Faculty IV - Electrical Engineering and Computer Science

TECHNISCHE UNIVERSITÄT BERLIN

Semantic Classification of 3D Point Cloud by Random Forest-based Feature Learning

Master Thesis in Electrical Engineering

Author:	Maosheng Tang
Supervisor:	Dr.-Ing. Ronny Hänsch
Reviewers:	Prof. Dr.-Ing. Olaf Hellwich, Prof. Dr. Marc Alexa
Submission Date:	January 29, 2019

I confirm that this master thesis in electrical engineering is my own work and I have documented all sources and material used.

Berlin, January 29, 2019 Maosheng Tang

Abstract

The semantic classification of a 3D point cloud refers to assign each point a semantic label. Unlike 2D images which usually have a well defined structure, a point cloud generally lacks of a clear one. Besides, with the development of measuring technology, it's not rare to see the number of points in a point cloud amounts to hundreds of millions, thus making the task challenging.

Previously, people would first extract features from the neighborhood of a point and then feed those feature vectors to a classifier to fulfill the task. In this work, a different method is proposed which tries to build an end-to-end framework, that makes use of the feature learning capability of a random forest. Instead of extracting the features explicitly and selecting the appropriate features by hand, the raw point cloud is fed directly into a random forest and let it choose the features that will best classify the data points. It then generates a label for the point being tested as output.

The framework is tested on a synthesized dataset first to prove its correctness and then applied to some publicly available benchmark point clouds, to show its effectiveness.

Zusammenfassung

Die semantische Klassifizierung einer 3D-Punktwolke bezieht sich darauf, jedem Punkt ein Semantikelabel zuzuweisen. Im Gegensatz zu 2D-Bildern, die normalerweise eine klar definierte Struktur darstellen, fehlt einer Punktwolke im Allgemeinen eine klare Struktur. Mit der Entwicklung der Messtechnik ist es immer häufiger zu sehen, dass die Anzahl der Punkte in einer Punktwolke hundert Millionen geworden ist, was die Aufgabe herausfordernd macht.

Im Forschungsstand haben die Wissenschafler zuerst Merkmale aus der Nachbarschaft eines Punktes extrahiert. Dann werden die daraus gewonnen Merkmal-Vektoren in einem Klassifikator eingeführt, um die Aufgabe zu erfüllen. In der vorliegenden Arbeit wird ein anderer Weg gegangen und versucht, ein End-to-End-Framework aufzubauen, das die Lernfähigkeit eines Random Forest ausnutzt. Anstatt die Merkmale explizit zu extrahieren, wird die originale Punktwolke direkt in einen Random Forest eingesetzt und mit der lassen sich die Merkmale auswählen, die die Punkte am besten klassifizieren können. Anschließend wird ein Label für jeden zu testenden Punkt generiert.

Die Richtigkeit der Methode wird als Erstes anhand eines synthetisierten Datensatzes nachgewiesen. Danach wird die Wirksamkeit der Methode mittels der veröffentlichen Datensätze aus der echten Welt geprüft.

Contents

1. Introduction

Almost ten years ago, Google launched its self-driving car project. The aim of the project is to build a reliable digital driver that is always alert and never distracted, so to make the driving more secure. It's apparently not the world's first trial to make a car that can drive itself, but it might be the world's first commercial car with fully autonomous driving functionality when being launched in the markets in the near future [16]. Once it is available in the market, it will no doubt enhance the mobility of everyone. It vastly eliminates the labor work when a human is driving the car and makes driving a simple thing. But the technology behind the scene is never simple, or to be exact, rather complicated. It's an interdisciplinary area where electrical engineering, mechanical engineering, software engineering, to name just a few, are all involved. Among all the challenging tasks, one of the fundamental problems is how to make an autonomous car perceive the environment it lives in. Just like a reliable human driver, it all depends on the driver's observation of the surroundings to decide whether he or she should accelerate, brake or steer the wheel, hence it's of essential meaning for a self-driving car to have the ability to "see" the world, otherwise reliable driving is not possible. This is where computer vision comes into play.

In the early days, researchers used sonar systems to enable the vehicle to gain understanding of the surroundings [26], just like a bat in the night. But as the roads become larger in number and so as the road users, the situation has become increasingly convoluted, especially in urban areas. Using a single kind of sensor alone cannot fulfill the requirements of reliable driving any more. It would be better to have the vehicle to directly see the environment. Hence engineers have outfit a self-driving car with different sensors in recent years, including cameras, radars and LiDARs, to ensure the car has a 360 degrees, blind-spot-free perception of the world.

While cameras provide the car with 2D images, LiDARs, a name short for LIght Detection And Ranging, give the car a 3D sensation of the real world. But unlike the images generated by cameras which are dense and structured in the form of regular grid arrangement, the data returned by a LiDAR sensor is in the form of point clouds, which lack clear structure, suffer from irregular sampling and come with varying

point density etc [17], see Figure 1.1.

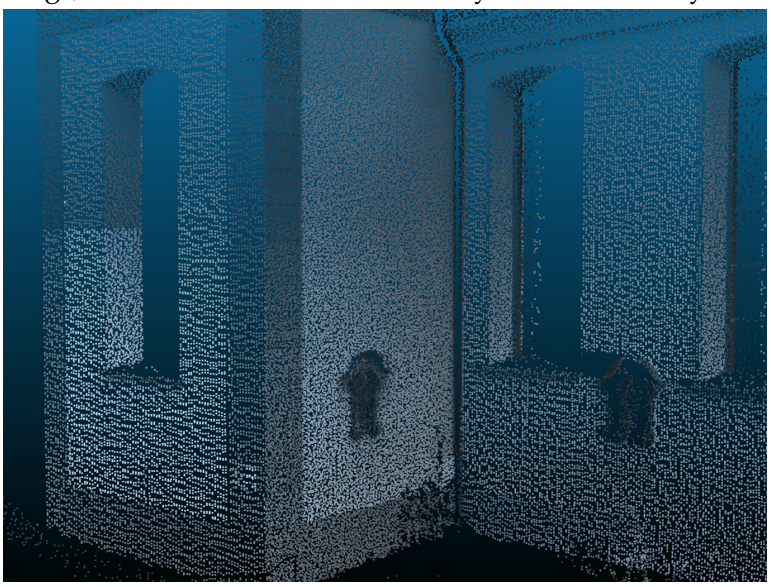
A LiDAR sensor emits hundreds of thousands beams of laser light in a second to the surroundings. The light will be reflected once it touches an object. The time elapse between firing-off and receiving is measured and then the coordinates of the object's surface are calculated. So essentially, a point cloud is merely a set of point coordinates. It gives quite accurate depth information of the scene in which the sensor lies. But it's neither dense nor structured. Besides, the number of points is massive. For example, the LiDAR sensor HDL-64E made by Velodyne which is seen in many of the prototype autonomous cars today can generate up to 2.2 million points per second. The fast generated large number of points on the one hand provides real-time and more precise description of the surrounding environment, on the other hand, it brings the difficulty in processing the data points.

By saying "perception of the world", we mean the car should distinguish a building from a tree, or another car from a pedestrian etc. Such a task is generally referred as classification. In 2D computer vision, this problem is well studied and recent Convolutional Neural Networks even beat an average human's performance in classifying objects [24]. But in the 3D case, this task remains challenging due to the characteristics of a point cloud we mentioned early.

There are broadly two kinds of workflow in classifying a point cloud. Roughly speaking, one of which extracts features from the point cloud and put the feature vectors into a classifier to achieve the goal. The other leverages the potential of a convolutional neural network either borrowed from a 2D version or built from scratch. In this work, we aim to build an end-to-end framework that directly consumes a point cloud and assigns each point in the cloud a class label by making use of the implicit feature learning capability of a random forest.

The remainder of this thesis is outlined as: Chapter 2 gives some background information and related work of the semantic classification of a point cloud. Chapter 3 presents the proposed approach in detail. Followed by that, evaluations of the approach are shown in Chapter 4. The final Chapter concludes this work.

(a) An overall view of a point cloud, in which different objects such as buildings, roads and trees can be visually identified easily.



(b) A close-up view of a section of the facade in the above point cloud. Note that unlike the images, the points in a point cloud are in reality not directly linked to each other and they appear only on the surface of the object.

Figure 1.1.: A point cloud showing various objects in two different views. Dataset from [19]

2. Related Work

In this chapter, some previous works related to the semantic classification of a point cloud are reviewed.

Like other research areas nowadays in computer vision, the strategies to tackle the problems bifurcate in two directions, one of which is by first extracting hand-crafted features then applying a classical classifier, while the other uses convolutional neural networks which don't rely on explicit feature extraction and provide an end-to-end solution to the presented problems.

2.1. Point Cloud Classification Using Handcrafted Features

Going the first way, there is already a standard processing pipeline being seen in a number of literature, for instance, [19], [33], [30], [2] etc.

The pipeline (see Figure 2.1) starts with the recovery of a local neighborhood of each point in the cloud. Possible neighborhoods are cylindrical, spherical or shaped by k nearest neighbors. The subsequent step is to extract geometric features which describe the spatial arrangement of points within the local neighborhoods. The features are categorized into *local 3D shape features* and *geometric 3D properties*. The former ones rely on the eigenvalues derived from the local 3D structure tensor of the

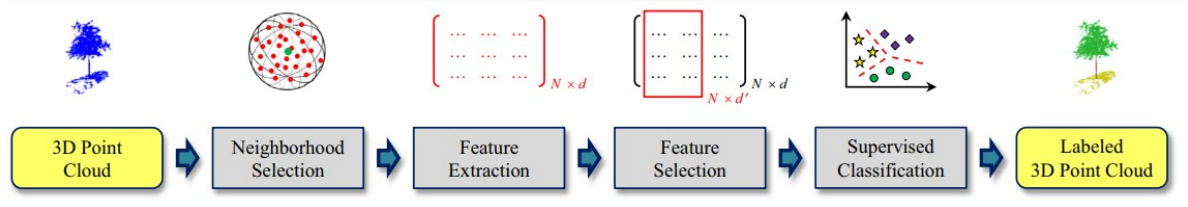


Figure 2.1.: General pipeline for processing a 3D point cloud using handcrafted features [49]

considered point and its neighboring points [9]. They demonstrate the linear, planar etc. properties of the neighborhood. The latter ones can usually directly obtained from the raw data, such as the absolute height of the point, the distance between the point itself and the farthest point in the neighborhood, and so like. Further, additional 2D features may also be considered. Such 2D features are motivated by the fact that specific assumptions about the point distribution can be made.The various features together form a feature vector that describes a point [50].

Apart from the relatively primitive features, more advanced features are also seen in some literature. Signature of Histogram of Orientations (SHOT) proposed by Tombari et al. [44] encodes histograms of the angle differences between normals of the points in the neighborhood and that of the considered point, giving a more representative description of the local structure of the surface compared to plain 3D coordinates. But its sensitivity to varying point density makes this method less practical in many situations [18]. Hackel et al. [20] even concluded that by incorporating the SHOT and SC3D [13] didn't enhance the accuracy too much but way too computationally demanding, thus should not be considered as feature descriptors in large and dense point clouds.

At the last step of this standard processing chain, feature vectors are fed into a classifier, such as Gradient Boosted Trees as in experiments conducted by Becker et al. [2] or more commonly, a Random Forest as seen in various works for 3D point cloud classification. Sometimes, it is necessary to add a selection step before classifying. It this step, features that are most distinguishable and help with the classification performance are chosen in a heuristic manner. To sieve out some features will on the one hand, lower the computational demand during predicting. On the other hand, the probable curse of dimensionality can be avoided during training. In [48] Weinmann et al. discussed which features are the most relevant to classifying a point cloud based on different metrics. In addition, it also shows that the classification performance is highly influenced by the sampling density, the chosen neighborhood type and the used classifier.

It's worth mentioning that along this "standardized" workflow, there are researchers (e.g. [38] [15] and [41]) who use elevation images to project the 3D point cloud into a 2D representative. Then the 2D images are segmented with algorithms like watershed. Each segment is given a label afterwards and back projected to the 3D point cloud.

2.2. Point Cloud Classification by Exploiting the Power of Convolutional Neural Networks

With the resurgence of Neural Networks in the past few years, especially after Convolutional Neural Networks enjoy gigantic successes in solving problems from the 2D computer vision domain, researchers begin to investigate the application of CNNs to point clouds. Boulch et al. [4] proposed a three-step approach that connected the 3D problem to a 2D CNN. In the first step, they projected the 3D data into multiple 2D images (called "snapshots") by looking at the point cloud from many perspective of views. Then pixel-wise labelling of each pair of 2D snapshots were performed. In this step, rather than building a novel convolutional network, already existing CNN architectures for 2D image segmentation were applied. The last step is to back project the label predictions to the 3D space. This three-step approach shares the ideas of those mentioned in last section except one uses traditional methods for 2D image classification while the other makes use of neural networks. Their work has achieved relatively satisfying result and outperformed some previous works that using handcrafted features.

Other than considering to solve the problem with 2D CNNs, authors of [27], [51], [25], [31] used similar strategies to directly classify a point cloud with the help of 3D convolutional networks. Their methods began with voxelizing the point cloud, which means, a cubic bounding box was around a chosen center point and the cubic box was further subdivided into many cells or voxels. After obtaining the voxels, 3D tensors were created and fed into a 3D CNN designed for objection classification. The 3D version CNNs generally share the same pattern. For instance, the model found in [25] was composed of 2 cascaded concatenation of a 3D convolutional layer and a 3D max-pooling layer, followed by a fully-connected layer and a softmax classifier as the final layer. Using this model, authors of [25] achieved an overall precision of 93.0% and claimed the training phase was highly efficient with regard to both time and space. However, memory consumption for voxel-based approaches is usually very large.

Qi et al. [34] proposed a unified architecture that directly consumes the point cloud (in the basic setting, only the coordinates of the points are used) and gives each point a label as output. What's different from the above-mentioned methods lies in the structure of their networks, where fully-connected layer and pooling layers have replaced the convolutional layers, which means each point was processed identically and independently. They demonstrated that their network can learn a set of optimization functions with the ability to automatically select interesting points and encode the reason for the selection. Later, they improved their learning layers [35] that

can adaptively combine features from multiple scales. This enhanced "PointNet++" is able to learn point clouds with varying densities and set a new state-of-the-art record in point cloud classification on several challenging benchmarks.

2.3. Image Classification with Random Forests

Though there are many differences between 3D and 2D objection classification, they are partially related. Methods developed for solving 2D problems can be borrowed to the 3D world, like we have seen in the previous section.

In 2D image classification, Hänsch and Hellwich [23] proposed a novel way to classify PolSAR images that takes advantage of the implicit feature learning and selection ability of a random forest. Their approach has inspired this work, so a brief review here is necessary.

Compared to normal images, PolSAR images are more complicated in the sense that each pixel of a PolSAR image consists of vectors in complex number domain. Thus traditional approaches for image classification will generally fail on such data if applied directly. One way to overcome the difficulty is to extract real-valued features, another is to model the statistical properties of the PolSAR data. However, either way suffers a series of shortcomings, for example, creating descriptive features is not an easy job, computing diverse and descriptive set of features increases the computational burden, projection of complex-valued data to real domain brings loss of information etc.

By extending the ideas of using image patches [12] instead of high dimensional feature vectors at each node in a decision tree, authors of [23] adapted random forests which combine feature learning and classification ability into a unified framework that can be directly applicable to complex-valued PolSAR data. Without explicit feature extraction, computation time and memory are saved. Furthermore, this framework is generic as no task-specific features being manually constructed so it can be applied to different datasets acquired by different sensors without redesigning the processing chain but by simply retraining the model.

3. Methodology

In the last chapter, some related works are briefly summarized. The ideas in [23] are found original and the most interesting, hence this thesis will borrow their ideas and adapt it to the 3D point cloud classification problem, i.e., to classify a 3D point cloud by making use of random forest-based feature learning. To be specific, we strive to build an end-to-end framework, which requires no data preprocessing, no explicit feature extraction relating to the considered point and no postprocessing. This framework takes a raw point cloud as input and gives each point a corresponding label as output. This chapter devotes to the description of the methods we propose and implementation details in each step of the workflow (shown in Figure 3.1) that guides the work.

3.1. Neighborhood Recovering

As mentioned earlier, a point cloud differs from an image in many aspects. The prominent properties of a point cloud include [34]:

- Unordered. Pixels of an image are arranged in a regular grid, in other words, they form a structured array. But points in a point cloud have no specific order. A permutation in $N!$ renders the same point cloud (N is the number of points in the point cloud).
- Interaction among points. As all the points in a point cloud are from a space with a distance metric, a single point is meaningless but with its neighboring points, a meaningful subset can be formed.
- Invariance under transformations. A point cloud after certain transformations (rotation and/or translation) still represents the same scene.

In light of the properties, it is necessary to build a neighborhood for each point such that the local structure and interactions among points can be captured. The aim of the first stage in our workflow is formulated as: **given a query data point, find its**

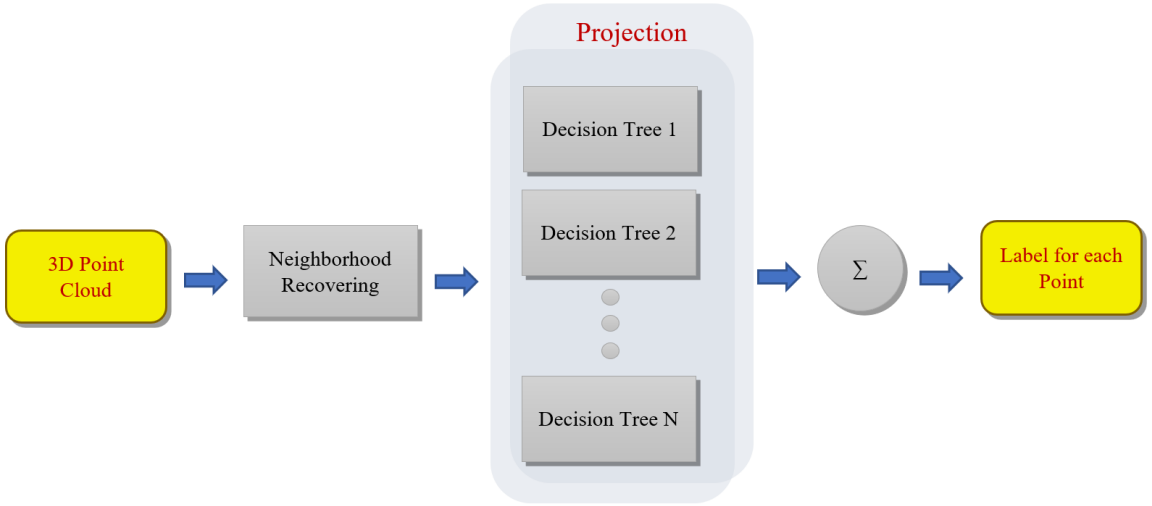


Figure 3.1.: Workflow of this work consists of two core steps: neighborhood recovering and random forest feature learning

nearest neighboring points in the cloud. To achieve this goal, we need 1) a suitable neighborhood definition; 2) an efficient way to retrieve the neighbors.

