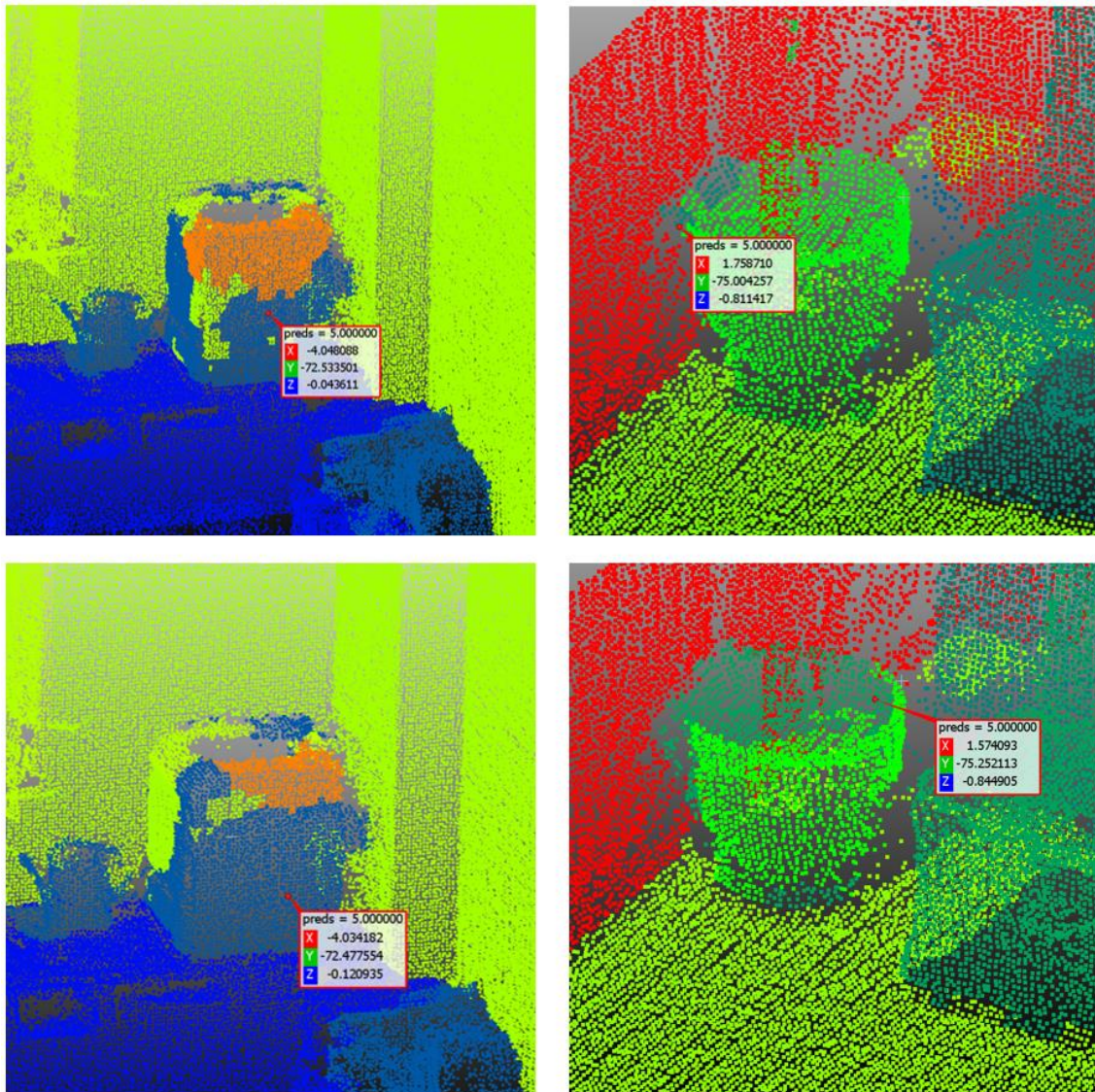


Figure 42: Predictions for printer (left) and dustbin (right) with different subsampling sizes

The same can be observed comparing the predictions for the class 'dustbin'. In the figure above, on the right, the predictions on the same section can be seen. The one at the top is from an experiment using the subsampling size 0.05 m (08-08-43), and the one at the bottom was subsampled using the subsampling size 0.03 m (12-16-37).

