Skolkovo Institute of Science and Technology

GEOSPATIAL POINT CLOUD CLASSIFICATION

Doctoral Thesis

by

EMRE ÖZDEMİR

DOCTORAL PROGRAM IN ENGINEERING SYSTEMS

Supervisors
Associate Professor Alessandro Golkar
Dr. Fabio Remondino

Moscow - 2021

I hereby declare that the work presented in this thesis was carried out by myself at Skolkovo Institute of Science and Technology, Moscow, Russia and Bruno Kessler Foundation, Trento, Italy except where due acknowledgement is made, and has not been submitted for any other degree.

Candidate (Emre Özdemir)
Supervisor (Prof. Alessandro Golkar)
Supervisor (Dr. Fabio Remondino)

Abstract

Deep learning is becoming popular for a growing number of tasks as a result of recent technological advancements. Point cloud classification, in particular, has been investigated for some time with numerous methods established for various applications, including geospatial, industrial, indoor mapping, terrestrial, and so on. Among these, the focus is on the geospatial point cloud classification. While these studies in the state-of-the-art achieve promising results on geospatial point clouds in terms of accuracy, they are not very feasible to be deployed to daily applications (i.e., being used by mapping agencies for city-scale or country-scale data). This infeasibility is commonly caused by at least one of the following characteristics of the methods: being sensor-specific (i.e., LiDAR); efficiency issues (i.e., requiring expensive GPUs with high memory capacity); being trained on manually labeled massive datasets (i.e., deeper networks tend to require more training samples); and being trained for each project due to lack of generalization ability. To my knowledge, no studies in the literature have focused on establishing an approach to address all of these challenges simultaneously. Therefore, these challenges are considered to be not addressed effectively before.

Considering these challenges, the goal of this study is to develop a framework that achieves better or similar accuracies compared to the state-of-the-art with a more efficient methodology requiring less computational resources and/or processing time, while addressing aforementioned challenges more effectively. Consequently, the objectives are:

(i) Processing point clouds from different airborne data sources (any of photogrammetry or LiDAR);

(ii) Generalization ability (i.e., predicting on datasets with different acquisition setup);

(iii) Dealing with density variations both within a dataset as well as among distinct datasets;

(iv) Requiring less computational power and memory (i.e., being significantly faster and requiring 4GB GPU memory rather than 16GB);

(v) Achieving better or similar accuracy (i.e., ≳80%) with the current state-of-the-art methods.

The introduced framework relies on the following steps in order to achieve capabilities listed above, and more:

(i) Point cloud preprocessing with downsampling;

(ii) Handcrafted feature extraction;

(iii) Deep learning for classification using convolutional neural networks;

(iv) Post-processing for instance segmentation of buildings.

The introduced framework is put to a series of thorough tests, where its accuracy, computational efficiency, and generalization capability are tested. Furthermore, the framework is compared with the state-of-the-art methods in terms of accuracy and efficiency. The experiments are held using five datasets in total. Three of them are used for generalization tests, and one is used for computational efficiency tests. Based on the achieved results, the framework's capabilities are proven quantitatively.

Publications

1. Özdemir, E, Remondino, F, Golkar, A. An Efficient and General Framework for Aerial Point Cloud Classification in Urban Scenarios. *Remote Sensing*. 2021 13(10).

2. Farella, EM, Özdemir, E, Remondino F. 4D Building Reconstruction with Machine Learning and Historical Maps. *Applied Sciences.* 2021 11(4)

3. Pârvu, IM, Özdemir, E, Remondino, F. Aerial Point Cloud Classification Using an Alternative Approach for the Dynamic Computation of K-Nearest Neighbors. *Journal of Applied Engineering Sciences*. 2020 10(2):155-62.

4. Özdemir, E, Remondino, F, Golkar, A. Aerial Point Cloud Classification with Deep Learning and Machine Learning Algorithms. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2019 XLII-4/W18:843-9.

5. Özdemir, E, Toschi, I, Remondino, F. A Multi-Purpose Benchmark for Photogrammetric Urban 3d Reconstruction in a Controlled Environment. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2019 XLII-1/W2:53-60.