3.1.1. Neighborhood Definition

Typically, the definition of a local neighborhood has three variants:

- cylindrical neighborhood $\mathcal{N}_c$, which is formed by all the 3D points whose 2D projections on a plane are within a circle of fixed radius $r_c \in \mathbb{R}$ around the projection of query point $\mathbf{p}$ [10]
- spherical neighborhood $\mathcal{N}_s$ with a fixed radius $r_s \in \mathbb{R}$
- spherical neighborhood $\mathcal{N}_k$ with fixed number of nearest neighbors $k \in \mathbb{N}$

To search for a suitable radius of either a cylindrical $\mathcal{N}_c$ or a spherical $\mathcal{N}_s$ neighborhood, a prior knowledge of the scene is usually helpful, otherwise it needs to be searched heuristically for each dataset. An obvious drawback should be noted here. Because the point density in a cloud is usually inconsistent, when we use a fixed radius, it is possible that there are too many points in certain neighborhoods in regions with very dense points while too few points in some other neighborhoods where the density is sparse. The former case could lead to high computational burden while in the latter case, computation of local 3D shape features may fail. Moreover, a

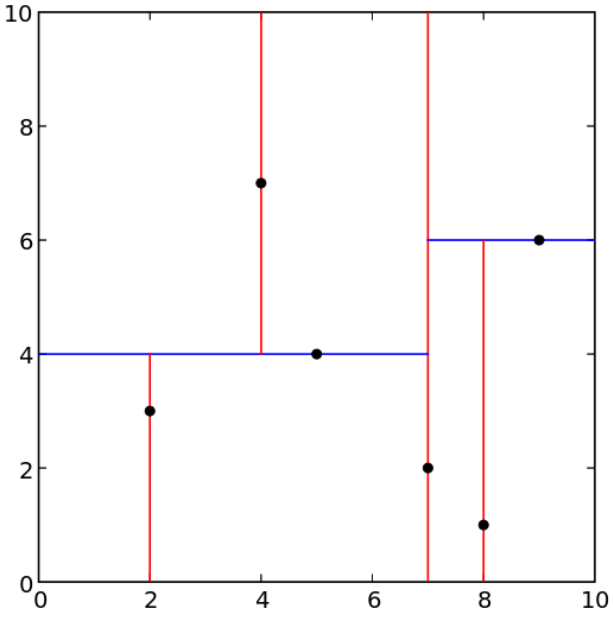
cylindrical neighborhood needs more than one parameters, namely the radius of the projected 2D circle, the height of the cylinder and the projection plane. This definition makes recovering such neighborhoods less practical in large point clouds. So the spherical neighborhood with fixed number of points is favored. Since the number of points in a neighborhood is fixed and the point density is varying, which means the volume of all neighborhoods are varying accordingly. We can use this property as an additional feature.

3.1.2. Neighboring Points Retrieving

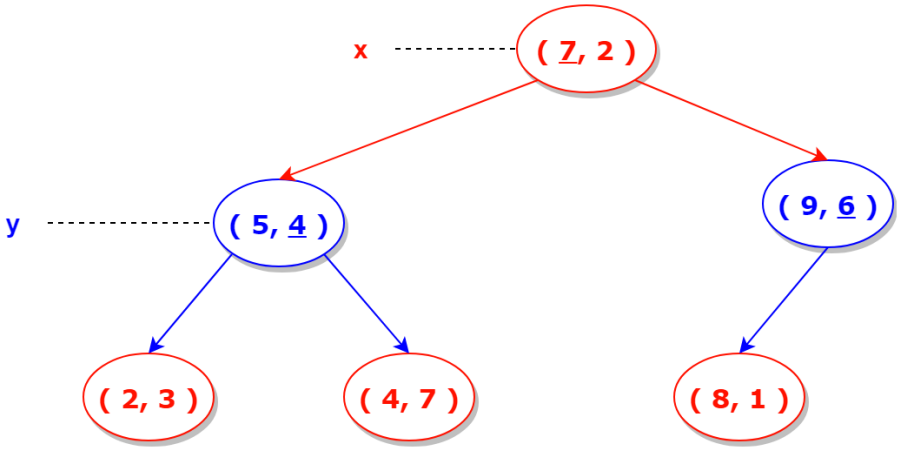
After we define a neighborhood, what follows next is how to retrieve those neighbors. Although the points we consider are low dimensional, or to be exactly, $\mathbf{p}_i \in \mathbb{R}^3$, the number of which is very huge. A very efficient way to get those neighbors is thus desired. Formally, this problem is stated as: given a point cloud $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ in a 3D Euclidean vector space, find k points in $\mathcal{P}$ to a query point $\mathbf{q}$ such that the Euclidean distance is the shortest.

If the number of points is not big, a brute-force search is generally enough, which computes the distances between all pairs of points in the dataset and the complexity is $O(Nd)$, given N data points with d dimensions for each data point. Since the number of points to be searched is huge in our point cloud case, and the data is in low dimension (just 3 dimensions, i.e. the spatial coordinates of each point, the other properties like colors need not be taken into account when searching nearest neighboring points), we favor a more efficient algorithm called k-d tree [11] that can accurately find k nearest neighbors of a query point in low dimensional space. The average complexity of building a k-d tree is $O(N \log(N))$ while finding one nearest neighbor in it takes only $O(\log(N))$ [47]. Concisely speaking, a k-d tree is of a binary tree structure. Each non-leaf node of the tree recursively splits the space into two halves, points to the left of the hyperplane go to the left child and those to the right go to the right child. Usually a hyperplane perpendicular to a selected dimension's axis at the node is chosen.

During construction, as one moves down the tree, axes used to select the splitting planes alternates [3]. For instance, in a 2-dimensional tree, if we choose a hyperplane (a line) perpendicular to x-axis at the root, both children of the root node are made to choose a hyperplane perpendicular to y-axis. The children of the nodes (i.e. the grandchildren of the root node) will again choose a hyperplane perpendicular to x-axis. We choose the median of the dimension being considered of all the points at that specific node as a criterion to move the points either to the left subtree (when the value is less than the median) or right subtree (when the value greater than the



(a) Visualization of splitting hyperplanes



(b) Tree representation

Figure 3.2.: Construction of a 2-dimensional k-d tree

median).

For a concrete example, suppose we have a set of points {(7,2), (5,4), (9,6), (2,3), (4,7), (8,1)}. At root node, we consider the x-coordinate of each point, the median of which is 7. A hyperplane perpendicular to x-axis and going through point (7,2) is created. Points whose x-coordinates are less than 7 are moved to the left subtree of the node and those greater than 7 to the right subtree. We have {(5,4), (2,3), (4,7)} for the left subtree and {(9,6), (8,1)} for the right subtree. At both children nodes of the root, we consider the y-coordinates. Since the median of y-coordinates of points in the left subtree is 4 (of right is 9), a hyperplane perpendicular to y and going through point (5,4) (or through point(9,6) respectively, shown as the two blue lines in Figure 3.2a), point (2,3) is move to the left and (4,7) to the right. We illustrate the process in Figure 3.2 and summarize the algorithm in Algorithm 1.

Algorithm 1 Building a k-d tree

Require: points in the dataset $\mathcal{P}$

kdtree($\mathcal{P}$, h)
if $\mathcal{P}$ has only one point **then**
    return a leaf node containing $\mathcal{P}$
else
    axis $\leftarrow h\%d$ {h is the current height and d is the number of dimensions}
    $m \leftarrow$ median($\mathcal{P}$[axis])
    Nodes.left $\leftarrow$ kdtree($\mathcal{P}_L$[axis]$< m$, h+1)
    Nodes.right $\leftarrow$ kdtree($\mathcal{P}_R$[axis]$> m$, h+1)
end if
return Nodes

Once a k-d tree is built, given a query point, we can quickly find its nearest neighbor in the tree as follows.

1. Compare the corresponding split dimension of the query point and that of the point at each node and shift the query point either to the left or right accordingly.
2. Set the first leaf node the query point reaches as the "current best".
3. Go one level up. Check if the query point lies closer to the splitting hyperplane than it does to the "current best". If so, update the "current best" point.
4. Check whether any point in the other subtree is closer to the query point.

During each search, the larger parts of the search space is excluded by the algorithm, therefore the search is very efficient.

There exists already a library available called FLANN [32] that is designed for nearest

neighbor search written in C++. It provides both accurate search realization using the algorithm we presented before as well as approximate search, which is more suitable for search in high dimensional space.

3.2. Random Forest Construction

The second stage of our work is to train a classifier to do the classification. Since we haven't explicitly extract any features, we will exploit the feature learning and selection ability of a Random Forest and let it do the remaining job for us.

Random Forests have been used in solving many computer vision related problems and enjoy great successes since its invention by Breiman [6] as we have shown in chapter 2. It is an ensemble method that trains multiple weak learners to build a stronger one. In general sense, weak learners can be decision trees, neural networks or other learning algorithms. In the specific case of a Random Forest, the weak learners used are the decision trees. In contrast to a single decision tree which is simple and has good interpretability, but also easily suffers from over-fitting or high variance, a random forest leverages the advantages of each individual trees and avoids their limitations [23].

3.2.1. Binary Decision Trees

The fundamental component of a random forest is a binary decision tree. It is a basic algorithm suitable for both classification and regression purposes. In this work, we keep our attention to the classification decision trees. It doesn't look much different than the k-d tree we created for nearest neighbor search except for the way we divide the dataset at each node.

In the k-d tree, we split the dataset according to the median value of alternately selected dimension (or axis). However, when growing a binary decision tree, we try to split the dataset on a certain feature of all known features by asking a question about the chosen feature, for the purpose that the data points in each of the resulting child nodes are as homogeneous as possible in the sense of the their labels. The feature and the question are stored in this node. This procedure is repeated recursively for each of the child nodes until a stopping criterion is met, when no further splitting is necessary and a leaf node is grown. We do this procedure because, as its name suggests, we will have the decision tree to make a decision for us in the end, that is, to predict the class label of new instances. The to-be-predicted data point will

eventually arrive at one of the leaf nodes according to the set of questions asked by the splitting nodes. The data point will be assigned a class label that's most frequent in the corresponding leaf node.

The set of questions is most naturally thought as a set of "if-then" rules. The test conditions for "if" statements are the features selected in the internal nodes. "If" test results are true, "then" the data point is shifted to the left child node, otherwise it is shifted to the right child node. Recursive application of these tests will lead to the creation of a binary tree. The tree stops growing when either a preset maximal tree height is reached or the node can't be split any further. For example, all the points in the node belong to the same class or there are too few points in this node that further splitting is not necessary. The set of rules obtained this way is complete and mutually exclusive, meaning that when the tests terminate, the data point will fall exactly into only one leaf node. We express the "if-then" rules in the form $x_i < \theta?$, where x_i is a feature of $\mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^d$.

There are two questions still remain to be answered. One is how do we measure the homogeneity of each node. The other is what's the proper value for θ.

To answer the first question, we should find a measure $I(\cdot)$ that is largest when all the data points are equally distributed in this node and smallest when the data points are of the same class. One possible measure could be the Gini criterion [7]. There are other measure definitions like entropy gain used by ID3 algorithm [36] or entropy gain ratio found in C4.5 algorithm [37], but in this work, we focus on the Gini criterion. Given the labeled dataset at a certain node is $\mathcal{P} = \{(\mathbf{p}_1, c_1), (\mathbf{p}_2, c_2), ..., (\mathbf{p}_n, c_n)\}$, with c_i being the respective class labels, the Gini criterion is given by:

$$I(\mathcal{P}) = 1 - \sum_{i=1}^{C} \left(\frac{|\mathcal{P}_i|}{|\mathcal{P}|} \right)^2 \tag{3.2.1}$$

Where $|\mathcal{P}_i|$ is the number of samples in the i-th class and $|\mathcal{P}| = \sum_{i=1}^{C} |\mathcal{P}_i|$ of the **current node**.

When we have the measure for node homogeneity at hand, we need an additional formula to evaluate which one of all the considered features that can lead to a split where both child nodes are as pure as possible. The formula for that purpose is defined as follows and is called decrease in impurity:

$$\Delta I(\mathcal{P}, f) = I(\mathcal{P}) - p_L I(\mathcal{P}_L) - p_R I(\mathcal{P}_R) \tag{3.2.2}$$

Where $p_L = \frac{|\mathcal{P}_L|}{|\mathcal{P}|}$ and p_R have a similar definition, i.e. proportions of the samples in the current node that go to the left child node and right child node respectively.

The answer to the second question is crucial to the successfulness of a decision tree. As we'll see later, the θ values or, thresholds, have a clear influence on the structure of a tree. Several approaches have been proposed to choose the threshold for each test function. One can either randomly sample a value from the corresponding attribute of the training dataset or use highly optimized values. The work of Hänsch and Hellwich [22] evaluated possible candidates for setting the thresholds and their effectiveness. They concluded that the median is a sufficiently stable choice. The tree grown with median as splitting threshold is balanced and requires less computational power.

It's worth pointing out that, during training, a decision tree splits the dataset recursively, in an attempt to classify the training samples as correctly as possible. This behavior could lead to too many branches in the tree that incorporate some characteristics found only in the training dataset but not in the general data points. When tests being carried out on unseen samples, the result is usually sub-optimal. This "overfitting" phenomenon of a binary decision tree is not rare to see and will usually be treated with a technique called "pruning" [37].

3.2.2. From Trees to Forest

The idea of combining the results from multiple decision trees resembles how a person makes a decision in daily life. When a person needs to make an important decision, naturally he would go and consult different experts in related fields. As every expert has his own expertise and perspective of view, the overall suggestions from all experts are likely to cover the whole spectrum of the advice seeker's problem. With all the information, it is more probable for the person to make an informed decision.

But how are the trees combined and how do we use the individual prediction of each tree? The random forest [6] proposed a solution.

If the trees are simply repeated multiple times, the result won't be different from a single tree. But if some randomness is injected in the component trees, making them different from one another, the correlation among the trees will intuitively be reduced, which in turn, the generalization and robustness of the forest will be enhanced [8]. Two of the most popular ways to introduce randomness into the trees (therefore the name random forest) are bagging (which stands for Booststrap Aggregating) [5] and randomized node optimization (see e.g. in [1]).

Bagging was used to decrease possible overfitting of component trees and to enhance the generalization capabilities of random forests. It is tree-level randomness. The core

concept of bagging is to train the trees on different sub-datasets which are randomly downsampled **with replacement** from the original dataset. Given the training set of a point cloud $\mathcal{P}$, for each tree $t(1 \leq t \leq T)$ in the forest, we can train a base learner on the consisting N' samples drawn randomly from the entire N samples, the prediction of each base learner are aggregated in a way we'll explain later. The algorithm is shown in Algorithm 2.

Algorithm 2 Bagging

Require: all the points in training set $\mathcal{P}$, a base learner L, number of trees T
Ensure: an aggregated strong learner $H(\mathbf{p})$
1: **for** $t = 1$ **to** T **do**
2: $\quad \mathcal{P}_t \leftarrow \text{sampleWithReplacement}(\mathcal{P}, N')$
3: $\quad h_t \leftarrow \mathsf{L}(\mathcal{P}, \mathcal{P}_t)$
4: **end for**
5: $H\left(\mathbf{p} \leftarrow \arg\max_{c \in C} \sum_{t=1}^{T} \delta(h_t(\mathbf{p}) = c)\right)$

The other kind randomness is injected at node-level. Recall that all the features (d) are used for the node splitting tests in a binary decision tree, when it comes to a random forest, only a randomly sampled fraction ($1 \leq d' \leq d$) of the whole features will be considered. Breiman [6] recommended the number of features being selected to be $k = \log_2 d$.

We mentioned just now the final prediction of individual learners should be aggregated somehow. Actually there are several schemes to do this. In Line 5 of Algorithm 2, a primitive plurality voting scheme is adopted by the bagging algorithm to leverage the individual prediction of each tree. That is, the tested data point is assigned a such label that the most of the trees vote for. Other voting schemes include majority voting and weighted voting [52]. Majority voting requires the final class receives at least half of votes. Weighted voting gives each classifier a weight with stronger ones more weights and outputs the prediction based on the weighted average voting result.

Instead of counting the absolute number of samples of each class in the leaf nodes, more generally, we parameterize the leaf nodes with a posterior class distribution. This is achieved easily by a frequency estimator. Given the total number of samples in a leaf node is N'', the number of samples belonging to class $c(1 \leq c \leq C)$ is N''_c. Then the probability of a point that arrives at this node being classified as $C = c$ is:

$$P(C = c|\mathbf{p}) = \frac{N''_c}{N''} \tag{3.2.3}$$

Anyhow, this simple formula hasn't taken the class prior into consideration. For example, we have in the root node 2000 samples of class A but 1000 of class B. Ten samples from either class arrive at a certain leaf node. Can we say that, when a test data point falls into this leaf node, it is equally likely this point comes from either class A or class B? Not really. Because there are twice as many samples of class A than in class B. Intuition tells us it is unfair for class B to share the same odds as class A in this node because it uses less samples (or more effort) to produce the same result as class A. We ought to assign the test point class label B twice likely than assigning it a class label A. Therefore we should weight the local sample number with the inverse of class prior and normalize it as:

$$p(c|\mathbf{p}) = \frac{\frac{N_c''}{N''} \cdot \frac{N'}{N_c'}}{\sum_{c=1}^{C} \frac{N_c''}{N''} \cdot \frac{N'}{N_c'}} \tag{3.2.4}$$

Where N' and N_c' are the number of data points in each bag (i.e. after a random sampling with replacement of the original dataset $\mathcal{P}^{N\times d}$) and corresponding number of data points in each class. N'' and N_c'' have similar meaning except they stand for data points in a certain leaf node.