With a decreasing subsampling size, the portion of points classified as 'clutter' (preds=5.0) increases, which in turn lowers the performance of the class 'dustbin' because more points of that class are wrongfully classified.

The question is, how are small items, like dustbins and printers, wrongfully classified as 'clutter', but other small items, like smoke detectors, wall lamps, and exit signs, are not?

---

In the author's opinion, the reason is that the latter is mounted to the wall (e.g., wall lamps) or hanging from the ceiling (e.g., exit signs). Dustbins and printers, on the other hand, rest on a cabinet, table, or lay on the floor, which is also true for all objects classified as 'clutter'. With that said, the class 'clutter' poses an issue that is not easily eliminated, namely, that the class 'clutter' is a class for all items that either cannot be classified in the data preparation process or the occurrence of the item is so rare, that the class could not be sufficiently represented.

The first problem might be eliminated in the future by taking detailed pictures of the area in which the point cloud is collected. One could also increase the resolution of the point cloud so that objects become more recognizable. This, however, would also increase the number of classes within the point cloud.

The second problem, the rare occurrence of items, could be eliminated by simply increasing the size of the point cloud. One could also argue that most data sets will eventually be a collection of various smaller data sets that were all segmented and classified using the same convention of classes. But this still does not rule out the possibility of having items that are so rare that they only occur once, even in a large number of data set.

In the author's opinion, it will take a long time and a lot of data until the class 'clutter' can be eliminated. Instead, the author would suggest creating multiple classes like clutter that are an abstraction of a specific shape. These various kinds of clutter would each share some properties and features but would include different objects that might be rare within the point cloud. That way, the classes 'clutter' would be more specific, and other classes might not be wrongfully classified as much.

## 6 Conclusion

In conclusion, it can be said that it is imperative to classify objects consistently. Only a small number of wrongfully classified objects can have a severe impact on the performance of the classes involved.

It is also important to ensure a sufficient number of objects in the test set to receive a representative IoU of that class. It has been shown that it is favourable to aim for six or more items of the class for classes with individual objects. Compared to other data sets, the data set used for this project is still relatively small (e.g., ModelNet40)(Qi, Su, Mo, & Guibas, 2017).

The balance between specialization and generalization of classes must be carefully chosen because it might have a great impact on the performance of each class. If the distinction between two classes is not properly defined, the neural network performs poorly on the classification task. It must be decided if the distinction between objects is necessary. In some cases, it might be better to delay this task to post-processing.

The class 'clutter' poses the problem of having a variety of objects which leads to a lower performance of related classes. Especially when the subsampling size is decreased, the classes that are related to the class 'clutter' in object size and location are more often wrongfully classified as clutter.

### Recommendation for Action

In order to receive accurate mIoUs, further research should aim for larger data sets. It could also be considered to use a common standard for the data preparation in order to share data sets among multiple research teams. The premise is that the names of classes are predefined, and the classification and segmentation process follows the same guidelines.

According to the application, it is beneficial for some classes to be generalized and merged into one class, while the class 'clutter' could increase the performance of other classes if it is split into multiple classes with different properties. Only then can parameters be adjusted, and investment in computational cost becomes viable.