6. Özdemir, E, Remondino, F. Classification of Aerial Point Clouds with Deep Learning. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2019 XLII-2/W13:103-10.

7. Grilli, E, Özdemir, E, Remondino, F. Application of Machine and Deep Learning Strategies for the Classification of Heritage Point Clouds. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2019 XLII-4/W18:447-54.

8. Özdemir, E, Remondino, F. Segmentation of 3D Photogrammetric Point Cloud for 3D Building Modeling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2018 XLII-4/W10:135-42.

9. Pârvu, IM, Remondino, F, Özdemir, E. LOD2 Building Generation Experiences and Comparisons. *Journal of Applied Engineering Sciences*. 2018 8(2):59-64.

Acknowledgments

To begin with, I would like to thank my supervisors Prof. Fabio Remondino and Prof. Alessandro Golkar, who advised me during my studies and always were on my side. In close collaboration with them, I had the best environments to be productive and succeed.

These years of work would not have been possible without my family and my one-and-only Fatma being tirelessly there for me. It is priceless to have the ones you love, and you can rely on them when you need them the most. I am sincerely grateful to them for their endless support.

Finally, I wish to thank my teammates at the 3D Optical Metrology Unit of Bruno Kessler Foundation and Strategic Innovation Research Group of Skoltech for having a great time together. They have always been helpful and good friends.

Table of Contents

List of Symbols, Abbreviations

2D – Two Dimensional

2.5D – Two-and-a-half Dimensional

3D – Three Dimensional

3DOM – 3D Optical Metrology

AI – Artificial Intelligence

ALS – Airborne Laser Scanning

CPU – Central Processing Unit

CNN – Convolutional Neural Network

DALES – Dayton Annotated LiDAR Earth Scan

DNN – Deep Neural Network

DL – Deep Learning

DTM – Digital Terrain Model

EDM – Electronic Distance Measurement

GCP – Ground Control Point

GNSS – Global Navigation Satellite System

GPU – Graphics Processing Unit

GSD – Ground Sampling Distance

IoU – Intersection-Over-Union

ISPRS – International Society for Photogrammetry and Remote Sensing

KNN – K-Nearest Neighbor

LASDU – Large-Scale Aerial LiDAR Point Clouds of Highly-Dense Urban Areas

LiDAR – Light detection and ranging

mIoU – Mean Intersection-Over-Union

ML – Machine Learning

OA – Overall Accuracy

RANSAC – Random Sample Consensus

RGB – Red, Green, Blue

RF – Random Forest

RNN – Recursive Neural Network

SAC – Sample Consensus

SGD – Stochastic Gradient Descent

SQRT – Square Root

SVM – Support Vector Machine

TLS – Terrestrial Laser Scanner

TOF – Time Of Flight

TONIC – efficienT classification Of urbaN poInt Clouds

UAV – Unmanned Aerial Vehicle

UTM – Universal Transverse Mercator

List of Figures

List of Tables

List of Algorithms

Chapter 1.

Introduction

Objects on maps or in databases connected to the Earth's surface by locations (coordinates, addresses, or different methods) are considered geospatial information, composing ~80% of all the data (Bossler et al., 2010). There are various use cases for geospatial data, including but not limited to 2D/3D cadaster, navigation, urban studies, flight simulation, and oceanography.

The type of the required geospatial data for a project depends on the purpose, so does the platform for acquiring the data. For instance, for cadaster purposes, the required accuracy for the final map is commonly fixed by the regulations, and frequently, it is in terms of centimeters, which is not possible to be achieved with spaceborne data by today's technology. Another example can be global land cover mapping (such maps include thematic information representing the type of the land, such as water, forest, or soil), which does not require the same geometric accuracy as cadaster. Considering the scale, it is more feasible to use spaceborne data for such purposes. As seen in these two examples, shown

in Figure 1-1, different types of geospatial data can be used for different purposes and can be considered complementary.

Keeping the focus on the scope of this study, the following sections will be detailing motivations, geospatial data acquisition techniques, point cloud generation and processing methods, goal and research objectives, and the structure of the thesis.