Back to the previous example, we have:

$$p(c = A|\mathbf{p}) = \frac{\frac{10}{10+10} \cdot \frac{2000+1000}{2000}}{\frac{10}{10+10} \cdot \frac{2000+1000}{2000} + \frac{10}{10+10} \cdot \frac{2000+1000}{1000}} = \frac{1}{3}$$

Similarly, we have $p(c = B|\mathbf{p}) = \frac{2}{3}$. The result justifies our intuition. By incorporating the class prior, the potential data imbalance problem that many machine learning algorithms encounter is solved.

With the class posterior at hand, another simple averaging scheme for predicting then arises:

$$p(c|\mathbf{p}) = \frac{1}{T} \sum_{t=1}^{T} p_t(c|\mathbf{p}) \tag{3.2.5}$$

3.3. Point Cloud Feature Set

In the standard Random Forest formulation, the input is a collection of feature vectors. Node splitting is based on one of the given features. As is seen in the previous works, to classify a point cloud, one of the workflows begins with extraction relevant features and putting them into a vector to represent the point in consideration, followed by feeding the feature vectors to a random forest to do the real classification job.

3.3.1. Inspiration from 2D Image Segmentation

In Ch. 2, we briefly mentioned that Hänsch and Hellwich [23] proposed a way to process PolSAR images using random forest without explicit feature extraction. The core idea is to project image patches from higher dimensional space to a scalar, so that within each node, a test in the form $x < \theta$? can be performed. This approach relies on the implicit feature learning and selecting capabilities of a random forest. Their work is an extension of the researches conducted by Lepetit and Fua [28] as well as Fröhlich et al. [12], where the authors compared various attributes generated within a local feature map patch. The feature maps were computed beforehand, which were nothing but matrices that stored a value for each pixel of the original image. One possible feature map could be the red channel of the image, for instance. The attributes were from very primitive ones like the differences of two random pixels to some relatively advanced features (the Haar-like [46]). We list an example and explain their implementation details in the caption in Figure 3.3.

3.3.2. Creating Basic 3D Features

The methods used in 2D computer vision can usually be adapted to 3D scenes, as we discussed in the related work chapter. It is their way of thinking gives us the inspiration for this work.

Since no explicit features should be extracted, the key task of the work becomes how to design the node test functions. Just like what the authors of [12] have done in their work, we derive possible node test functions from 3D shape primitives and some 2D representations as well as radiometric properties.

Local 3D Shape Features

The local 3D shape features are obtained from the eigenvalues of the 3D structure tensor derived from a considered point $\mathbf{p}$ with its neighbors. They are rather intuitive and represented by a scalar value [48], which is suitable for the test node functions.

The 3D structure tensor of a given point $\mathbf{p}$ within its neighborhood $\mathcal{N}^{k\times d}$ is:

$$C = \frac{1}{k}\sum_{i=1}^{k}(\mathbf{p}_i - \hat{\mathbf{p}})(\mathbf{p}_i - \hat{\mathbf{p}})^{\mathrm{T}} \tag{3.3.1}$$

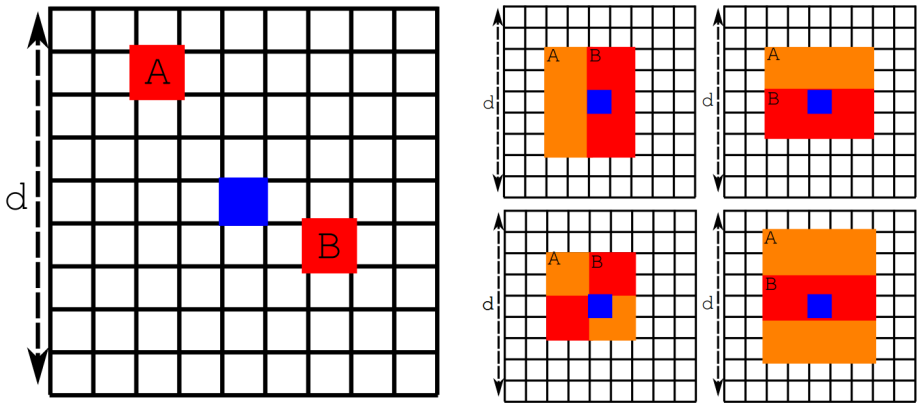


Figure 3.3.: Getting attributes from a local feature map patch. On the left side, a random pixel position (blue pixel) is selected from the feature map and a random sized window ($d \times d$) is built around it. Within this window, two other pixels (red pixels) are also randomly picked. The (absolute) difference between the two pixels or the sum of the two pixels serves as a possible node test function. On the right side, a local patch is built the same way as described before. But within this patch, two vertical strips are used (top-left corner). The sum of pixels in strip A is subtracted from that in strip B, leading to another node test function. Other variants are also shown in the top-right corner (horizontal Haar-like), bottom-left corner (four-rectangle Haar-like) and bottom-right corner (three-rectangle Haar-like) [12].

where $\hat{\mathbf{p}} = \arg\min_{\mathbf{p}} \sum_{i=1}^{k} \|\mathbf{p}_i - \mathbf{p}\|$ is the centroid of $\mathcal{N}$. With help of eigen decomposition of C, three eigenvalues $\lambda_1, \lambda_2, \lambda_3$ (after being sorted in descending order) are obtained, based on which, the shape features are defined as:

- Linearity: $L_\lambda = \dfrac{\lambda_1 - \lambda_2}{\lambda_1}$
- Planarity: $P_\lambda = \dfrac{\lambda_2 - \lambda_3}{\lambda_1}$
- Sphericity: $S_\lambda = \dfrac{\lambda_3}{\lambda_1}$
- Anisotropy: $A_\lambda = \dfrac{\lambda_1 - \lambda_3}{\lambda_1}$
- Omnivariance: $O_\lambda = (\lambda_1 \cdot \lambda_2 \cdot \lambda_3)^{\frac{1}{3}}$
- Eigenentropy: $E_\lambda = -\sum_{i=1}^{3} \lambda_i \cdot \ln(\lambda_i)$
- Sum of Eigenvalues: $\Sigma_\lambda = \lambda_1 + \lambda_2 + \lambda_3$
- Change of Curvature: $C_\lambda = \dfrac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$

The first three is rather intuitive and provides information as their names suggest, that is, the presence of a linear 1D structure, a planar 2D structure or a volumetric 3D structure. The rest several measures encapsulate special geometric properties as discussed in [45] and [29].

Other 3D shape features are based on the corresponding eigenvectors of C. Denote the largest eigenvector as $\mathbf{e}_1$, the second largest as $\mathbf{e}_2$ and the smallest as $\mathbf{e}_3$, the possible features include:

- Verticality ($\times 2$): $\left| \dfrac{\pi}{2} - \text{angle}(\mathbf{e}_i, \mathbf{e}_z) \right|_{i \in \{0,2\}}$
- Absolute Moment ($\times 6$): $\dfrac{1}{k} \sum \left| \langle \mathbf{p} - \hat{\mathbf{p}}, \mathbf{e}_i \rangle^\beta \right|_{i \in \{0,1,2\}}$
- Vertical Moment ($\times 2$): $\dfrac{1}{k} \sum \left| \langle \mathbf{p} - \hat{\mathbf{p}}, \mathbf{e}_z \rangle^\beta \right|$

Where $\mathbf{e}_z$ is the unit vector along the z axis, β is either 1 or 2. The verticality of the first eigenvectors encodes the verticality of linear objects while the verticality of the third eigenvectors encodes the verticality of the normal vector of planar objects [43].

The first and second order moments around the eigenvectors and the unit z vectors help with the classification of crease edges and occlusion boundaries [20].

Geometric 3D Properties

Geometric 3D properties are based on the raw data points, they reflect the basic geometry of a neighborhood. Such properties include:

- Height of $\mathbf{p}$: $\mathbf{p}_z$
- Longest Distance within Neighborhood: $D_{3D} = \arg\max_{i \in k} \|\mathbf{p}_i - \mathbf{p}\|$
- Local Point Density: $\frac{k}{\frac{4}{3}\pi D_{3D}^3}$
- Vertical Range: $z_{\max} - z_{\min}$

Radiometric Properties

With the development of sensing technology, it is not rare to see that a point cloud comes with color information. Therefore, this information can also be utilized in the node test functions.

- Average Color: $\frac{1}{|\mathcal{N}|} \sum_{i=1}^{k} c_i$
- Color Variance: $\frac{1}{|\mathcal{N}-1|} \sum_{i=1}^{k} (c - \bar{c})^2$
- Color of a Single Channel: $c_x, x \in \{R, G, B\}$

Note that usually the color information is given in RGB color space, which is simple but also has some drawbacks, for example, visually similar colors may not close to each other in terms of Euclidean distance [21]. So we transform the colors from RGB space to HSV space (Hue, Saturation, Value). In this space, the intensity of a pixel is expressed independently from its chromaticity. We simply add this color space description to each existing point and let the random forest to decide which one it will eventually use.

It's worth mentioning that there are other very advanced 3D shape descriptors existing. Representative ones include Shape Context 3D (SC3D) [13], Signature of Histogram of Orientations (SHOT) [44], Fast Point Feature Histograms (FPFH) [40] etc. Although these histogram-based descriptors provide more descriptive power in

some complex situations, the computational load is very heavy. As we will use the node test functions enormous times during tree construction, the functions should be as simple as possible and easy to compute, therefore, unwieldy descriptors like these should be avoided.

Also note that we manually divide the test node functions into 3 groups, not only because of their intra-group connection/similarity, but more importantly, unlike the third category where a single point is sufficient to complete the calculation, to implement the test node functions in the first two categories a neighborhood is involved. Especially, the first category needs to create structure tensors and perform eigen decomposition which can be time-consuming and therefore should be treated with extra care. Nonetheless, all the test node functions are able to map a high dimensional value to a scalar (a $N \times d$ dimensional matrix – a neighborhood – to a real number). We call this operation "projection".

3.4. Putting Everything Together

Now we have all the building blocks for our work. The creation of tree is similar to that shown in Algorithm 1. Except the nodes are split in a different way on which we will elaborate next. Each tree is added to the forest via Line 3 of Algorithm 2. Specifically, the learning algorithm L is substituted by the binary decision tree here. The highlight of our work is the node splitting algorithm, which we show in Algorithm 3.

Given all the points that arrive at a certain node are denoted by $\mathcal{P}^{N' \times d}$, meaning that there are N' points with d dimensions each. Usually d is 6 because there are three coordinates and three color channels, but some point clouds may have only coordinates, in which case d is 3. It is also possible a point contains other information like intensity etc. Then the d is greater than 6. However, in none of these cases, the raw point cloud demonstrates its inherent geometric properties directly. The parameter number of features n_f will be clear later after we've gone through the whole algorithm.

We check all the features from 1 to n_f one by one. For each feature, we iterate all the points in this node. A neighborhood $\mathcal{N}_{\mathbf{p}_i}$ is retrieved for each point. In practice, we'll search the k nearest neighbors for each point using the k-d tree algorithm and store the indices of the neighboring points in a matrix. Then operation of Line 4 is merely a table look-up routine, whose complexity is $O(1)$. This neighborhood is analogous to a local image patch defined in [12] or in [23].

Algorithm 3 Node splitting

Require: points in this node $\mathcal{P}^{N' \times d}$, number of features n_f

bestDrop, bestPos, bestSize, bestType $\leftarrow 0, 0, 0, 0$
for $n = 1$ **to** n_f **do**
    for $i = 1$ **to** N' **do**
        $\mathcal{N}_{\mathbf{p}_i} \leftarrow$ recoverNeighborhood$(\mathbf{p}_i)$
        $m \leftarrow$ partition$(\mathcal{N}_{\mathbf{p}_i})$
        $b \leftarrow$ rand$\{1, 2, 4\}$
        for $j = 1$ **to** b **do**
            $pos_j \leftarrow$ rand(sizeof(m))
        end for
        $u \leftarrow$ rand(numberOfProjections)
        $\mathbf{r} \leftarrow \mathbf{Pj}_u(\mathcal{N}_j)$
        if sizeof($\mathbf{r}$)=1 **then**
            $r \leftarrow \mathbf{r}_{[1]}$
        else if sizeof($\mathbf{r}$)=2 **then**
            $r \leftarrow \mathbf{r}_{[1]} - \mathbf{r}_{[2]}$
        else
            $r \leftarrow (\mathbf{r}_{[1]} - \mathbf{r}_{[3]}) - (\mathbf{r}_{[2]} - \mathbf{r}_{[4]})$
        end if
        store r in a vector $\mathbf{v}$
    end for
    $\hat{r} \leftarrow$ median($\mathbf{v}$)
    for $i = 1$ **to** N' **do**
        if $\mathbf{v}_i \leq \hat{r}$ **then**
            $\mathbf{p}_i \in \mathcal{P}_L$
        else
            $\mathbf{p}_i \in \mathcal{P}_R$
        end if
    end for
    drop = impurityDrop$(\mathcal{P}, \mathcal{P}_L, \mathcal{P}_R)$
    if drop>bestDrop **then**
        bestDrop, bestPos, bestType $\leftarrow$ drop, pos_j, u
    end if
end for

In the following line (Line 5), we partition the neighborhood into many smaller parts, or sub-voxels. This partition operation should be implemented in such a way that the spatial relations between the neighboring points and the query point $\mathbf{p}_i$ are maintained during both training and test. In other words, a robust spatial ordering of the neighboring points should be established. The details are revealed until the end of this chapter.

Likewise, we will randomly choose 1, 2 or 4 (Line 6) sub-voxel(s) within this neighborhood and denote them $\mathcal{N}_j$. This is similar to the regions defined in [23].

Next, in Line 10 and 11, we randomly pick a projection from those we defined in 3.3.2 and apply it to the sub-voxels. The projection maps a matrix to a scalar. (Because there are possibly more than one sub-voxels, we use a vector representation $\mathbf{r}$ for gathering the result.) The scalar value can either indicates the linearity (or planarity etc.) of a sub-voxel, or the average red (or green etc.) channel value of it, for example. Then we compare the scalars from Line 12 to 18 according to whether the vector $\mathbf{r}$ is a one-value, two-value or four-value vector. The purposes of the comparison are to find the differences of specific characteristics between the sub-voxels. The comparison results are saved temporarily as they are needed to find the median, which we use to set the threshold θ. Recall that in [22], the authors drew a conclusion that using a median as threshold requires insignificant computational effort but can lead to a balanced tree.

We go over all the points in this node again. This time each point is associated with a scalar value stored in $\mathbf{v}$. We let the points whose corresponding $\mathbf{v}_i$ value is less than the median go to the left child node and the others to the right. What follows is to compute the impurity drop using Equation 3.2.2.

From the previous description we know that, for each value of n_f, a randomly selected sub-voxel position(s) and a projection type are linked to it. We call these parameters together a "feature". Then, how many possible features are there in a node? Suppose the neighborhood is partitioned into m sub-voxels, there are three kinds of sub-neighborhoods (1/2/4) and p projection types, the total number of features is

$$n_{\text{total}} = \left(\binom{m}{1} + \binom{m}{2} + \binom{m}{4} \right) \times \binom{p}{1}$$

With that being shown, a partition of neighborhood into 27 sub-voxels with 27 candidate projection functions can lead to about 480,000 features in a node! That's really a big number. We only use a very small fraction of it, namely, the number of features n_f.

Of all the n_f features that we consider, the best one in the sense that it makes the

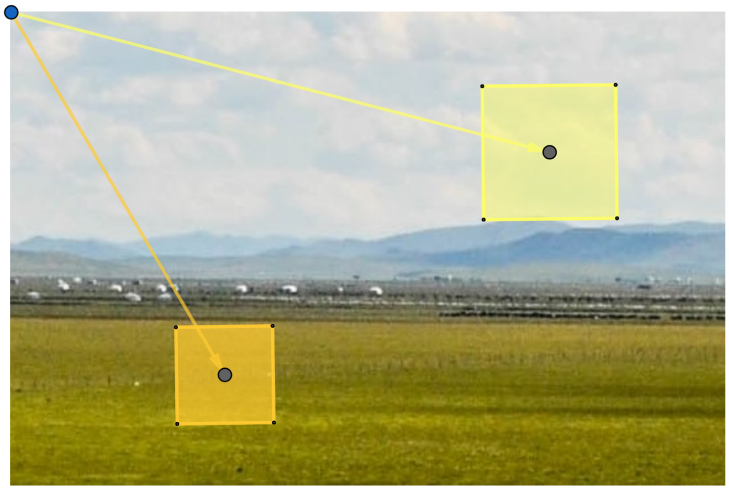


Figure 3.4.: 2D images have well defined structure. A randomly selected pixel, representing the position of a local region, can always be located uniquely with respect to the reference point of the image (usually the upper left corner). The size of a local region can also be randomly determined.

impurity drop the most significantly is chosen and stored in this node.

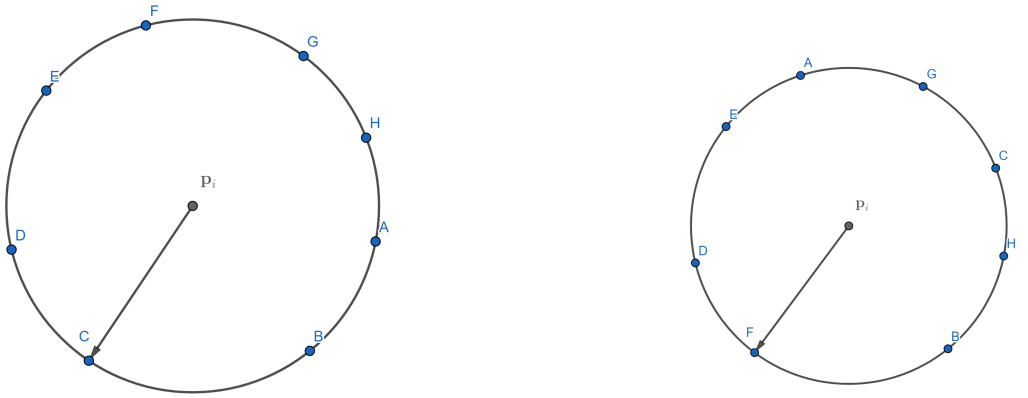
3.4.1. Further Implementation Details

In Line 5 of Algorithm 3, we strive for a stable way to partition the neighborhood. By saying "stable", it means, when a certain sub-voxel is selected during training, a similar sub-voxel should also be selected during test, if the structure of the test neighborhood resembles that of a training neighborhood. The central point to this question is to establish a robust scheme of selecting the positions of sub-voxels.