## References

- Jha, A., & Pillai, G. (2021). *Mastering PyTorch*. Packt Publishing.
- B1M, T. (Ed.). (2017). *The B1M*. (F. Mills, Producer) Retrieved August 13, 2021, from theb1m.com: <https://www.theb1m.com/video/what-is-scan-to-bim>
- Barron, J., & Google Research. (2019). *A General and Adaptive Robust Loss Function*.
- Borrmann, A., König, M., Koch, C., & Beetz, J. (2015). *Building Information Modeling*. Wiesbaden: Springer Fachmedien.
- Chollet, F. (2018). *Deep Learning mit Python und Keras*. mitp Verlag.
- CloudCompare. (n.d.). *Introduction*. Retrieved August 24, 2021, from [cloudcompare.org: http://www.cloudcompare.org/](http://www.cloudcompare.org/)
- Hugues, T. (Director). (n.d.). *KPConv Method* [Motion Picture]. Retrieved September 04, 2021, from [https://www.youtube.com/watch?v=uwuvp9mc\\_0o](https://www.youtube.com/watch?v=uwuvp9mc_0o)
- Hugues, T., Qi, C., Deschaud, J.-E., Marcotegui, B., Goulette, F., & Guibas, L. (2019). *KPConv: Flexible and Deformable Convolution for Point Clouds*.
- Johnson, T. (2017). *SCAN TO BIM IS A WIN FOR COST-EFFECTIVENESS IN POST CONSTRUCTION*. Retrieved September 08, 2021, from [blueentcad.com: https://www.bluentcad.com/press-release/post-construction-scan-to-bim/](https://www.bluentcad.com/press-release/post-construction-scan-to-bim/)
- Joshi, A. (2020). *Machine Learning and Artificial Intelligence*. Cham, Switzerland: Springer Nature Switzerland AG.
- Maalek, R. (2021). *Field Information Modeling (FIM)<sup>TM</sup>: Best Practices Using*. MDPI. doi:<https://doi.org/10.3390/rs13050967>
- Macher, H., Landes, T., & Grussenmeyer, P. (2017). *From Point Clouds to Building Information Models: 3D Semi-Automatic Reconstruction of Indoors*. Strasbourg, France: MDPI. Retrieved August 14, 2021, from <file:///C:/Users/simon/Downloads/applsci-07-01030.pdf>



- 
- Müller, A., & Guido, S. (2016). *Introduction to Machine Learning with Python*. O'Reilly Media, Inc. Retrieved from <https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch01.html#why-machine-learning>
- National Institute of Building Sciences (ed.). (2007). National Institute of Building Sciences: United States.
- Peltarion. (2021). *Loss functions*. Retrieved August 17, 2021, from peltarion.com: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions>
- Qi, C., Su, H., Mo, K., & Guibas, L. (2017). *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*.
- Qi, C., Yi, L., Su, H., & Guibas, L. (2017). *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*.
- Ramsundar, B., & Zadeh, R. (2018). *TensorFlow for Deep Learning*. O'REILLY MEDIA, INC.
- Rebala, G., Ravi, A., & Churiwala, S. (2019). *An Introduction to Machine Learning*. Cham, Switzerland: Springer Nature Switzerland AG.
- Tiu, E. (2019). *Metrics to Evaluate your Semantic Segmentation Model*. Retrieved August 25, 2021, from [towardsdatascience.com: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2](https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2)
- Vasilev, I. (2019). *Advanced Deep Learning with Python*. Packt Publishing.

## List of Appendices

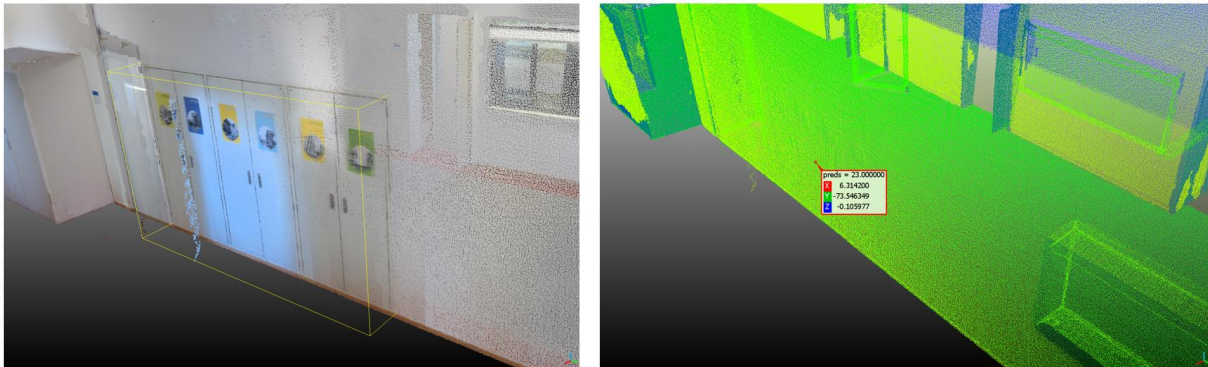
Appendix A: Experiment Results	59
Appendix B: Closet wrongfully classified as wall	60
Appendix C: Parameters Experiment 08-53-14	60