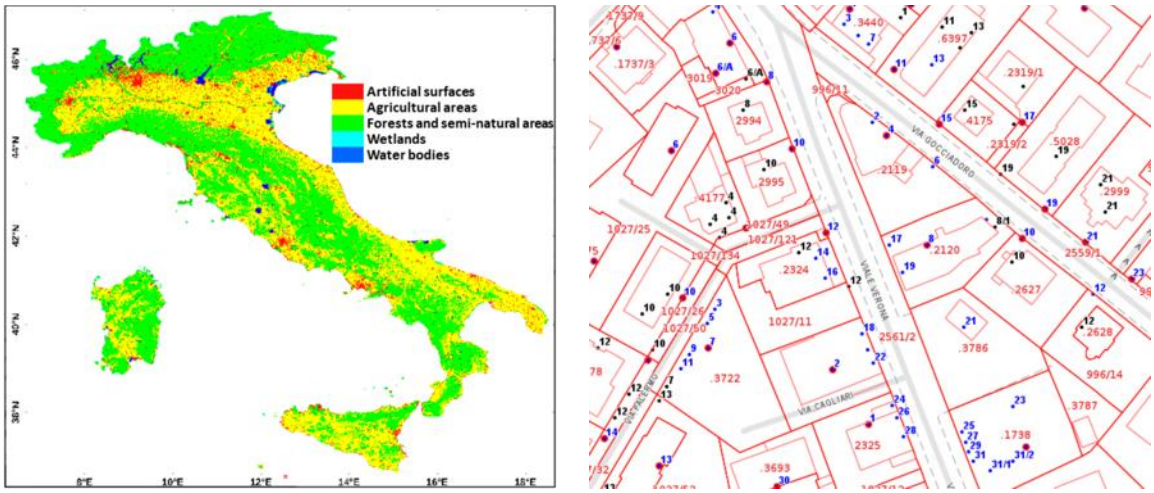


Figure 1-1. Land cover map (Pulvirenti et al., 2020) and cadastral map examples (Comune di Trento, 2021), respectively left and right images.

1.1. Motivations

In recent years, point cloud processing techniques are extensively investigated by the research community for various applications (Grilli et al., 2017; Liu et al., 2019; Bello et al., 2020). Among these, geospatial point cloud classification methods hold an important place, as assigning semantic information to these point clouds allows for the widespread use of such geospatial data (Rottensteiner et al., 2012). Such enriched point cloud could be

preparatory for change detection (Qin et al., 2016), 3D building modeling (Özdemir and Remondino, 2018), planning (Urech et al., 2020), and so on.

Being an important and challenging task, semantic enrichment of geospatial point clouds is also one of the main topics in the research community. There are many studies presented in the literature focusing on a semantic interpretation of 3D point clouds with different techniques (Su et al., 2015; Kanezaki et al., 2018; Wang et al., 2019) and for various approaches (Becker et al., 2018; Bittner et al., 2018; Huang et al., 2018; Maltezos et al., 2019), as presented in Chapter 2.

To the best of my knowledge, many of the current point cloud classification solutions are confined to either specific data (e.g., only for LiDAR acquisitions) or scenarios (indoor, outdoor, terrestrial, or aerial). This is due to (i) the complexity of the point cloud classification process, (ii) differences in the data structure, (iii) the need for specific training data, as well as (iv) generalization problems. Despite these challenges, various studies in the literature have demonstrated to achieve high accuracies ($\gtrsim$80%) in geospatial benchmark datasets. However, it was noticed that these studies are not developed to cope with data from different data sources, density variations, generalization, and low GPU memory at once. Therefore, these challenges are considered as open research issues to be faced in the PhD work. Most of the existing solutions for point cloud classification are considered impracticable for daily applications (such as city-scale or country-scale applications of a national mapping agency), which roots the motivations for developing TONIC framework (Chapter 3), i.e., a practical and powerful AI-based solution

for geospatial point cloud classification which tried to tackle the aforementioned challenges and open issues.