In a 2D image, the local regions' positions can be randomly selected, so can the sizes. This is because 2D images have a well-defined structure and are not permutation invariant. See Figure 3.4.

But for 3D point cloud, the situation is more complicated, as a point cloud has no clear structure and it's invariant to the ordering of the points. So we can't simply choose some points within a neighborhood and build sub-voxels around them mimicking the techniques used in the 2D image situation.

To illustrate the difference, we give a concrete example as follows. Suppose a circular



(a) In training phase (b) In test phase

Figure 3.5.: Point with equidistant neighboring points

structure is seen during training, as shown in the left hand side of Figure 3.5, the neighboring points all have the same distance to the query point $\mathbf{p}_i$. The random forest has chosen point C to cast projections upon during training phase. Since the neighboring points are obtained by searching for the nearest neighbors of $\mathbf{p}_i$ based only on distances, points with the same distance to the query point may come in any order in another search.

During test, a similar circular structure, as shown in the right hand side of Figure 3.5, is to be classified. Anyhow, this time, its neighbors are ordered differently than the similar circular structure seen during training. What we need to do in fact is to cast the projection type – which is determined during training phase – on point F instead of point C, to ensure the properties we calculated now indeed match to the properties revealed by the random forest during training phase. Suppose that we have adopted a simple method based on the ordering of the points, so that we have an ordered point list, we would have ended up with the wrong point C by simply choosing the third point from the point list. Point C now is to the right above of $\mathbf{p}_i$, other than the correct point F, which is to the left below of $\mathbf{p}_i$.

To overcome this issue, the way of random selection of point as sub-voxel position must be abandoned. Instead, we define a minimal axis aligned bounding box of the neighborhood. The bounding box is partitioned into $3 \times 3 \times 3$ parts, representing a sub-voxel to the above-left-front, above-left-middle, above-left-behind, ... below-right-behind of the query point.

In practice, it is easy to find the bounding box of the neighborhood that is relative to

(a) Points expressed in global frame

(b) Points transformed in local frame

Figure 3.6.: The same sample point cloud expressed in different coordinate frames with their bounding boxes shown. Note that in the left hand side of the figure, the bounding box is aligned to the global frame. After translating and rotating the point cloud, the bounding box is aligned to the local frame.

the global coordinate frame. However, this bounding box is obviously not invariant to rotation of the point cloud. To retain rotation invariance, the bounding box should be expressed in a local frame. This local frame naturally arises from the eigenvectors of the covariance matrix of query point's neighborhood. This is because the covariance matrix of a neighborhood is symmetric and positive semi-definite, the eigenvalues are real numbers $\lambda_s \in \mathbb{R}$ ($s \in \{1,2,3\}$). The eigenvectors $\mathbf{e}_s (s \in \{1,2,3\}$ therefore always form an orthogonal coordinate system satisfying the right-hand rule (major eigenvector represents x-axis and minor eigenvector z-axis). To express the considered neighborhood in the local frame $\mathcal{F}_u$, we regard the points as being translated and rotated from the global frame $\mathcal{F}_g$ by a vector $\mathbf{t}$ and a rotation matrix $\mathbf{R}$, as shown in Figure 3.6, hence we have:

$$\mathbf{p}_g = \mathbf{t}_{gu} + \mathbf{R}_{gu} \cdot \mathbf{p}_u \tag{3.4.1}$$

In the above equation, $\mathbf{p}_g$ is the coordinates of query point $\mathbf{p}$ expressed in the global frame, $\mathbf{t}_{gu}$ is the translation between the origins from the global frame to the local frame. The columns of $\mathbf{R}_{gu}$ are the eigenvectors corresponding to the descending ordered eigenvalues which reflect the rotation matrix from the global frame to the local frame. The point expressed in local frame is denoted as $\mathbf{p}_u$. From Equation 3.4.1, we can derive the point coordinates expressed in the local frame as:

$$\mathbf{p}_u = \mathbf{R}_{gu}^{-1} \cdot (\mathbf{p}_g - \mathbf{t}_{gu}) \tag{3.4.2}$$

In Equation 3.4.2, $\mathbf{R}_{gu}^{-1}$ stands for the rotation from the local frame to the global frame.

Recall that $\mathbf{R}_{gu}$ is an orthogonal matrix, which means

$$\mathbf{R}_{gu}^{-1} = \mathbf{R}_{gu}^{T} \tag{3.4.3}$$

The more computationally demanding matrix inverse operation is thus substituted by the matrix transpose operation.

But the local frame derived from above is not unique. This is due to the fact that when we eigen decompose the covariance matrix C, the sign of the eigenvectors cannot be unambiguously determined. The ambiguity will lead to the inconsistent orientation of the deduced eigenvectors. By the way, the minor eigenvector is also called the normal since it is orthogonal to the surface spanned by the major and middle eigenvectors. Figures in the left column of Figure 3.7 demonstrates the arbitrary orientation effects on two parts taken from a larger point cloud representing a kitchen environment. Since the kitchen dataset was aquired from a single viewpoint, the normals shown in the figure should be present only on half of the normal sphere. But due to the orientation inconsistency, they are spread across the entire sphere [40].

To resolve this problem, we manually force the normals $\mathbf{e}_{3i}$ to orient towards a unified viewpoint v_p, so that the equation

$$\mathbf{e}_{3i} \cdot (\mathrm{v}_p - \mathbf{p}_i) > 0 \tag{3.4.4}$$

satisfies. The right column of Figure3.7 presents the results after all normals have been consistently oriented towards the viewpoint. In implementation, the viewpoint is set as the origin of the global frame.

Once the points are expressed in the local frame, a bounding box, that is now aligned to the local frame, can be easily built. This bounding box is equally divided into 27 parts, that is, the length, width and height of the bounding box are equally cut into three segments. Following the partitioning scheme, the query point resides in the central sub-voxel. The rest of the sub-voxels are by definition relative to it in the above-left-front corner of the bounding box until the below-right-behind corner.

The neighborhood partitioned this way will inevitably contain many empty sub-voxels, as in real dataset, the points are spread on the surface of the objects, which implies that the point cloud is actually hollow (see Figure 3.8). But in theory, this should not be a problem because once an empty sub-voxel is selected, the random forest cannot cast projections on it, and it will therefore make no contribution to the impurity drop. By working principal of the random forest, such a sub-voxel will most likely be abandoned. Compared to the 2D image case, where both the position and size of a region within a local image patch are randomly sampled, which leads to millions of combinations, the number of possible features in a tree node in the 3D

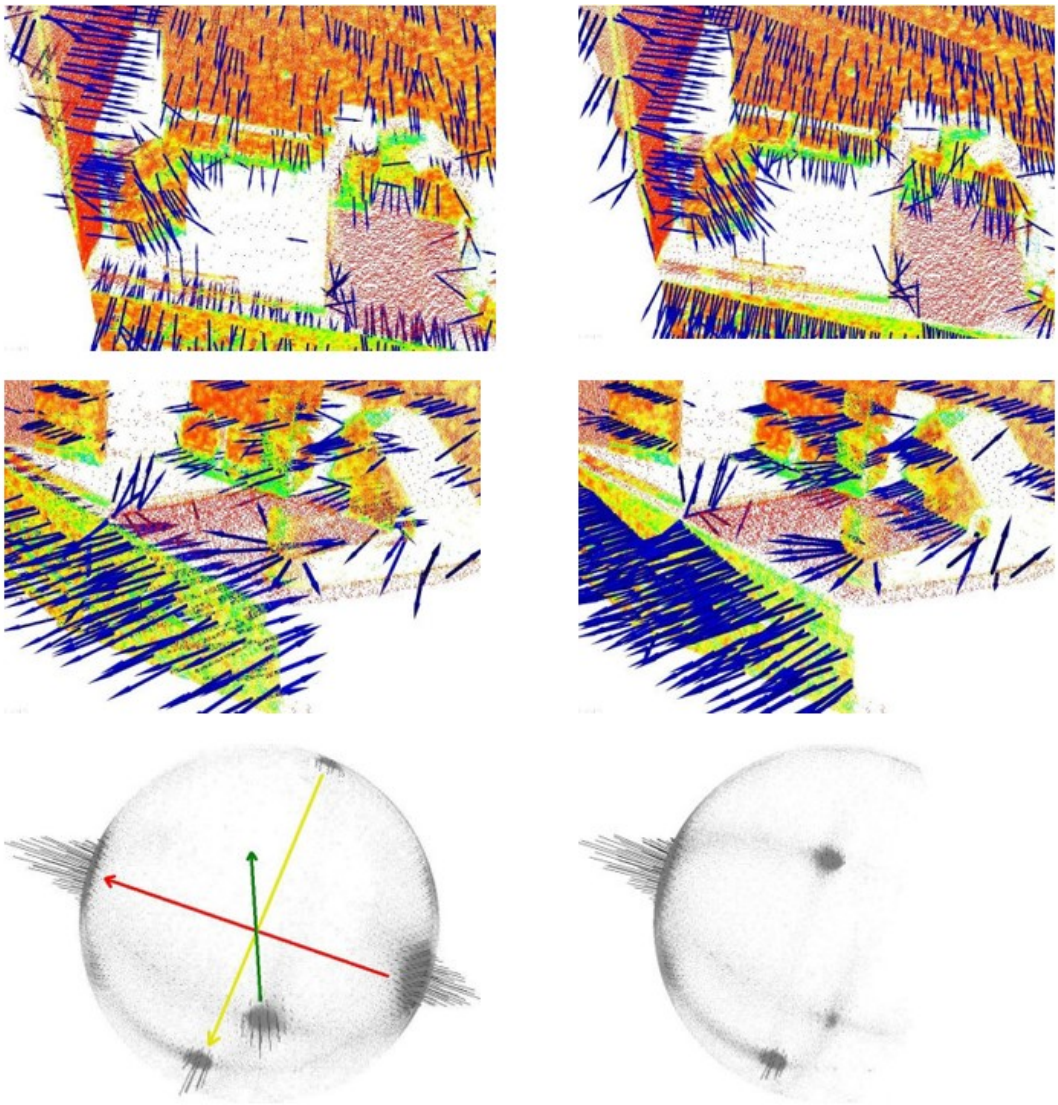


Figure 3.7.: The top and middle figures in the left column show the inconsistently oriented surface normals (i.e. the third eigen vectors) obtained by eigen decomposition of covariance matrix of point neighborhoods. The bottom figure in the left column shows the normal sphere, also known as Extended Gaussian Image (EGI), which depicts the aggregated normals within the presenting point cloud. The right column shows the normals being flipped to a predefined viewpoint and the corresponding normal sphere. [40].

point cloud case, is much less and is of the magnitude of hundreds of thousands as is shown earlier. To make the non-empty sub-voxel be selected with higher chance, the number of features in each node should be set larger than the recommended value.

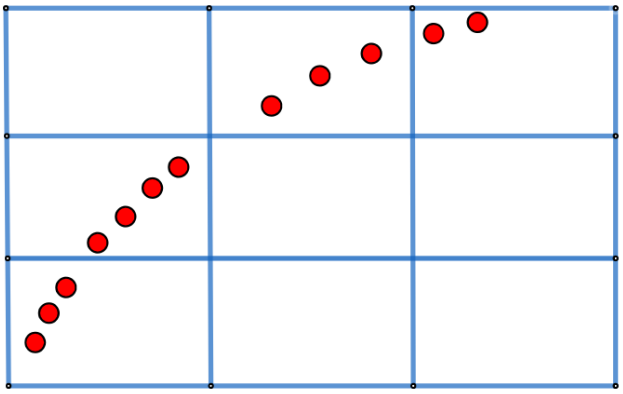


Figure 3.8.: Partition the bounding box of a neighborhood into sub-voxels. Some of the sub-voxels are empty because the neighborhood is hollow. Here, to avoid complexity in display, a 3D neighborhood is projected onto a 2D plane. The red points, located on a curve, represent a surface of an object in 3D world.

4. Experiments

In this chapter, results of various experiments are reported. Experimental setup, used datasets and evaluation metrics are described first. Then we present the results on a synthesized point cloud to show the correctness of the novel method, followed by demonstrating the results obtained when our method is applied to real world datasets.

The majority of the code is implemented in C++. Some data extraction and analysis scripts are written in Python. All the experiments are completed on a consumer laptop with a modest Intel i5-7200U @ 2.5 GHz CPU and 8 GB RAM.

4.1. Metrics

As suggested by authors of [20], "Intersection over Union" and "Overall Accuracy" are chosen as the evaluation metrics. Let $C^{N\times N}$ be the "Confusion Matrix" of a test result, where N is the number of classes and $c_{ij} \in C$ is the number of samples with ground truth label i being classified into label j. Obviously, the confusion matrix provides the information of how absolute errors are distributed over difference classes. A better result is always reflected by larger main diagonal values. However, the confusion matrix is less intuitive when compared to the multi-class IoU for each class i which is defined as:

$$\mathrm{IoU}_i = \frac{c_{ii}}{c_{ii} + \sum_{j\neq i} c_{ij} + \sum_{k\neq i} c_{ki}} \tag{4.1.1}$$

Where c_{ii}, c_{ij} and c_{ki} are the true positives (TP), false negatives (FN) and false positives (FP) for each class. Equation 4.1.1 is a measure of a test's class-wise accuracy. It is also to find in the literature that some authors just used the recall and precision as their primary metrics (e.g. in [50], [20]), which are defined as:

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

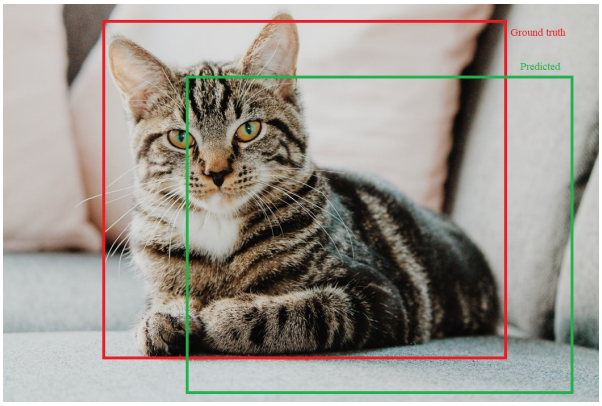


Figure 4.1.: Intersection over Union is a measure of similarity widely used in evaluating experiment results in objection detection and segmentation in computer vision. It is defined as the area of overlap over the area of union. Ideally, the predicted bounding box (green) should be identical to the ground truth bounding box (red), in which case, the IoU is 1.0.

and

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Precision and recall can be converted to intersection over union by:

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} = \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall} - \text{Precision} \times \text{Recall}} \tag{4.1.2}$$

The recall and precision metrics are converted into intersection over union when necessary to make the results comparable.

From the above equations, it is easy to see that intersection over union takes both the precision and the recall into account. It also compensates for different class frequencies as well. Intuitively, this metric is illustrated in Figure 4.1.

To gain a general understanding of the IoU over all classes, we use a simple average of the IoUs as

$$\overline{\text{IoU}} = \frac{1}{N} \sum_{i=1}^{N} \text{IoU}_i \tag{4.1.3}$$

Since the (averaged) IoU is the most strict measure, it is hence used as the **primary evaluation metric** in this work.

The overall accuracy is used as an auxiliary measure:

$$\mathrm{OA} = \frac{\sum_{i=1}^{N} c_{ii}}{\sum_{j=1}^{N} \sum_{k=1}^{N} c_{jk}} \tag{4.1.4}$$

It quantifies the number of correctly predicted points in proportion to the number of total points without considering any class-related information. Unlike the IoUs, overall accuracy can be misleading for imbalanced datasets. This metric is kept to make the results in this work consistent with results of other works.

4.2. Datasets

The datasets used in this work are a synthesized dataset and two real world datasets. The synthesized dataset is introduced to verify the correctness of the novel method proposed in this work. The other two datasets are used to show the effectiveness and potential of the method.

The synthesized dataset is rather primitive. It consists only two classes, and each class contains exactly one geometric object, namely a plane and a sphere. The training set contains 80000 points in total with equal number of points in each class. The test set is much larger and has 260000 points, 100000 of which belongs to the sphere class and the rest goes to the plane class. All points in both training and test sets have RGB colors with sphere in red and plane in green. To make the dataset more realistic, Gaussian noise is added to the appearance of two geometric objects. Thus the surface is not absolutely smooth and the color is not purely red or green. The algorithm is supposed to separate the two classes successfully as the two classes are very distinctive.

One of the real world datasets is called Oakland 3D point cloud dataset [33]. It is relatively outdated and small, but is reasonable to be used as a benchmark to test the effectiveness of our method because the dataset has been evaluated in different papers, for example in [50]. The data was acquired with a mobile platform equipped with side looking SICK LMS laser scanners and used in push-broom mode. It was split into three sets, namely a training set, a validation set and a test set. Each point was assigned one of the five class labels: wire, pole/trunk, facade, ground and vegetation. Training set contains 36932 points, validation set 91579 points and test set 1.3 million points. The class prior distribution is highly imbalanced as we can see from the histogram (Figure 4.2) enclosed in the dataset (note that the y-axis is in

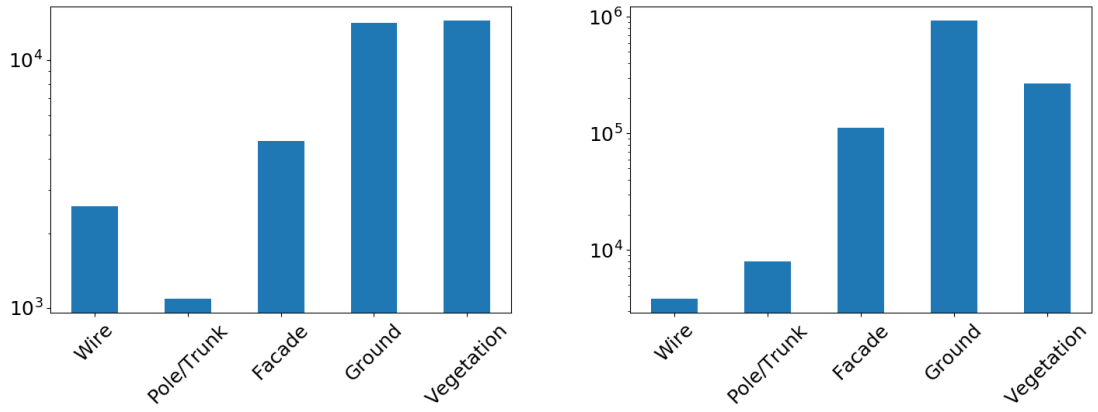


Figure 4.2.: Training set (left) and test set (right) label distribution of Oakland 3D point cloud dataset. The y-axis is logarithmic-scaled.

logarithmic scale). Originally this point cloud has no color information (see Figure 4.3b).