## Appendix A: Experiment Results

experiments_no	15-26-01	08-08-43	12-16-37		15-31-06	11-41-59	08-52-41 2 (larger test set)
DataSet	1	1	1		2	2	
Run_time, sec	44588,1	37643,18	77052,85			162422,99	
Run_time, h	9	3	4		76657,221	4	38298,511
dl subsampling	12,4	10,5	21,4		21,3	45,1	10,6
Max IoU	0,05	0,05	0,03		0,03	0,02	0,05
Mean_100 IoU	0,6203	0,6612	0,6608		0,7248	0,7413	0,6821
0: 'bookshelf',	0,5889	0,6211	0,6366		0,6937	0,7223	0,6519
1: 'cabinett',	0,5829	0,6143	0,5618	0: 'bookshelf',	0,5252	0,6082	0,6344
2: 'ceiling',	0,4667	0,4786	0,4899	1: 'cabinett',	0,4760	0,5848	0,6029
3: 'chair',	0,9625	0,9562	0,9612	2: 'ceiling',	0,9649	0,9683	0,9621
4: 'closet',	0,8782	0,9123	0,8949	3: 'chair',	0,7788	0,8680	0,9050
5: 'clutter',	0,3345	0,2915	0,3660	4: 'closet',	0,3519	0,3091	0,2864
6: 'deskarea',	0,3660	0,3602	0,4073	5: 'clutter',	0,3891	0,3921	0,4162
7: 'door',	0,7469	0,7289	0,7394	6: 'frame',	0,6686	0,6919	0,5080
8: 'dustbin',	0,7513	0,6437	0,7723	7: 'door',	0,6647	0,6553	0,6054
9: 'elevator',	0,6650	0,6218	0,5546	8: 'dustbin',	0,6693	0,5508	0,6629
10: 'exitsign',	0,5889	0,6210	0,6366	9: 'elevator',	0,6938	0,7223	0,6518
11: 'fan',	0,0060	0,6206	0,7555	10: 'exitsign',	0,4822	0,8341	0,2540
12: 'floor',	0,5544	0,6386	0,6225	11: 'fan',	0,5652	0,4612	0,5599
13: 'floorunit',	0,9838	0,9791	0,9730	12: 'floor',	0,9763	0,9798	0,9780
14: 'hallwaydoor',	0,1643	0,1607	0,1764	13: 'floorunit',	0,2340	0,4650	0,0836
15: 'monitor',	0,7666	0,7638	0,6858	14: 'hallwaydoor',	0,6829	0,7195	0,7095
16: 'noise',	0,8481	0,8755	0,9011	15: 'monitor',	0,9184	0,9262	0,8926
17: 'overhead-light',	0,4853	0,4374	0,5032	16: 'noise',	0,4653	0,5486	0,5389
18: 'plant',	0,9021	0,8885	0,9020	17: 'overhead-light',	0,8997	0,8993	0,9485
19: 'showcase',	0,5889	0,6210	0,6366	18: 'plant',	0,6938	0,7223	0,6518
20: 'smokedetector',	0,9029	0,9023	0,8786	19: 'showcase',	0,9115	0,9087	0,8707
21: 'standinglamp',	0,0061	0,2346	0,2644	20: 'smokedetector',	0,2913	0,3735	0,1856
22: 'table',	0,6898	0,7489	0,7828	21: 'standinglamp',	0,7409	0,7577	0,7336
23: 'wall',	0,3242	0,2846	0,2638	22: 'table',	0,7964	0,8345	0,8072
24: 'window',	0,8848	0,8680	0,8936	23: 'wall',	0,8895	0,8911	0,8824
25: 'heating',	0,3180	0,3442	0,2916	24: 'window',	0,8956	0,8832	0,8883
26: 'beam',	0,5458	0,5672	0,6723	25: 'heating',	0,6861	0,7101	0,5832
27: 'walllamp',	0,9520	0,9615	0,9669	26: 'beam',	0,9666	0,9735	0,9630
28: 'bench',	0,7425	0,8284	0,8497	27: 'walllamp',	0,8575	0,8846	0,8355
29: 'fireextinguisher',	0,5889	0,6210	0,6366	28: 'bench',	0,6938	0,7223	0,6518
30: 'fixedlight',	0,7707	0,7992	0,8679	29: 'fireextinguisher',	0,9032	0,9043	0,8448
31: 'handrail',	0,5889	0,6210	0,6366	30: 'fixedlight',	0,6938	0,7223	0,6518
32: 'steps',	0,5889	0,6210	0,6366	31: 'handrail',	0,6938	0,7223	0,6518
33: 'skylight',	0,8717	0,8878	0,8737	32: 'steps',	0,8821	0,8717	0,8852
	0,1249	0,1383	0,1254	33: 'skylight',	0,8380	0,8784	0,8540