In summary, the overall motivations of this PhD work are as follows:

- Need for a sensor independent classification framework: the tools commonly used for daily purposes by the mapping agencies are sensor dependent, which means that the software can process data coming from a specific sensor (i.e., LiDAR), but not from an alternative (i.e., photogrammetry).
- Large-scale applicability: national mapping agencies usually deal with enormous datasets; hence memory usage, computational power, generalization capabilities, and varying data structures are all important issues.
- Time and cost optimization: many geospatial point cloud classification tasks are handled manually, whereas the automation of these steps in the mapping pipeline will speed up map-making, therefore reducing the overall cost.

1.2. Geospatial Data Acquisition Techniques

The geospatial data acquisition techniques can be categorized by the platform used, such as terrestrial, airborne, and spaceborne. In the following sections, geospatial data acquisition techniques will be detailed in these three categories.

1.2.1. Terrestrial

The history of terrestrial mapping with land surveys dates back to the Babylonian era to specify the property borders with stones (Brinker and Minnick, 1995). Since then,

terrestrial surveying has come a long way by using current technologies like total stations and laser scanners (Figure 1-2). Terrestrial geospatial data acquisition techniques can reach accuracies below a centimeter while providing fast data acquisition (Gopi et al., 2018; Stenz et al., 2020).

Figure 1-2. Trimble C5 total station (Trimble, 2021) and TLS (Riegl, 2021), respectively left and right images.

In a land surveying study, which can be considered the most basic terrestrial data acquisition, the workflow includes planning the survey, data collection in the field, post-processing of the acquired data with computers, and delivering the data (most commonly to a GIS database). Typical applications with the data acquired using terrestrial techniques include but are not limited to; cadaster, construction, topographic map production, mine surveying, GCP surveying for photogrammetry/remote sensing studies, cultural heritage modeling with terrestrial photogrammetry, and mobile mapping applications.

Figure 1-3. Example of fieldwork for GCP surveying (Ground Control Points for Drone Mapping, 2021)

1.2.2. Airborne

The first airborne data acquisition dates back to 1858 by Gaspard Tournachon using a manned balloon over Paris. Ever since, airborne data acquisition techniques have adapted the use of balloons, pigeons, kites, rockets, helicopters, airplanes, and UAVs (Eisenbeiß, 2009). Such airborne data are used for photogrammetry and remote sensing studies. Photogrammetry and remote sensing are defined by Gomarasca (2009) as follows:

"*Art, science and technology to obtain valid information about physical objects and the environment, through the processes of collection, measure and interpretation of images (photographic or digital) and analog or digital representation of the models of electromagnetic energy derived from survey systems (photographic cameras or scanning systems), without contact with the objects.*"

In order to understand the major difference between photogrammetry and remote sensing, one may prefer the definition of photogrammetry made by Luhmann et al. (2019)

as follows: "*Photogrammetry encompasses methods of image measurement and interpretation in order to derive the shape and location of an object from one or more photographs of that object.*". This definition suggests the main focus of photogrammetry as metric information ("*shape and location*") extraction using photographs. On the other hand, the definition for remote sensing made by Konecny (2014) clarifies the difference: "*Remote sensing can be considered as the identification or survey of objects by indirect means using naturally existing or artificially created force fields.*". It can be inferred from this definition that the remote sensing technique's primary focus is on thematic information ("*identification or survey objects*") extraction.

The abovementioned techniques utilize images for metric and semantic information extraction. In addition to these two, LiDAR (light detection and ranging), utilizes laser light. The laser light is used for *ranging*, which means the main focus of the technique is again on the metric information, generating a collimated light beam (Gomarasca, 2009). The LiDAR technique has different advantages and disadvantages compared to photogrammetry. On the one hand, a primary advantage can be seen as the delivery of the 3D point cloud without image processing, while on the other hand, a common disadvantage can be seen as the missing color information. One of the most recent trends in this field is using an acquisition system combining these two techniques (LiDAR and photogrammetry) to take advantage of both, which is named *hybrid*, and an example sensor can be Leica CityMapper-2 Hybrid Airborne Sensor (2021). A point cloud produced in this way includes color information from photogrammetry along with additional LiDAR features.

Besides these introductory discussions, photogrammetry and LiDAR techniques will be discussed under Section 1.3 Point Cloud Generation Methods.