The other dataset from real world is a more up-to-date one – the Paris-rue-Madame database [1] [42]. This point cloud dataset was acquired in 2013 in Paris. It shows a section of a street scene with an approximate length of 160 meters. When compared to the Oakland dataset, Paris-rue-Madame dataset is much larger in terms of the number of points and consists of exactly 20 million points, making it much denser than the Oakland dataset. Moreover, there are more categories in the Paris-rue-Madame dataset which are even more imbalancedly distributed (See Appendix A.3). The dataset is divided into two equal parts, with each of 10 million points. One is called GT_Madame1_2, the other is called GT_Madame1_3. To acquire the dataset, a Velodyne HDL32 was used. To annotate, human assistance was involved. Finally, 642 objects belonging to 26 classes are found. This dataset has no color information.

The detailed statistics regarding the respective datasets used in the work, including the class names, is found in the Appendix A.

[1] MINES ParisTech created this special set of 3D mobile laser scanning (MLS) data for the purpose of detection-segmentation-classification research activities, but does not endorse the way they are used in this project or the conclusions put forward.

4.3. Testing of Correctness

In order to verify whether the novel method is indeed correct, a first experiment is conducted on a synthesized dataset. The dataset is carefully designed such that the training set and test set are visually similar in terms of shape and color, but they are actually different in terms of the numbers that represent the coordinates and colors. There is no single point in the test set that is seen in the training set. Also the color is slightly different. To make the dataset resemble a real world dataset as if it is acquired by a LiDAR sensor, the points reside only on the surface of a cubic (a plain) and a sphere (see Figure 4.3a). Furthermore, the number of points in both sets are not equal to mimic the general case in real world.

Usually the number of points in a point cloud is very large, on the one hand, it is impossible to train every point in the training set otherwise the training time will become unbearably long once the model grows more complicated. On the other hand, it is generally the case that the classes of the labels are heavily imbalanced [2]. To account for these problems, a subset of the training set is sampled out. The subset is small enough to have the required training time fall in a reasonable range. The labels in that subset are also made balanced by downsampling equal number of points from each class.

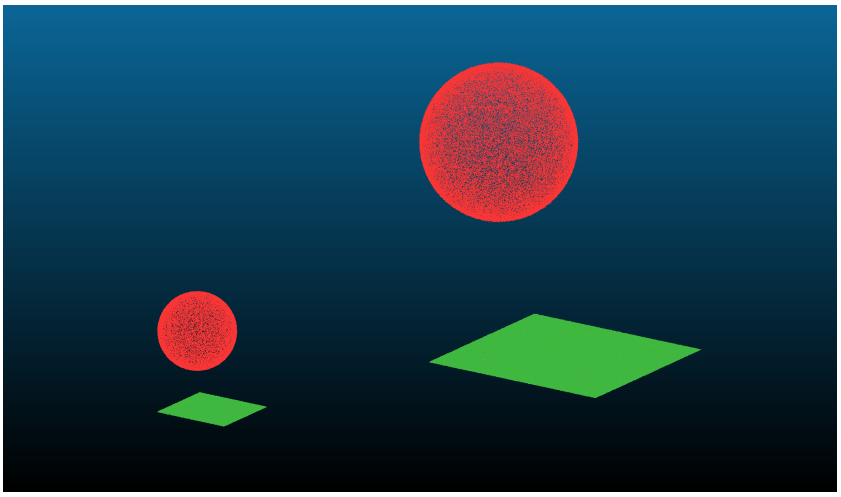
As a starting point and for the purpose of simplicity, a minimal setup with merely 2 trees in the forest is selected and tested on this synthesized dataset. Each tree has a maximal height of 10. This classifier is trained on 10000 sample points out of 80000 points (5000 points per class) and tested on the whole test set containing 260000 points. The confusion matrix of the test result is given in Table 4.1.

Substitute the recall and precision into Equation 4.1.2 and further into Equation 4.1.3, we have the IoU for the plane class is 0.885, the IoU for the sphere class is 0.838 and the average IoU is 0.862. The overall accuracy reaches 0.928, indicating that our algorithm is able to distinguish the two classes.

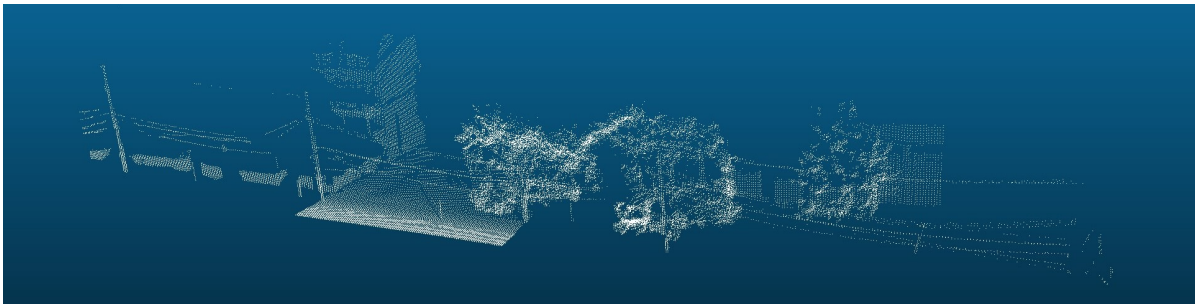
Considering that there are only two trees in the forest, the result is already very impressive. We go one step further by increasing the number of trees to 20 and keep the other parameters unchanged and train the model with the same training set. This time, the classifier achieved a remarkable overall accuracy of 1.000, an average IoU of 0.999, as seen in the right side of Table 4.1.

The test results of the experiments clearly testify the correctness of this novel method,

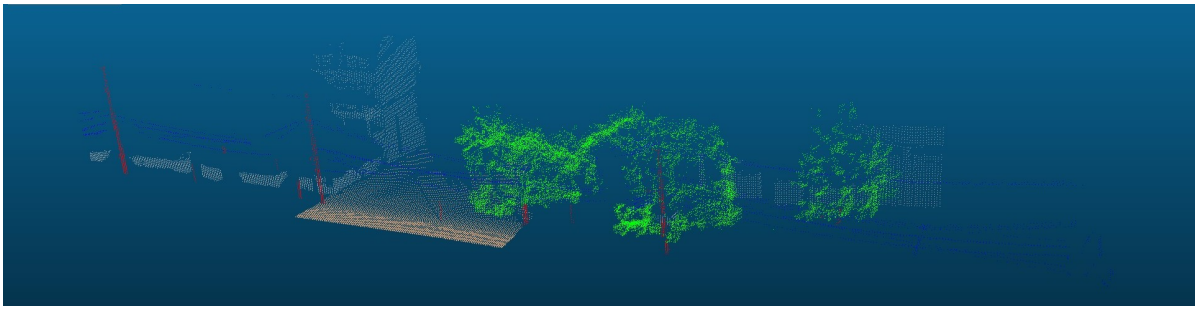
[2] Although the algorithm is designed to be able to work with imbalanced datasets, however, to avoid potential side effects introduced by imbalanced data, a balanced dataset is used in the experiments if otherwise stated.



(a) A synthetic dataset consists of a sphere and a plane. In the left half of the figure is the training set, in the right the test set. Both training set and test set share the same reference frame whose origin is (0, 0, 0). This figure illustrates that no single point in the test set is identical to the points in the training set since they differ by a large margin in both positions and sizes.



(b) Training set of Oakland 3D point cloud (no color information is available)



(c) Ground truth labels of training set: vegetation - green, wire - blue, pole - red, ground - brown, facade - gray

Figure 4.3.: Some datasets used in this work

	Experiment 1		
	plane	**sphere**	**recall**
plane	144372	15628	0.902
sphere	3091	96909	0.969
precision	0.979	0.861	
IoU	0.885	0.838	

Experiment 2		
plane	**sphere**	**recall**
159940	60	0.9996
54	99946	0.9995
0.9997	0.9994	
0.999	0.999	

Table 4.1.: Confusion matrix of test results of synthetic dataset. The ground truth labels are listed on the left column, the predicted labels on the upper row. Tree depths = 10, number of features per node = 30, number of nearest neighbors = 100. Left: number of trees = 2, Right: number of trees = 20

that is, rather than calculating the features associated with each point explicitly, a random forest's feature learning capability is exploited.

4.4. Acceleration by Precomputing Neighborhood Partitions and Eigen Decomposition

Naively the algorithm is designed to work without any data prepossessing involved. Everything can be calculated on-the-fly and only points to be trained together with the associated point cloud is needed. The process is summarized as follows: 1), a subset of training points (after bagging operation) and the point cloud arrive at the root node of a tree in the forest; 2), the points propagate down the tree following Algorithm 3. At each node, the points will do nothing but retrieve their neighboring points and each neighborhood is partitioned into 27 sub-voxels following the description presented in Section 3.4.1 of Chapter 3; 3), the node cast projections on the sub-voxels and compare the results; 4), points are separated into two child nodes based on the results; 5), repeat steps 2 to 4 until the stopping criteria fulfilled.

Apparently, there is a lot of repeated work being done. First, the neighborhood of a point will stayed unchanged no matter at which node arrives the point. Thus, it is unnecessary to retrieve the neighborhood and partition it again and again. Second, although not all projections require the results of eigen decomposition of sub-voxels, it is expected that a large number of such eigen decomposition is being performed at each node since there are much more covariance matrix related features (namely the 3D properties) than the simple radiometric features (Cf. Section 3.3) in the "feature factory". Again, the result of eigen decomposing the covariance matrix of the sub-voxels, that is, the eigen values and eigen vectors of each sub-voxel at each node

	Experiment 1 (number of trees = 2)				Experiment 2 (number of trees = 20)			
	training		prediction		training		prediction	
	w/o pre.	w. pre.	w/o pre.	w. pre.	w/o pre.	w. pre.	w/o pre.	w. pre
time consumption (s)	1493	2.36+83	2191	54+114	15622	2.25+890	18010	54+615
memory consumption (MB)	18.5	41.7	211	964	25.2	50.6	218	966

Table 4.2.: Resource consumption for different experiment setups with/without precomputing sub-voxels and eigen decomposition.

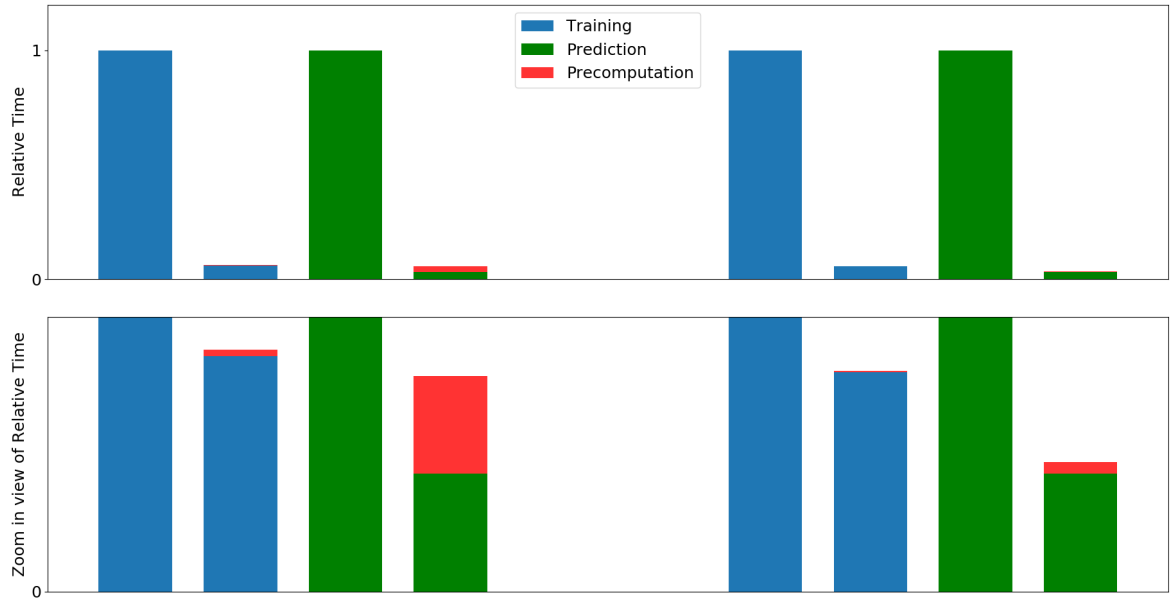
remain unchanged but will be computed repeatedly in the naive formulation of the algorithm.

To avoid the afore-discussed repeated work and save time in training and test, minimal precomputing work is introduced. Different from precomputing the feature vectors for each query point as other works usually did, the precomputing does only the following two things: 1), find the neighborhood of every training point and partition the neighborhood into 27 sub-voxels, then store the results; 2), for each sub-voxel, compute the eigen values and eigen vectors by eigen decomposing the covariance matrix of the sub-voxel, then store the results. Notice: no features is explicitly computed yet! In case a sub-voxel has less than three points, no eigen values or eigen vectors can be calculated, a placeholder -1 is stored.

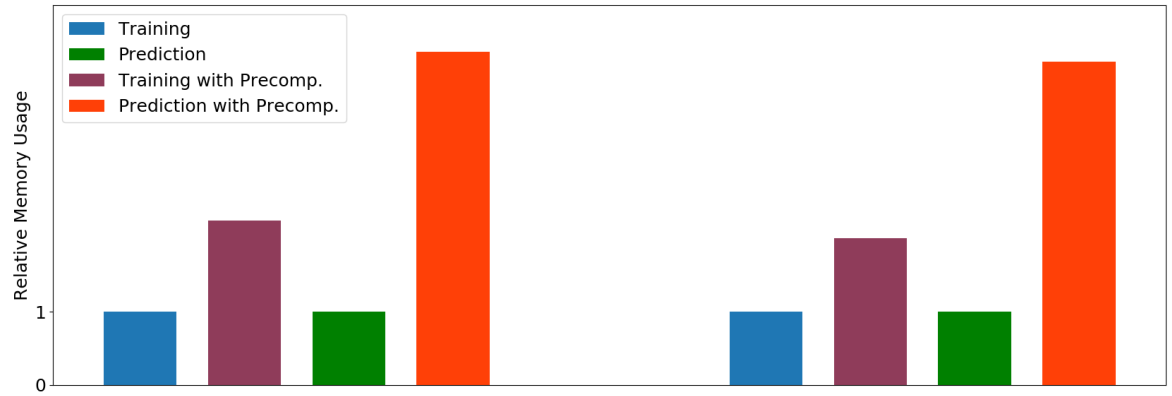
Although extra space is used (for storing the sub-voxels, eigen values and eigen vectors) , the enhancement in time is not negligible, see Figure 4.4. This is verified by the following experiments.

Two experiment setups with difference only in the number of trees are applied to the synthesized dataset. In Setup 1, there are just 2 trees while in Setup 2, the number of trees is increased to 20. Both setups share the other parameter setting as: tree height is 10, number of features per node is 30 and neighborhood size is 100. The time and memory usage during training and test are recorded and summarized in Table 4.2.

Clearly, by precomputing and storing the variables that are going to be accessed frequently in the algorithm will considerably save the time for both training and prediction. Precomputing takes about 2.36s in training phase or 54s in prediction phase in Setup 1 while the time for training/prediction plummets from 1493s/2191s to 83s/114s. Especially, when there are more trees (or equivalently more nodes) in the forest, the precomputing time becomes far less significant than the actual training/prediction time, for example, precomputing takes 2.25s and actual training takes 890s during training phase in Setup 2. This is because the required time for



(a) Running time is reduced greatly by precomputing frequently used sub-voxels and eigen decomposition. Above: left part showing results of Setup 1, right part Setup 2. Below: a zoom-in view to reveal that the precomputing time becomes less significant with the growing number of nodes in the forest when compared to the actual training/prediction time. The training/prediction time without precomputation is normalized to 1 and the time with precomputation is scaled accordingly.



(b) Memory usage is increased after precomputing. The impact is more obvious during prediction because the number of points to be classified is used general very huge. Memory usage during training/prediction without precomputation is normalized to 1 and the memory with precomputation is scaled accordingly.

Figure 4.4.: Comparison of resource consumption (training/test time and memory usage) when whether precomputing operation is performed.

precomputing is only influenced by the number of query points. Once the number of query points is determined, the time for precomputation becomes a constant which will not increase along with the growing number of nodes. For either setup in the experiment, the time for precomputation varies only within variance. For a vivid illustration of the enhancement, see Figure 4.4a.

Nonetheless, it should be noticed that the precomputation has a greater impact during the test phase. This is due to the fact that, only several thousands points' sub-voxels and eigen values are required during training while hundreds of thousands points' sub-voxels and eigen values are needed during test. The random forest itself will consume some memory, but this amount of memory is relatively small or even incomparable to the memory dedicated to store the intermediate variables. In either setup of the experiments, a tree takes up on average about 0.45 MB memory, which means even 20 trees use less than 10 MB. The most of the memory is used for storing the point cloud and its relevant values. This fact provides an idea for possible optimization in the future.

4.5. Testing of Parameters

There are a lot of meta parameters to adjust, among which the most important ones include the size of the neighborhood, the number of trees, the height of each tree, the number of features at each node. The impact of these parameters on the final results are two folds. On the one hand, they will affect the overall classification quality.