34: 'couch',	0,5889	0,6210	0,6366	34: 'couch',	0,6938	0,7223	0,6518
35: 'printer',	0,0688	0,0950	0,0993	35: 'printer',	0,2249	0,1234	0,0545
36: 'overheadprojector',	0,5889	0,6210	0,6366	36: 'overheadprojector',	0,6938	0,7223	0,6518
37: 'partition',	0,5889	0,6210	0,6366	37: 'partition',	0,6938	0,7223	0,6518
38: 'sign',	0,5889	0,6210	0,6366	38: 'sign',	0,6938	0,7223	0,6518
39: 'desk lamp',	0,5889	0,6210	0,6366	39: 'desk lamp',	0,6938	0,7223	0,0016
40: 'hallway window',	0,5889	0,6210	0,6366	40: 'hallway window',	0,6938	0,7223	0,6518
41: 'airsystem'	0,5889	0,6210	0,6366	41: 'airsystem',	0,6938	0,7223	0,6518
				42: 'Ochair',	0,9093	0,9345	0,9224

## Appendix B: Closet wrongfully classified as wall



## Appendix C: Parameters Experiment 08-53-14

```

# -----#
# Parameters of the training session #
# -----#

# Input parameters
# *****

dataset = S3DIS
dataset_task = cloud_segmentation
num_classes = 44
in_points_dim = 3
in_features_dim = 4
in_radius = 1.500000
input_threads = 10

# Model parameters
# *****

architecture = simple resnetb resnetb_strided resnetb resnetb res-
netb_strided resnetb resnetb resnetb_strided resnetb_deformable res-
netb_deformable resnetb_deformable_strided resnetb_deformable res-
netb_deformable nearest_upsample unary nearest_upsample unary near-
est_upsample unary nearest_upsample unary
equivar_mode =
invar_mode =
num_layers = 5
first_features_dim = 128
use_batch_norm = 1
batch_norm_momentum = 0.020000

segmentation_ratio = 1.000000

```

```
# KPConv parameters
# *****

first_subsampling_dl = 0.030000
num_kernel_points = 15
conv_radius = 2.500000
deform_radius = 5.000000
fixed_kernel_points = center
KP_extent = 1.200000
KP_influence = linear
aggregation_mode = sum
modulated = 0
n_frames = 1
max_in_points = 0

max_val_points = 50000

val_radius = 51.000000

# Training parameters
# *****

learning_rate = 0.010000
momentum = 0.980000
grad_clip_norm = 100.000000

augment_symmetries = 1 0 0
augment_rotation = vertical
augment_noise = 0.001000
augment_occlusion = none
augment_occlusion_ratio = 0.200000
augment_occlusion_num = 1
augment_scale_anisotropic = 1
augment_scale_min = 0.900000
augment_scale_max = 1.100000
augment_color = 0.800000

weight_decay = 0.001000
segloss_balance = none
class_w =
deform_fitting_mode = point2point
deform_fitting_power = 1.000000
deform_lr_factor = 0.100000
repulse_extent = 1.200000
batch_num = 6
val_batch_num = 10
max_epoch = 500
epoch_steps = 500
validation_size = 50
checkpoint_gap = 50
```