1.2.3. Spaceborne

The roots of spaceborne geospatial data date back to the Cold-War era, based on the reconnaissance needs. The Soviet Union used the Zenit satellites system, while the USA used the Corona system in the early 1960s (Figure 1-4). The actual data acquisition steps were analog, based on retrieving the films from the orbiting satellites (Pelton et al., 2017).

Shifting the focus to the present day, it is not only the technology evolutions pushing spaceborne techniques but also the evolutions in the needs and common use cases. In addition to original motivations, spaceborne techniques have been one of the main data sources for modern map-making studies. The most common applications of spaceborne geospatial data are thematic maps produced with remote sensing techniques. Being acquired from space, two of the main advantages for such data can be considered large coverage (orbits covering the most if not all the Earth's surface) and high temporal resolution (due to continuous orbiting of satellites). With the recent advances in technology, spaceborne systems can achieve very high spatial resolutions such as 1 meter or better (Tapete and Cigna, 2018).

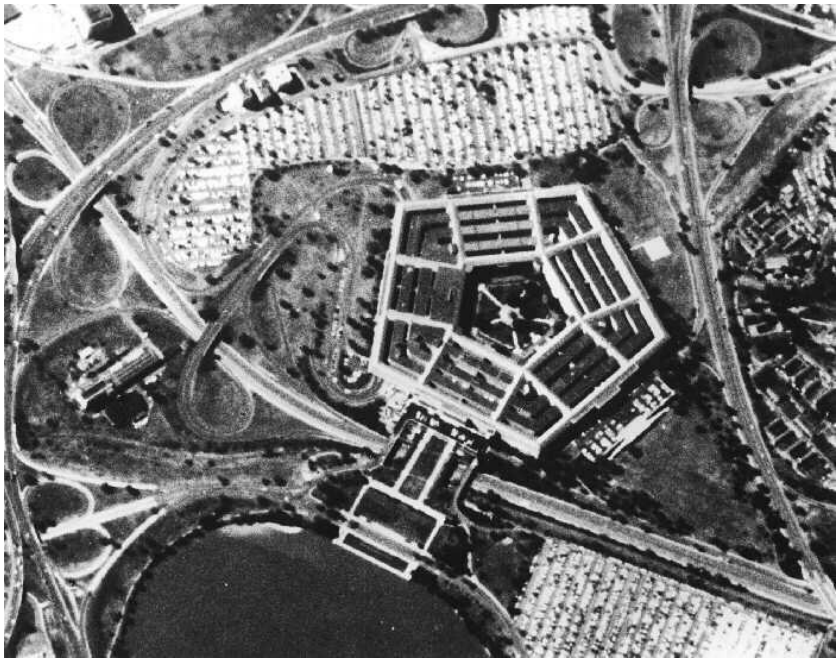


Figure 1-4. Sample Image sample from Corona Spy Satellite System, The Pentagon, 25 September 1967 (Corona (satellite), 2021)

Spaceborne data are also becoming increasingly popular for DSM generation, taking advantage of high-resolution satellite imagery and photogrammetry techniques. DSM, being a grid-based height representation (structured as an image, where pixel values are the heights) of the Earth's surface (Figure 1-5) with all the objects on it, can be used for 3D city modeling purposes as well asw many other GIS studies.

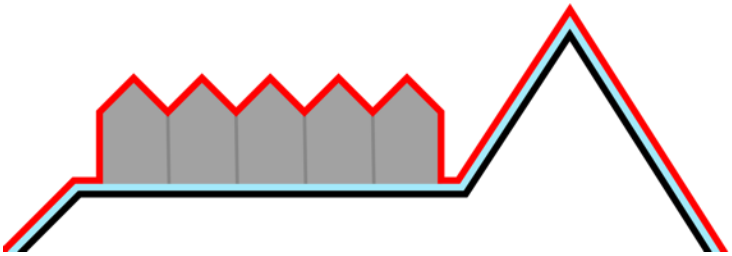


Figure 1-5. Simplified DSM (red) and DTM (blue) comparison (Digital Elevation Model, 2021).