Among all the adjustable parameters, take the size of a neighborhood for an instance. The size of a neighborhood plays an important role in the classical 3D semantic segmentation methods. Rusu stressed this problem in [39] by showing that a small neighborhood was often desired to capture the details of a small object but may be too computationally costly. Furthermore, fine details were not always necessary, for example, when larger objects such as facades or grounds were to be classified. There is no universally applicable method to determine the best size of a neighborhood as it is data dependent. Weinmman et. al. [49] presented a fully automated framework that can determine the optimal size of each neighborhood individually for extracting the features. However, the trial and error process made the training extremely tedious. Apart from the size of a neighborhood that can affect the test results, also the number of trees and number of features per node will have an impact on the final outcome. Intuitively, more trees and more tests will enhance the decorrelation among the trees and in turn boost the classification quality.

On the other hand, a bigger forest with taller trees will inevitably lead to the increase

of training and prediction time. The trade-off between classification quality and running time is what we should be looking for.

To explore the influences of the parameters on the test results, a series of experiments are carried out. The next experiments in this section are based on the Oakland 3D point cloud dataset since it is a reasonable dataset from which to draw some conclusions, in the sense that, first, it is a real world dataset; second, the number of points is relatively small which will keep the running time of the experiments tractable; third, a baseline can be found in the literature [49], with which the test result of the proposed novel algorithm can be compared.

Specifically, 1000 points from each class of the original training set are randomly picked out to form the real training set that goes into the to-be-trained classifier. Instead of using the whole point cloud, only a subset is used because it was pointed out earlier, that too many points in the training phase will potentially make the training time grow excessively long. The obtained classifier is evaluated on the validation set offered by the dataset provider. To keep the test results comparable with each other, the same training set is used across all the experiments, that is, the training set is downsampled exactly once. Finally, the well trained classifier is applied on the test set (contains about 1.3 million points) to evaluate the effectiveness of the algorithm.

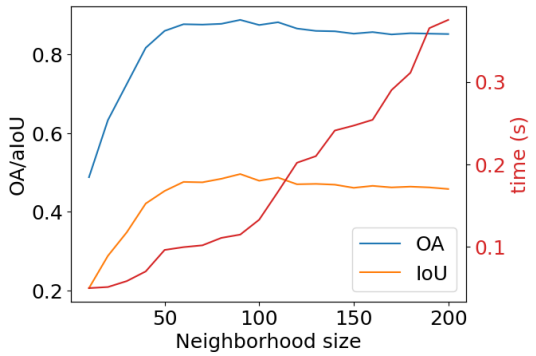
4.5.1. Varying Neighborhood Sizes

As the size of the neighborhood is important to the semantic segmentation of a point cloud, these experiments dedicate to the problem.

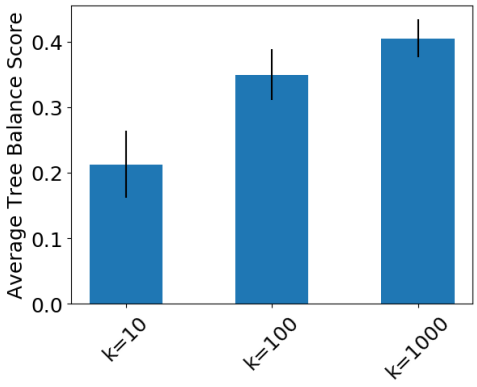
The neighborhood is searched by the k-d tree algorithm presented in Chapter 3. As is shown there, the algorithm is very efficient in low dimensional search space. In this experiment, we report the consumed search time for various cases as well as the test results in Figure 4.5a.

The number of trees is fixed to 20 with each having a maximal height of 10. This setup corresponds to an average number of points at a leaf node about 10 given the training set size. Number of features per node is set naively 20 following the suggestion of [6]. We start with 10 points in a neighborhood and all the way up to 200 points with an incremental step of 10 points.

The intersection over union and overall accuracy have a similar tendency. When the neighborhood size is small, both of then ranked a low score, starting off at around 0.2 for the IoU or lower than 0.5 for the OA. Both metrics indicate that the prediction



(a) Test results of varying neighborhood sizes



(b) Tree balance scores of varying neighborhood sizes

Figure 4.5.: Neighborhood sizes have influences on the intersection over union/overall accuracy and on the balance of the trees

is no better than a random guess. With an increasing number of points is included in the neighborhood, the IoU and the OA climb up till the turning point, where the neighborhood size is 90. At this point, the two metrics reach the best scores in this experiment, with the former being 0.496 and the latter 0.888. If the neighborhood size is further enlarged, the performance deteriorates. When the number of points in a neighborhood reaches 200, the IoU drops from the best score 0.496 to 0.458. So does the OA (from 0.888 to 0.852). A neighborhood with 1000 points is also tested. It turns out that, with such a big neighborhood, the OA decreases to 0.827 (or 0.383 for the IoU). It is rational to infer that, with an even larger neighborhood size, the performance will become even worse. A reasonable explanation involves a looking at the structure of the tree.

To interpret the structure of a tree, one can make use of the number of non-leaf nodes N_N and the number of leaf nodes N_L. Given a fixed tree height, a better result is obtained on condition that both types of nodes are as many as possible. The more non-leaf nodes will bring in more tests on the dataset. While the more leaf nodes will result in the higher resolution of the splitting. The ratio between the actual number of nodes in a tree and theoretically possible number of nodes in that tree, is defined as the the balance of the tree (B) [21]:

$$B = \frac{N_N}{2^{(H+1)} - 1} \tag{4.5.1}$$

This measure becomes 1 when the tree is fully balanced. When the tree is extremely

unbalanced, it approaches 0. A very unbalanced tree is grown when the split is highly unequal. In that case, a very small number of data points is shifted to one of the child nodes. When the number is small enough that satisfies one of the terminating criteria, an early stop of splitting of this child node occurs.

By inspecting the balance of trees (Figure 4.5b), one can conjecture that, when the neighborhood is small, many of the sub-voxels in this neighborhood is empty. As a result, many of the node tests will fail. Since the points of failed tests are shifted by default to one of the nodes, the tree grown in this scenario is unbalanced, hence a lower balance score. When more points are incorporated in the neighborhood, the number of empty sub-voxels decreases. This means more meaningful tests can be made and the associated points can be shifted either to the left or to the right child node. In the ideal case, suppose all the sub-voxels of every point is non-empty, all the points can propagate downstream the tree based on their inherent properties. For example, if the linearity is tested on two of the sub-voxels of every point in a certain node, the points whose test results are less than the median value (half of the data points) are propagated to the left child and those greater than the median value to the right. Under this ideal assumption, a balanced split is achieved and a higher balance score is expected. In short, a higher balance score indicates a more balanced tree, which implies that there are less empty voxels existing in the neighborhood and more meaningful tests being carried out.

However, on the other hand, when the neighborhood size is too large, it will include much unrelated information in it. For instance, assume that a point on the facade which is in the vicinity of a tree, is going to be classified. If the neighborhood is set too large, it may find points that belong to the tree as the neighboring points of the facade. As a consequence, though there exists less empty sub-voxels and more node tests are likely to be carried out, the node tests become confusing to the random forest. This behavior will worsen the final performance.

This experiment also manifests the efficiency of the nearest neighbor searching algorithm (Figure 4.5a). The searching time grows almost linearly with the neighborhood size. Even 5000 points look for their 100 nearest neighbors in a 37000-point point cloud takes a fraction of a second (0.133s). During test, although the query points amount to about 91000 points, the searching time increases to less than 2 seconds (1.748s) [3]. Since it's highly efficient, the time spent in neighborhood retrieving plays an unimportant role in the overall performance of the algorithm. In the following experiments, the time with regard to neighborhood building will not be given and

[3]The searching time here refers to the tree building plus the tree querying time. The number shows here only for the purpose of a rough comparison. Actual searching time for the same setup may vary.

compared with. Detailed consumed time in different setups (fixed neighborhood size with varying number of query points, different neighborhood sizes with fixed number of query points or varying neighborhood sizes with varying number of query points etc.) is included in the appendices (Table B.1 and Table B.2).

4.5.2. Varying Number of Features per Node

As explained earlier, the number of features per node affects the training efficiency and the decorrelation of the trees. If all the potential features are used, no randomness is ejected in the forest and thus no decorrelation happens between trees (no bagging procedure is considered here). In addition, the possible number of features in a node is usually very huge, if all of them are used, the computation time will become intractable. To induce a sufficient decorrelation among the trees and make the training/test computationally tractable, only a small fraction of the total number of features is used in each node.

In the standard formulation, the value for the number of features per node is recommended as $\log_2 d$ (d is the total number of possible features). Nonetheless, the number of features differs greatly in the literature focusing on random forest-based methods for compute vision tasks. It is as small as 500 seen in [8] and goes all the way up to 20000 as seen in [14] for example. The selection of this parameter seems arbitrary and less motivated. In Chapter 3, a rough calculation of d reveals the possible features are as many as 480000. This number may seem at first glance already very big, but it is merely a fraction of the number of the possible features available in 2D images tasks. In a typical 2D image segmentation task, this number can amount to millions. That's why the number of selected features can be as large as 20000.

As a starting point, the initial value set for the number of features is 20. The neighborhood size in this experiment uses the optimal size found in the last section, namely 90. The number of trees and maximal tree height are fixed to 20 and 10 respectively. Figure 4.6 shows the impact of number of feature per node on the classification performance.

The average intersection over union of the experiments starts from 0.496. It gradually increases to 0.532 with the number of features per node increased from 20 to 55. Then the aIoU plateaus regardless of the growing number of features per node. The overall accuracy shows a similar pattern. Detailed results are found in Table B.3 and Table B.4 in the appendices.

We argue the result as follows: although smaller number of feature per node corresponds to a larger randomness and lower correlation among trees, which is supposed

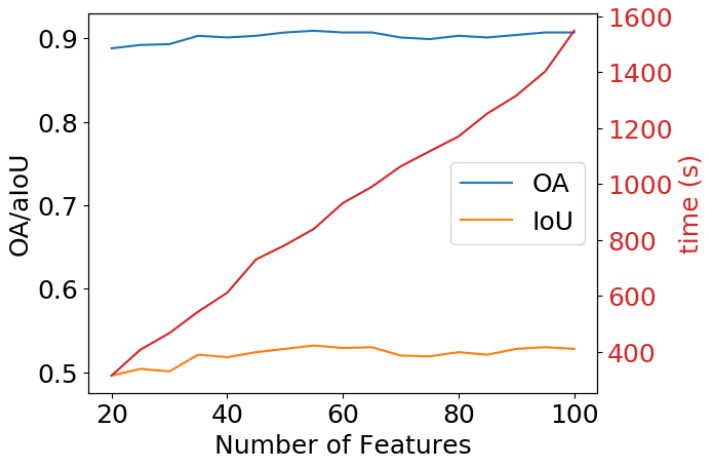


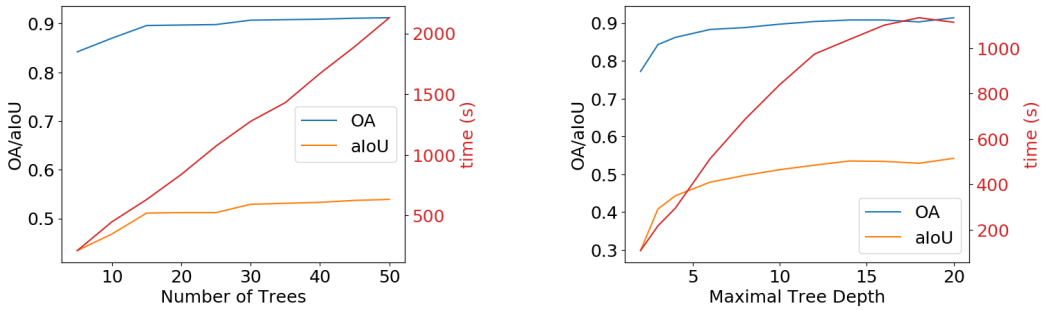
Figure 4.6.: The influences of varying number of features per node on the overall accuracy/intersection over union as well as the training time.

to boost the classification performance. However, when the number of features is too small, specifically in our case, where not all selected features can make a successful test (due to empty sub-voxels), increase the number of feature per node will increase the probability of successful tests and in turn will influence the tree structure (Cf. discussion in Subsection 4.5.1) . But when there are too many features per node, the correlation among trees degrades. The information brought by the extra tests becomes redundant. It should be pointed out that, the best number of features per node is data dependent and also depends on considerations of efficiency and accuracy. As is shown in Figure 4.6, the efficiency should be of priority when determining the actual number of features per node since the training time escalates monotonically with it while this value has less impact on the accuracy in this work.

4.5.3. Varying Number of Trees and Maximal Tree Height

The number of trees and the maximal tree height are regarded as the most important parameters of a random forest. They have a direct impact on the final classification quality. In this section, experiment results with different number of trees and tree height are reported.

If the optimal combination of these two parameter is desired, a grid search should be applied. This process can be extremely time consuming. Assume the number of trees is in the range of [2, 50] and the maximal height of each tree is in [2, 50], each with an incremental step of 1, a total of nearly 2500 searches are expected. This will of course best reveal the interrelationship between the two parameters, but not



(a) Test results of varying number of trees (b) of varying maximal tree depth

Figure 4.7.: Both the number of trees in the forest and the maximal depth of each tree have impacts on the classification quality. In general, the bigger the forest and the higher each tree, the better the result. But the trend in time increase differs in the experiment setups. Only the time used for training is shown in the figures.

very necessary, since some conclusions are already drawn in relevant researches, for example: [21], [8] or [52] etc. The conclusions are universally applicable. Hence, the search for the optimal parameters will be saved until the final experiment of this series of experiments. At this point, the two parameters on the test results are investigated separately.

Generally speaking, the more trees in a random forest, the better the classification performance, the longer the training/prediction process and the higher the memory usage. They all change in the same tendency monotonically. So it is a matter of time and memory that one can afford when choosing the number of trees.

In comparison, the time and memory usage rises exponentially in the tree depth. In addition, a very deep tree is prone to overfitting, which cannot be alleviated by incorporating more trees in the forest [8]. Hence the most appropriate value for the maximal depth of a tree should be selected with more care.

Figure 4.7 shows the test results with different number of trees and maximal tree height. Results in Figure 4.7a are based on the following parameters: maximal tree height = 10, number of nearest neighbors in each neighborhood = 90, number of features per node = 55. Results in Figure 4.7b are obtained by setting number of trees = 20 and the other two parameters are the same as the former experiment setup.

The experiments justify the conclusions mentioned earlier. Although the overall accuracy/intersection over union rises monotonically as the there are more and more trees in the forest, the contribution of even more trees are less prominent when there are already many trees in the forest (compare the increase from 10 trees to

20 trees vs. the increase from 40 trees to 50 trees.). The training time grows up linearly. The maximal tree height affects the results differently. Despite the fact that it boosts the results greatly when the tree grows from shallow to deep, just like the phenomenon observed in the varying number of trees experiment, one can notice the slight fluctuation in OA/aIoU when the tree grows from deep to too deep (aIoU goes from 0.535 to 0.534 then to 0.529 when the tree height grows from 14 to 16 and to 18), which implies that the tree at that height leads to an overfitting.

Other than the OA/aIoU, one should also pay attention to the training time in the varying maximal tree height experiment. At the beginning, training time increases sharply. Then it flattens out gradually afterwards. This is expected because the trees are not fully balanced grown. When the tree is low, the tree balance score can reach 1, meaning it is perfectly balanced. The training time doubles when the tree grows one level higher since the number of nodes are also doubled. Later on, the tree cannot develop into a balanced one. One level higher doesn't bring with twice as many nodes as it would. The training time will not be doubled as a consequence. For example, at tree height 20, the number of nodes of a fully balanced tree should be 4 times that of a tree with height 18. But in the experiment, the tree balance score at height 18 is 0.003 which is equivalent to about 1572 nodes, and the score at height 20 is 0.001 which equals approximately to 2097 noes. The number implies an increase of about 33% instead of 4 times of nodes when the tree grows two level higher.

Detailed report regarding the experiments above is found in Table B.5, Table B.6, Table B.7 and Table B.8.

4.6. Test Results on Oakland 3D Point Cloud Dataset

In this section, we present the performance of the proposed methodology on the small scale Oakland dataset to demonstrate the effectiveness of the proposed method.

The above experiments reflect the influences of the parameters on the classification performance. Generally speaking, the larger number of trees in the forest always leads to a more positive outcome at the cost of increased training/predicting time. Other parameters affect the results in a non-uniform manner, which means, the tendency of the parameter values (increasing or decreasing) doesn't certainly coincide that of the overall accuracy or intersection over union. Indeed, the experiments above used a heuristic way to find the optimal combination. A reasonable trade-off between time and quality is achieved when the number of trees is set at 50, maximal tree height at 20, number of features per node at 55 and the neighborhood size at 100. The random forest is trained on a subset from the given training set (see Section 4.2). The

	This Work						Baseline					
	v	w	p	g	f	Re.	v	w	p	g	f	Re.
v	242515	3575	1332	7287	12616	0.907	223148	19042	10308	208	724	0.835
w	933	1228	628	203	802	0.324	906	1831	125	208	724	0.483
p	3557	937	1565	341	1533	0.197	1432	1378	4729	8	386	0.596
g	8638	1599	420	915344	8145	0.98	11356	4690	501	917418	191	0.982
f	19514	1584	1268	17063	71683	0.645	9128	7506	5577	237	88664	0.798
Pre.	0.881	0.138	0.300	0.974	0.756		0.907	0.053	0.223	0.996	0.876	
IoU	0.808	0.107	0.135	0.954	0.534		0.769	0.0502	0.194	0.978	0.542	

Table 4.3.: Confusion matrix for the test results of the trained classifier on test set of the Oakland 3D point cloud dataset. v – vegetation, w – wire, p – pole/trunk, g – ground, f – facade. Re.: Recall, Pre.: precision. Reference labels are shown in the left column, the predicted labels in the upper row. Overall accuracy of our method: 0.931, baseline: 0.933. Average intersection over union of our method: 0.508, baseline: 0.542.

subset has a balanced label distribution with each class has 1000 points. The obtained classifier is tested on the 1.3M-point test set. We compare our results to the results reported in [50], not only because the same dataset is used, but the results found there are detailed. Better results on this dataset are reported in recent published papers [48], but they are not as detailed as the one in [50] on the one hand, and may not be reproducible on the other hand. We present our results in Table 4.3 (baseline is also include in the table for easy comparison [4]).