Spaceborne DSM generation technique will be further discussed in Section 1.3 Point Cloud Generation Methods. As an example, Figure 1-6 shows two DSMs generated using airborne LiDAR (Vaihingen, Germany (Cramer, 2010)) and spaceborne imagery (Trento, Italy (Poli et al., 2013)), where both are rendered via hillshade with 45.0° altitude and 315.0° azimuth using open-source GIS software QGIS (QGIS.org, 2021).

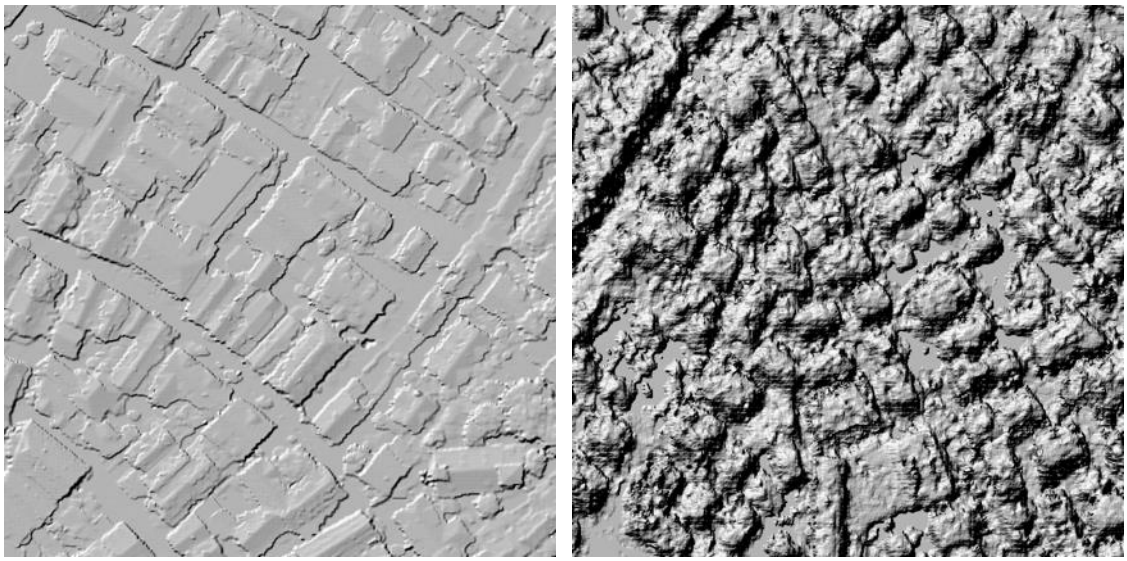


Figure 1-6. DSM samples were produced with airborne and spaceborne data, respectively left and right images.

In addition to the abovementioned techniques, GNSS is another spaceborne data acquisition technique. GNSS technique, in brief, can be described as a spaceborne radio navigation system. Due to the scope of the thesis, it is not discussed in detail, yet, readers may refer to a related book, such as the one from Hofmann-Wellenhof et al. (2012).

1.3. Point Cloud Generation Methods

Point clouds can be simply defined as a group of data points in space. In geospatial science, a 3D point cloud is defined in a projected -Cartesian- coordinate system such as UTM. As the coordinate systems are not within the scope of this study, readers may refer to a related book, such as the one from Maling (1992).

There are two common data types representing 3D geospatial data; 3D point clouds and DSMs. On the one hand, 3D point clouds have irregular distribution in space. In other words, they are not in a grid structure. On the other hand, DSMs can be described as 2.5D representations of the 3D point clouds in a grid structure. This means, to produce a DSM, the 3D point cloud is projected to a 2D grid (like an image), where each grid's value represents the height. This procedure is also known as rasterization. In Figure 1-7, a 3D irregular point cloud (Dortmund, Germany (Gerke et al., 2016)) and a DSM (Imst, Austria (Toschi et al., 2021)) are shown side-by-side. The point cloud generation techniques described in the upcoming sections are used for generating 3D point clouds. Then, based on the necessity, a DSM can be produced. DSMs have been a popular product for many mapping agencies as they include very similar geometric data yet, easier to process with the GIS techniques.