We observe that our results match the baseline (0.931 vs. 0.933) in terms of overall accuracy and are comparable in terms of average intersection over union to the baseline (0.508 vs. 0.542). Our aIoU is only about 6% less than the baseline. But if we take a closer look at the table, we can find that the IoU of the wire class defeats that of the baseline. One should first note that either the wire class or the pole class is relatively more difficult than the other classes to be categorized [5] not only because they are the minority in the dataset [6], but they both lack prominent features, especially the wires, which are slim and long shaped. Intuitively, only linearity property can be favored in this case. Although the baseline receives a relative high recall score for the wire class (0.483) when compared with our recall score (0.324), the precision score of which is much lower (0.053) than ours (0.138), making our IoU twice as much as that of the baseline. This roughly implies that our classifier is 2 times better in recognizing

[4] In the original paper, only recall and precision are provided. These metrics only reflect one aspect of the test results if used individually as discussed in Section 4.1. So they are converted into intersection over union using Equation 4.1.2.

[5] This is proved by the low IoU returned by both our work and the baseline.

[6] Cf. Figure 4.2

the most difficult objects.

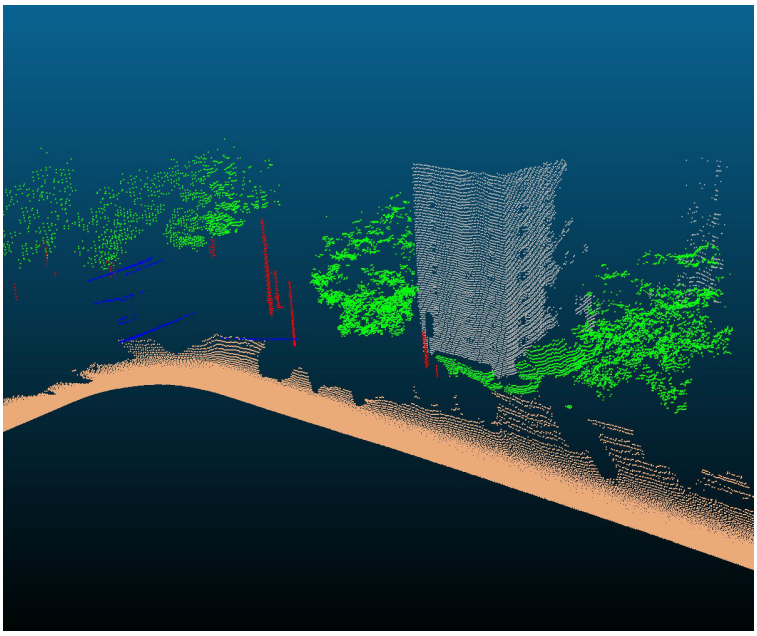
It should also be pointed out the results obtained by the baseline is based on the selection of best features from each point's feature vector. Since the selection procedure is heuristic which means it is inefficient, the training time should be very long despite the time consumption is not specified in the paper ~~of baseline.~~ Besides, which features to select as the "best features" is task and data dependent. This gives it limited power to adapt the framework to different datasets. If the framework ~~is~~ able to generalize to different datasets, all the features in each point's feature vector should be used as described in [50], in which case, the performance of the baseline deteriorates. Our results are seen outperform both metrics of the baseline, whose overall accuracy drops to 0.895 and aIoU drops to 0.463 respectively. Especially, our method outdoes the baseline in classifying the wire class and the pole/trunk class by a large margin, where the baseline reaches an IoU of 0.0458 for the wire class (ours: 0.107) and 0.0677 for the pole/trunk class (ours: 0.135).

This proves the random forest-based feature learning capability of the proposed framework, that is, the random forest itself will select the best features from all the possible features to do the classification. No hand-picking the feature is necessary. This make it possible to apply the proposed framework to different datasets without big modification.

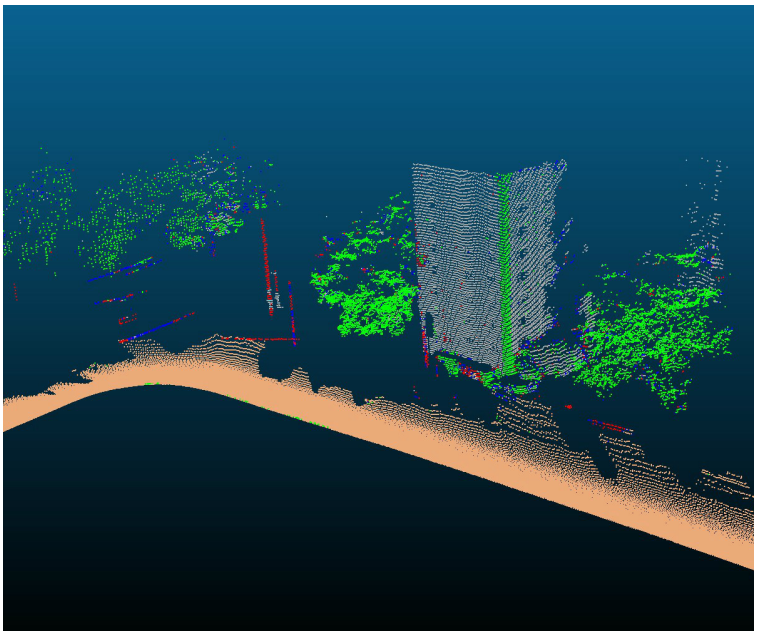
A visual representation of the test results is given in Figure 4.8.

4.7. Test Results on Paris-rue-Madame Database

To see the potential of the proposed algorithm, experiments on a second real world dataset are conducted. The Paris-rue-Madame database is much more challenging than the Oakland 3D point cloud dataset in several aspects. First, Paris-rue-Madame has much more points. The giant number of points in the cloud is more similar to the situations nowadays due to the development of measuring technology. Second, there are more categories appearing in the Paris-rue-Madame. In the Oakland dataset, points are grouped into 5 classes, whereas in the Paris-rue-Madame dataset, 17 different classes are seen. Many of the classes have far less points and are visually similar, leading it difficult to distinguish from one class to another even with human eyes. Third, the Paris-rue-Madame dataset has a wider range of point density. Naturally, in regions that are nearer to the sensor, point density is higher while point density becomes low when the sensor moves away from an object. In Oakland point cloud, the point density is almost constant.



(a) Ground truth of test set



(b) Predicted labels of test set

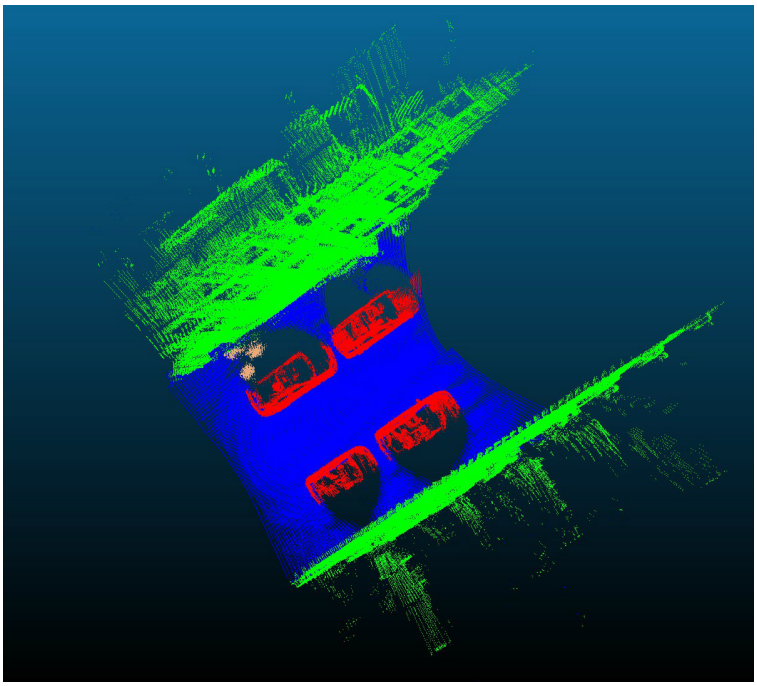
Figure 4.8.: Test results on test set of Oakland 3D point cloud dataset. Only part of the entire test set is shown in the figure. Colors: vegetation - green, wire - blue, pole - red, ground - brown, facade - gray

In order to keep consistency with the former experiments made on the Paris-rue-Madame dataset as seen in other works, for example in [49], [43], only four dominant classes are kept, namely, facade, ground, cars and pedestrians. These four classes already take up 99% of the entire dataset either in the first or the second file. In the training phase, 2000 points from each of the four classes are randomly sampled from the first file (GT_Madame1_2). A random classifier is trained with the number of trees being 50, the maximal tree height being 25, the number of features per node being 50 and the number of neighboring points being 4. Due to the limitation of the hardware[7], the trained classifier is tested on a subset (a continuous part cropped out of the original point cloud, approximately 1.6 million points) of the second file. Note that this kind of test still makes sense because, on the one hand, all the points in the test set belong to exactly one of the four classes appeared in the training set, the geometry and density is not changed. On the other hand, no points in the test set has been seen by the classifier during training. A relatively good result is achieved. We report the confusion matrix and relevant metrics in Table B.9 and depict the result in Figure 4.9.

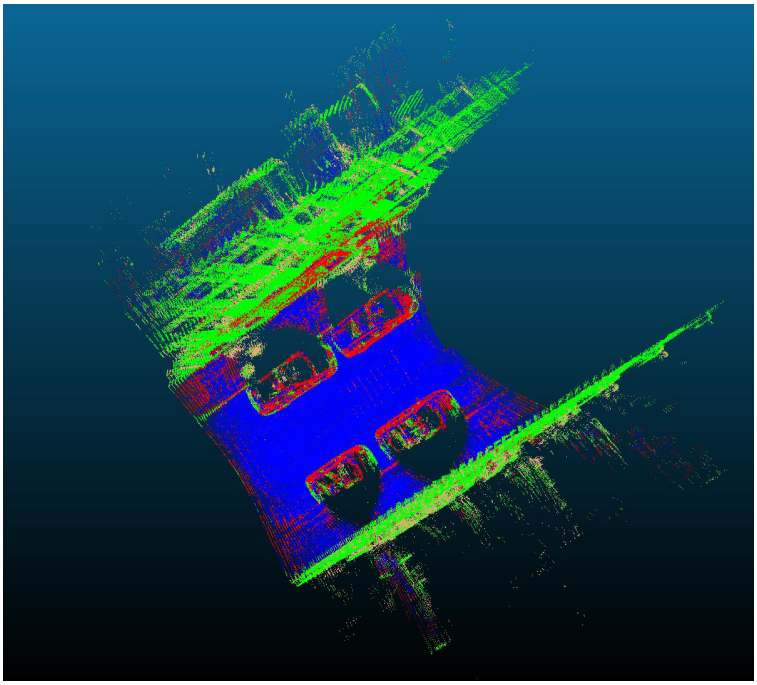
Although our results on this dataset fall behind the baseline but we think it's still within the comparable range. Especially, our performance and the baseline are neck and neck in classifying the ground class and the pedestrian class, but loses to the baseline in classifying the facade class and the car class because our method is confused by the two classes. Many points of the facade class are labeled car and vice versa. A solid explanation for this cannot be established since our method can perform better in classifying the difficult pedestrian class than the baseline. A hypothesis is that the classifier is not exposed well enough to the training instances. The cars come in many differing appearances but only some of them are included in the training set. Besides, strictly speaking, the comparison to the baseline is not fair because the test carried out here uses only part of the datasets. Noises appearing at the cleavages are induced by cropping out part of the points from the original complete point cloud. For instance, a car may be divided into two parts because of the cropping which makes the car no more look like a car. The noises will certainly bring in extra difficulties into the task.

Since the aim of the work is to propose a new framework that can exploit the feature learning capability of the random forest but not to beat the state-of-the-art method, the result has sustained the primary aim in that sense. Nevertheless, some more discussion about the method as well as the result is necessary.

[7] Consider that the "standard" hardware configuration in this field of study is an i7 workstation processor with at least 32 GB memory, our hardware is really limited.



(a) Ground truth of test set



(b) Predicted labels of test set

Figure 4.9.: Test results on a subset of the Paris-rue-Madame dataset (file 2). Colors: facade - green, ground - blue, cars - red, pedestrians - brown

4.8. Discussion and Future Work

Most analysis is given along the respective experiment results, some common issues remain to be further discussed. They might serve as potential aspects from which the proposed algorithm can be bettered.

The most important one is the node test functions. In the formulation of the proposed algorithm, most node test functions will "fail" because many of the sub-voxels are empty, in which case the tests cannot be performed. Though the medians of the successfully projected values are selected as the threshold of such test functions, in the hope that a balanced tree is grown which in turn means the information will be used as much as possible, the many empty-voxels prevent the tree growing that way because the median value we choose can't reflect the true median of all the data point projections. Intuitively, a number of such projections is valued "-1" and are therefore ignored in the process of choosing the median. Only the remaining fewer projections, the ones whose test functions return a success, is used by the node. This indicates that most of the information encoded in the neighborhood of a point is not used. To avoid the above mentioned situation, a more elaborate selection of the threshold or a re-definition of the neighborhood should be considered.

The next thing one should pay attention to is the memory consumption and training/test time balance. A proof by experiment has been given in the previous section to demonstrate that one can trade memory with time. However, extra optimization of the trade-off between memory and time can be realized. To boost the training/test speed, a direct way to achieve this is to parallelize the construction of the forest. A random forest provides this potential in its natural structure, that it, each tree is individually grown. Ideally all the trees can be trained at the same time using parallel programming [8]. Furthermore, the memory footprint can be saved during testing by letting forgo the precomputation step. During predicting phase, only one node test will be performed in each node, the neighborhood of the point can be recovered before the actual predicting and destroyed afterwards. Since only a fraction of the points are used for training while a very large number of points are tested during predicting phase, if all values pertaining to the precomputation step are calculated and stored, it will cause a great burden on the memory.

At last, it should be pointed out that the proposed framework is applicable to different kinds of datasets, whether they has additional information like colors or not. This can be seen from experiments on the synthesized dataset, which has color information, and the real world datasets, in which no such information is available.

[8] This is not the focus of this thesis, so the parallel programming is not realized.

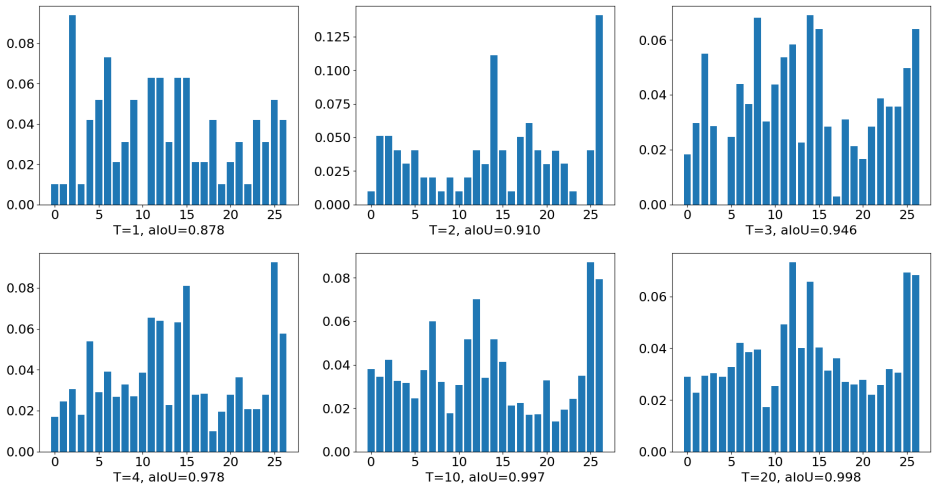


Figure 4.10.: Distribution of features being used in each node. The selection of features in individual trees seems random (when $T = 1$). As the number of trees increases, the selection of features begins to show a pattern (lower row).The distribution is generated by calculating how often a feature is selected by the node test functions in the whole forest when $T \geq 2$.

Under any circumstance, the random forest can learn the suitable features to do its job. Especially, when one takes a closer look at the synthesized dataset experiment, despite the color information alone may separate the two classes, both color and geometry properties are actually used. Also, with more trees are grown in the forest, the average intersection over union also grows up, and the features that are selected by the random forest also begin to show a consistent pattern. For example, when there are fewer trees in the forest, which features are used in each node of each tree seems random. With the growing number of trees, the features that can better classify the data points will be favored as a whole, see Figure 4.10. This validates that the random forest can select the appropriate features to do its job.

5. Conclusion

The aim of the work is to leverage the potential of a random forest's feature learning capability to semantically classify a point cloud without explicit feature precomputation and selection as the general approaches would do. It is intended to build an end-to-end framework that can directly process a raw point cloud and give each point a semantic label as output.

In previous works, a feature vector that incorporates as many features as possible in order to describe the query point from different aspects will first be designed, followed by a manual selection of which features should be used as the actual components of a feature vector that can better the test results in a trial-and-error way. This kind of processing a point cloud is inefficient in terms of both time and memory cost. Moreover, such a framework is data dependent and cannot be generalized well to different datasets.

By exploiting the node test functions of the decision trees in a random forest, namely, in each node, randomly selected sub-voxels of the query point is projected to a scalar value that encodes a certain property in the neighborhood of the query point, it is shown that the random forest can automatically find the best features which split the data points into more pure subsets. Going this way, the effort to explicitly compute feature vectors and to manually select the best features is saved.

To test its correctness, the framework was first applied to a synthetic dataset. It was found that the framework indeed worked very well in this simple situation. In order to save the training time, a minimal amount of work, that is, precomputing the values that are going to be used repeatedly during training can boost the speed significantly. The time cost of the precomputing step was negligible when compared with the actual time cost for training the model, especially when the size of the model expanded. However, during predicting, such precomputing might also be saved to reduce the memory expenditure.

To show the effectiveness of the proposed method, experiments on real world datasets were conducted. These experiments demonstrated that on the easier dataset, the framework can catch up with or even surpass to some extent the state-of-the-art performance in terms of intersection over union or overall accuracy. However, on the

more complicated dataset, the performance fell behind the best performance found in the literature but was still comparable to the baseline. Hypotheses and analysis of the possible reasons were provided in the context of respective experiments. The most noticeable reason is the accredited with the empty sub-voxels appeared in the neighborhood. This implies a direction of future work, namely to find a better way in dealing with the empty sub-voxels or find a more proper definition of the neighborhood.