Geospatial point cloud generation methods can be named as photogrammetry and LiDAR, and the platform use can be terrestrial, airborne, and spaceborne. For the sake of content integrity, airborne and spaceborne photogrammetry, as well as airborne LiDAR techniques, will be further discussed in this section.

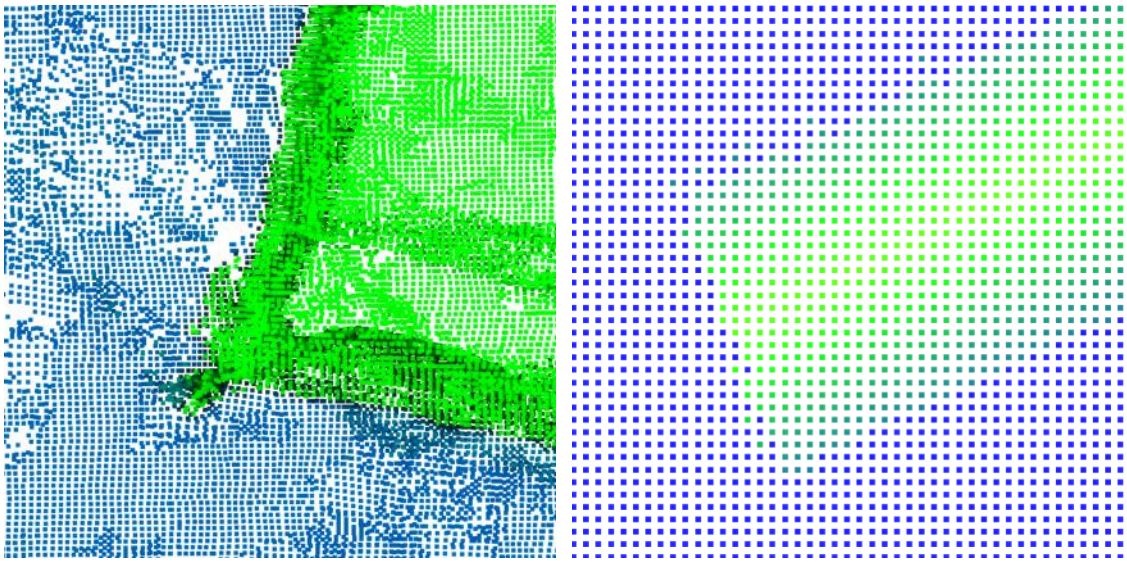


Figure 1-7. Top view for building and surroundings: 3D irregular point cloud (left) and DSM, colorized based on the height (blue: lower, green: higher).

1.3.1. Airborne and Spaceborne Photogrammetry

To describe photogrammetry, in addition to the definitions made in Section 1.2.2, one may look in detail at the word 'photogrammetry'. The word is derived from three Greek words 'photos', 'gramma' and 'metron', which mean 'light', 'drawn', and 'measure', respectively. Therefore, 'photogrammetry' can be inferred as the art and science of measuring with photographs.

The geometric basis of photogrammetry is central perspective geometry, as shown in Figure 1-8.

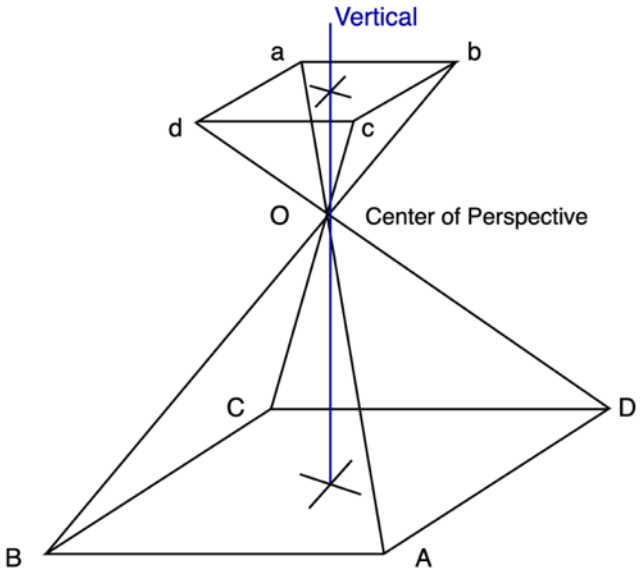


Figure 1-8. Visualization of central projection adapted from Gomarasca (2009).