Conclusively, this thesis explored the feature learning capability of a random forest and found it has a strong potential to semantically classify a 3D point cloud without precomputing/manually selecting appropriate features.

List of Figures

List of Tables

List of Algorithms

A. Label Distribution of Datasets Used in this Work

	plane	sphere	total
trainset	40000	40000	80000
testset	160000	100000	260000

Table A.1.: Label distribution of synthesized dataset

	shrub	wire	pole/trunk	ground	facade	total
trainset	14441	2571	1086	14121	4713	36932
valset	8485	899	1441	67419	13271	91515
testset	267325	3794	7933	934146	111112	1324310

Table A.2.: Label distribution of Oakland dataset

	GT_Madame1_2	GT_Madame1_3
background	3	11
facade	4769417	5209018
ground	4333059	3691236
cars	790822	1044561
lightpoles	0	2610
pedestrians	3656	6392
motocycles	81745	17122
traffic signs	11463	4017
trash can	2542	2144
wall light	3030	2356
balcony plant	983	791
parking meter	111	2374
fast pedestrian	1915	7515
wall sign	384	1246
pedestrian and others	491	0
noise	379	5173
pot plant	0	3434
total	10,000,000	10,000,000

Table A.3.: Label distribution of Paris-rue-Madame database

B. Detailed Test Results of Some Experiments Carried Out in this Work

In the following tables, if not stated otherwise, symbols have the meanings:

- t_1: Time used for searching the nearest neighbors during training
- t_2: Time used for precomputation in the training phase
- t_3: Time used for training
- t_4: Time used for searching the nearest neighbors during predicting
- t_5: Time used for precomputation in the predicting phase
- t_6: Time used for predicting

All time consumption is measured in seconds (s).

k	t_1	t_2	t_3	t_4	t_5	t_6	OA	aIoU
10	0.0592	0.367	246.13	0.263	4.054	116.81	0.488	0.206
20	0.0515	0.524	264.74	0.468	5.464	123.95	0.633	0.288
30	0.0587	0.696	267.16	0.588	7.402	135.52	0.725	0.348
40	0.0704	0.807	280.53	0.773	9.323	143.12	0.817	0.421
50	0.0964	0.925	293.10	0.881	11.253	169.13	0.860	0.453
60	0.0992	1.161	313.18	1.071	12.439	172.84	0.877	0.476
70	0.0981	1.065	285.74	1.183	12.697	186.89	0.876	0.475
80	0.104	1.114	312.60	1.407	14.127	194.66	0.878	0.484
90	0.115	1.371	314.66	1.594	14.889	211.80	0.888	0.496
100	0.133	1.262	319.89	1.748	15.650	218.14	0.875	0.479
110	0.188	1.632	328.98	1.783	16.534	232.42	0.882	0.487
120	0.212	1.521	343.38	1.966	17.330	230.80	0.866	0.470
130	0.210	1.566	362.73	2.168	18.078	263.44	0.860	0.471
140	0.241	1.612	370.42	2.314	19.126	258.77	0.859	0.469
150	0.247	1.578	387.71	2.773	20.215	264.44	0.853	0.461
160	0.254	1.565	403.85	2.981	20.963	274.29	0.857	0.466
170	0.290	1.561	414.28	3.261	21.289	285.32	0.851	0.462
180	0.311	1.513	405.88	3.449	21.012	298.13	0.854	0.464
190	0.365	1.630	409.21	3.734	22.897	311.17	0.853	0.462
200	0.375	1.625	417.78	3.797	22.995	320.03	0.852	0.458

Table B.1.: Detailed results by varying neighborhood size k on Oakland 3D point cloud dataset. Number of trees = 20, number of features per node = 20, maximal tree height = 10.

	vegetation	wire	pole	ground	facade
vegetation	7426	279	402	122	256
wire	127	349	172	5	246
pole	269	143	951	24	54
ground	1495	445	613	64251	615
facade	1408	1483	1527	577	8276
IoU	0.63	0.107	0.229	0.943	0.573

Table B.2.: Confusion matrix for neighborhood size = 90 in Table B.1. aIoU = 0.496, OA = 0.888

n_f	t_1	t_2	t_3	t_4	t_5	t_6	OA	aIoU
20	0.115	1.371	314.66	1.594	14.889	211.86	0.888	0.496
25	0.156	1.447	407.57	1.393	15.154	208.55	0.892	0.504
30	0.195	1.479	466.59	1.459	15.635	206.94	0.893	0.501
35	0.117	1.192	543.92	1.425	15.113	205.93	0.903	0.521
40	0.114	1.182	611.06	1.492	15.025	205.97	0.901	0.524
45	0.136	1.238	730.51	1.485	15.177	210.01	0.903	0.524
50	0.136	1.181	781.57	1.442	15.018	207.63	0.892	0.506
55	0.110	1.179	839.86	1.470	14.935	216.51	0.909	0.532
60	0.168	1.389	932.15	1.444	15.210	199.06	0.907	0.529
65	0.154	1.176	990.32	1.327	14.983	202.21	0.907	0.530
70	0.128	1.316	1063.41	1.443	15.038	204.72	0.901	0.520
75	0.115	1.438	1117.33	1.414	15.729	203.11	0.899	0.519
80	0.142	1.198	1170.54	1.435	15.673	202.89	0.903	0.524
85	0.124	1.410	1252.89	1.497	15.062	208.18	0.901	0.521
90	0.118	1.329	1316.63	1.428	14.389	206.46	0.904	0.521
95	0.171	1.216	1403.27	1.411	15.375	204.48	0.907	0.530
100	0.168	1.268	1548.79	1.456	16.646	205.03	0.907	0.528

Table B.3.: Detailed results by varying number of features per node n_f on Oakland 3D point cloud dataset. Number of trees = 20, maximal tree height = 10, neighborhood size = 90.

	vegetation	wire	pole	ground	facade
vegetation	7427	302	382	74	300
wire	128	353	168	1	249
pole	203	146	935	34	123
ground	718	284	461	65346	610
facade	1229	1555	1046	281	9160
IoU	0.690	0.111	0.267	0.964	0.629

Table B.4.: Confusion matrix for number of features per node = 55 in Table B.3. aIoU = 0.532, OA = 0.909

T	t_1	t_2	t_3	t_4	t_5	t_6	OA	IoU
5	0.118	1.124	212.01	1.417	14.977	66.33	0.842	0.434
10	0.146	1.366	449.37	1.415	15.123	112.23	0.870	0.468
15	0.147	1.192	632.39	1.439	15.260	158.16	0.896	0.511
20	0.110	1.179	839.86	1.470	14.935	216.51	0.909	0.532
25	0.133	1.189	1072.79	1.422	15.107	248.86	0.904	0.522
30	0.155	1.167	1276.62	1.446	15.123	286.88	0.907	0.529
35	0.141	1.175	1392.68	1.477	14.989	301.65	0.908	0.531
40	0.113	1.268	1670.50	1.436	15.120	378.98	0.909	0.533
45	0.128	1.174	1898.62	1.429	15.872	424.02	0.911	0.537
50	0.161	1.287	2129.04	1.481	15.040	470.90	0.912	0.539
100	0.128	1.121	4110.91	1.425	14.757	889.31	0.913	0.540

Table B.5.: Detailed results by varying number of trees T on Oakland 3D point cloud dataset. number of features per node = 55, maximal tree height = 10, neighborhood size = 90.

	vegetation	wire	pole	ground	facade
vegetation	7655	307	295	33	195
wire	119	361	165	1	253
pole	280	176	924	6	55
ground	769	281	460	65526	383
facade	1295	1795	854	249	9078
IoU	0.699	0.104	0.287	0.968	0.641

Table B.6.: Confusion matrix for number of trees = 100 in Table B.5. aIoU = 0.540, OA = 0.913

H	t_1	t_2	t_3	t_4	t_5	t_6	OA	IoU
2	0.103	1.014	107.99	1.484	16.163	51.18	0.772	0.298
3	0.145	1.231	217.12	1.439	15.787	81.41	0.843	0.408
4	0.173	1.366	422.61	1.530	14.833	104.47	0.862	0.443
6	0.114	1.128	599.29	1.455	15.295	144.69	0.883	0.479
8	0.120	1.086	763.52	1.435	15.301	173.88	0.888	0.497
10	0.110	1.179	839.86	1.470	14.935	216.51	0.909	0.532
12	0.143	1.198	979.79	1.445	15.160	218.16	0.904	0.524
14	0.144	1.290	1038.39	1.550	15.426	234.76	0.908	0.535
16	0.136	1.241	1101.88	1.530	14.831	223.59	0.908	0.534
18	0.161	1.414	1134.18	1.531	15.044	216.99	0.903	0.529
20	0.142	1.118	1114.70	1.408	15.221	223.51	0.914	0.542

Table B.7.: Detailed results by varying maximal tree height H on Oakland 3D point cloud dataset. number of trees = 20, number of features per node = 55, neighborhood size = 90.

	vegetation	wire	pole	ground	facade
vegetation	7461	247	373	94	310
wire	130	370	132	1	266
pole	265	207	875	19	75
ground	809	373	326	65399	512
facade	1305	1396	776	279	9515
IoU	0.679	0.119	0.287	0.964	0.659

Table B.8.: Confusion matrix for maximal tree height = 20 in Table B.7. aIoU = 0.542, OA = 0.914

	facade	ground	cars	pedestrians	Recall	aIoU
facade	616943	9915	223670	87070	0.658	-
ground	13347	469137	25434	5334	0.914	-
cars	16859	31859	109519	4338	0.674	-
pedestrians	1030	7	50	3336	0.754	-
Precision	0.952	0.918	0.305	0.033	-	-
IoU	0.637	0.845	0.266	0.033	-	0.445
IoU (baseline)	0.918	0.849	0.555	0.016	-	0.585
IoU (SotA)	0.982	0.966	0.954	0.779	-	0.920

Table B.9.: Confusion matrix and relating metrics of the test results on Paris-rue-Madame database. Overall accuracy of our method:0.741. Baseline results from [49], State-of-the-art (SotA) results from [43].

Acknowledgments

First, my sincere gratitude goes to Dr.-Ing. Ronny Hänsch, my thesis supervisor, who gave me wise guidance on every critical step in my completing of this work. You always answered my questions not only in a timely manner, but patiently and insightfully with your knowledge and expertise. It is your generous help that makes the completion of my thesis possible. Thank you, Ronny!

I'd also like to express my deepest gratefulness to my parents. They not only bestowed me a life but more importantly, they raised me well-behaved and taught me early childhood ABCs. Especially my mother, an ordinary Chinese woman, but a great woman in my life, without her unconditional love and support, I couldn't have finished my study.

Last but not least, I would like to say thank you to Technische Universität Berlin for her diversity, generosity and reputability, who offered me, an international student, a free higher education with world-class quality.

Bibliography

[1] Y. Amit and D. Geman. *Randomized Inquiries About Shape: An Application to Handwritten Digit Recognition.* Tech. rep. CHICAGO UNIV IL DEPT OF STATISTICS, 1994.

[2] C. Becker, N. Häni, E. Rosinskaya, E. d'Angelo, and C. Strecha. "Classification of Aerial Photogrammetric 3d Point Clouds." In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* (2017), pp. 3–10.

[3] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008, pp. 99–101.

[4] A. Boulch, J. Guerry, B. Le Saux, and N. Audebert. "SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks." In: *Computers & Graphics* (2017).

[5] L. Breiman. "Bagging predictors." In: *Machine learning* 24.2 (1996), pp. 123–140.

[6] L. Breiman. "Random forests." In: *Machine learning* 45.1 (2001), pp. 5–32.

[7] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone. "Classification and regression trees." In: (1984), pp. 24–28, 103.

[8] A. Criminisi and J. Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013, pp. 21, 25–43.

[9] J. Demantke, C. Mallet, N. David, and B. Vallet. "Dimensionality based scale selection in 3D lidar point clouds." In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38.Part 5 (2011), W12.

[10] S. Filin and N. Pfeifer. "Neighborhood systems for airborne laser data." In: *Photogrammetric Engineering & Remote Sensing* 71.6 (2005), pp. 743–755.

[11] J. H. Friedman, J. L. Bentley, and R. A. Finkel. "An algorithm for finding best matches in logarithmic time." In: *ACM Trans. Math. Software* 3.SLAC-PUB-1549-REV. 2 (1976), pp. 209–226.

[12] B. Fröhlich, E. Rodner, and J. Denzler. "Semantic segmentation with millions of features: Integrating multiple cues in a combined random forest approach." In: *Asian conference on computer vision*. Springer. 2012, pp. 218–231.

[13] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik. "Recognizing objects in range data using regional point descriptors." In: *European conference on computer vision*. Springer. 2004, pp. 224–237.

[14] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky. "Hough forests for object detection, tracking, and action recognition." In: *IEEE transactions on pattern analysis and machine intelligence* 33.11 (2011), pp. 2188–2202.

[15] A. Golovinskiy, V. G. Kim, and T. Funkhouser. "Shape-based recognition of 3D point clouds in urban environments." In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 2154–2161.

[16] L. Gomes. "When will Google's self-driving car really be ready? It depends on where you live and what you mean by" ready"[News]." In: *IEEE Spectrum* 53.5 (2016), pp. 13–14.

[17] E. Grilli, F. Menna, and F. Remondino. "A review of point clouds segmentation and classification algorithms." In: *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci* 42.2 (2017), W3.

[18] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, and J. Wan. "3D object recognition in cluttered scenes with local surface features: a survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (2014), pp. 2270–2287.

[19] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys. "SEMANTIC3D.NET: A new large-scale point cloud classification benchmark." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. IV-1-W1. 2017, pp. 91–98.

[20] T. Hackel, J. D. Wegner, and K. Schindler. "Fast semantic segmentation of 3D point clouds with strongly varying density." In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 3.3 (2016).

[21] R. Hänsch. *Generic object categorization in PolSAR images-and beyond*. 2014.

[22] R. Hänsch and O. Hellwich. "Evaluation of tree creation methods within random forests for classification of PolSAR images." In: *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*. IEEE. 2015, pp. 361–364.

[23] R. Hänsch and O. Hellwich. "Skipping the real world: Classification of PolSAR images without explicit feature extraction." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 140 (2018), pp. 122–132.

[24] K. He, X. Zhang, S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[25] J. Huang and S. You. "Point cloud labeling using 3d convolutional neural network." In: *International Conference on Pattern Recognition*. IEEE. 2016, pp. 2670–2675.

[26] T. Kanade, C. Thorpe, and W. Whittaker. "Autonomous land vehicle project at CMU." In: *Proceedings of the 1986 ACM fourteenth annual conference on Computer science*. ACM. 1986, pp. 71–80.

[27] K. Lai, L. Bo, and D. Fox. "Unsupervised feature learning for 3d scene labeling." In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3050–3057.

[28] V. Lepetit and P. Fua. "Keypoint recognition using randomized trees." In: *IEEE transactions on pattern analysis and machine intelligence* 28.9 (2006), pp. 1465–1479.

[29] C. Mallet, F. Bretar, M. Roux, U. Soergel, and C. Heipke. "Relevance assessment of full-waveform lidar data for urban area classification." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 66.6 (2011), S71–S84.

[30] A. Martinovic, J. Knopp, H. Riemenschneider, and L. Van Gool. "3d all the way: Semantic segmentation of urban scenes from start to end in 3d." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 4456–4465.

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

[32] M. Muja and D. G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." In: *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, pp. 331–340.

[33] D. Munoz, J. A. Bagnell, N. Vandapel, and M. Hebert. "Contextual classification with functional max-margin markov networks." In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 975–982.

[34] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation." In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* 1.2 (2017), p. 4.

[35] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." In: *Advances in Neural Information Processing Systems*. 2017, pp. 5099–5108.

[36] J. R. Quinlan. "Induction of decision trees." In: *Machine learning* 1.1 (1986), pp. 81–106.

[37] R. J. Quinlan. "C4. 5: Programs for Machine Learning." In: (1993), pp. 20–24.

[38] X. Roynard, J.-E. Deschaud, and F. Goulette. "Fast and robust segmentation and classification for change detection in urban point clouds." In: *ISPRS 2016-XXIII ISPRS Congress*. 2016.

[39] R. B. Rusu. "Semantic 3D object maps for everyday manipulation in human living environments." In: *KI-Künstliche Intelligenz* 24.4 (2010), pp. 345–348.

[40] R. B. Rusu, N. Blodow, and M. Beetz. "Fast point feature histograms (FPFH) for 3D registration." In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. Citeseer. 2009, pp. 3212–3217.

[41] A. Serna and B. Marcotegui. "Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 93 (2014), pp. 243–255.

[42] A. Serna, B. Marcotegui, F. Goulette, and J.-E. Deschaud. "Paris-rue-Madame database: a 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods." In: *4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014*. 2014.

[43] H. Thomas, F. Goulette, J.-E. Deschaud, and B. Marcotegui. "Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods." In: *2018 International Conference on 3D Vision (3DV)*. IEEE. 2018, pp. 390–398.

[44] F. Tombari, S. Salti, and L. Di Stefano. "Unique signatures of histograms for local surface description." In: *European conference on computer vision*. Springer. 2010, pp. 356–369.

[45] A. Toshev, P. Mordohai, and B. Taskar. "Detecting and parsing architecture at city scale from range data." In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[46] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features." In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–I.

[47] G. Vosselman, R. Klein, M. Vosselman, and H. Maas. "Visualisation and structuring of point clouds." In: *Airborne and terrestrial laser scanning* (2010), pp. 45–81.

[48] M. Weinmann, B. Jutzi, and C. Mallet. "Geometric Features and Their Relevance for 3d Point Cloud Classification." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4 (2017), p. 157.

[49] M. Weinmann, B. Jutzi, S. Hinz, and C. Mallet. "Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 105 (2015), pp. 286–304.

[50] M. Weinmann, B. Jutzi, and C. Mallet. "Feature relevance assessment for the semantic interpretation of 3D point cloud data." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 5 (2013), W2.

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

[52] Z.-H. Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012, pp. 67–76.