Using a single photograph with photogrammetry technique will result in a scaled version of that photograph, which means the result will be in a 2D object space. Thus, in order to achieve a 3D object space, there should be at least two images to make a stereo pair, as seen in Figure 1-9. Such a stereo pair is expected to include photographs of the same objects from different locations, enabling the photogrammetric evaluation of acquired data. The object point P_{xyz} is observed from both images with image points *P'* and *P"* through a line passing through *C*. Therefore, *P'*, *C* and P_{xyz} are collinear points, as well as *P"*, *C* and P_{xyz}. This collinearity condition brings *Epipolar Plane* when combined with the *base* vector. This is known as the *coplanarity condition* in photogrammetry.

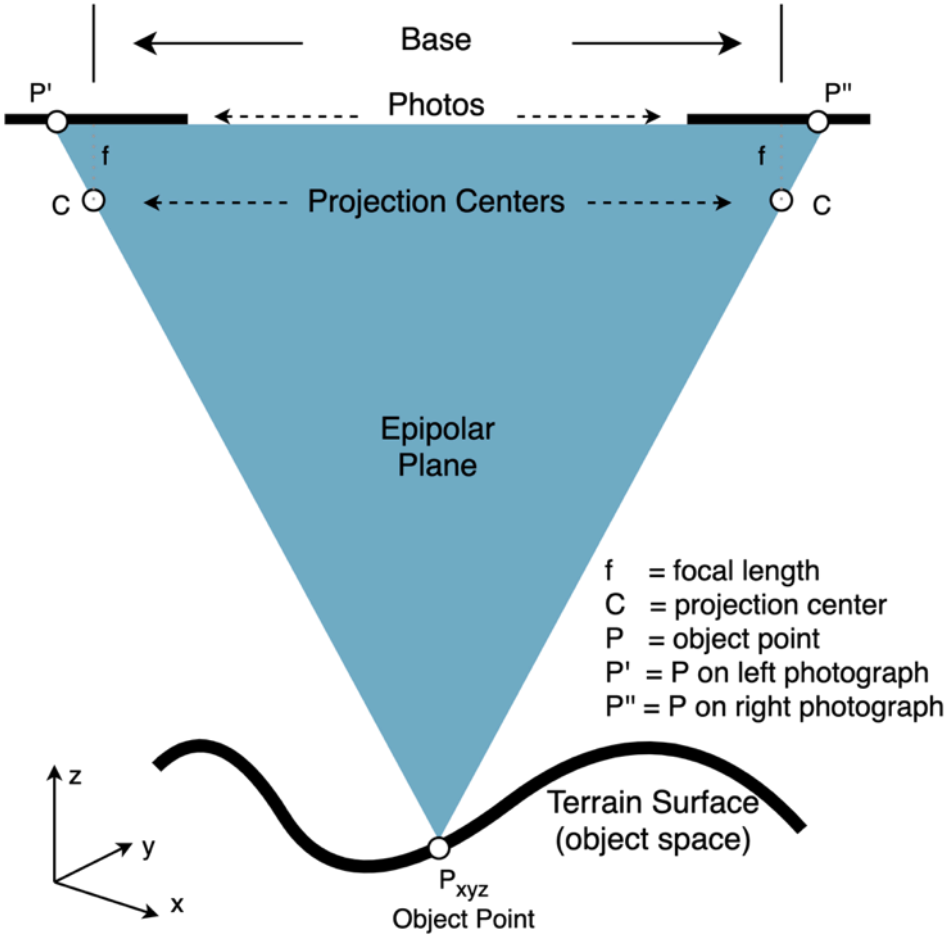


Figure 1-9. Photogrammetric stereo pair geometry overview, adapted from Linder (2009).

The mathematical model of photogrammetry is based on central projection, collinearity, and coplanarity conditions mentioned above. All these allow the relationship between the images and the objects to be defined as shown in Figure 1-10, where *FC* is the center of the image coordinate system, *PP* is where the imaginary axis of the optical center passes through the image plane and the others are as described in the figure.

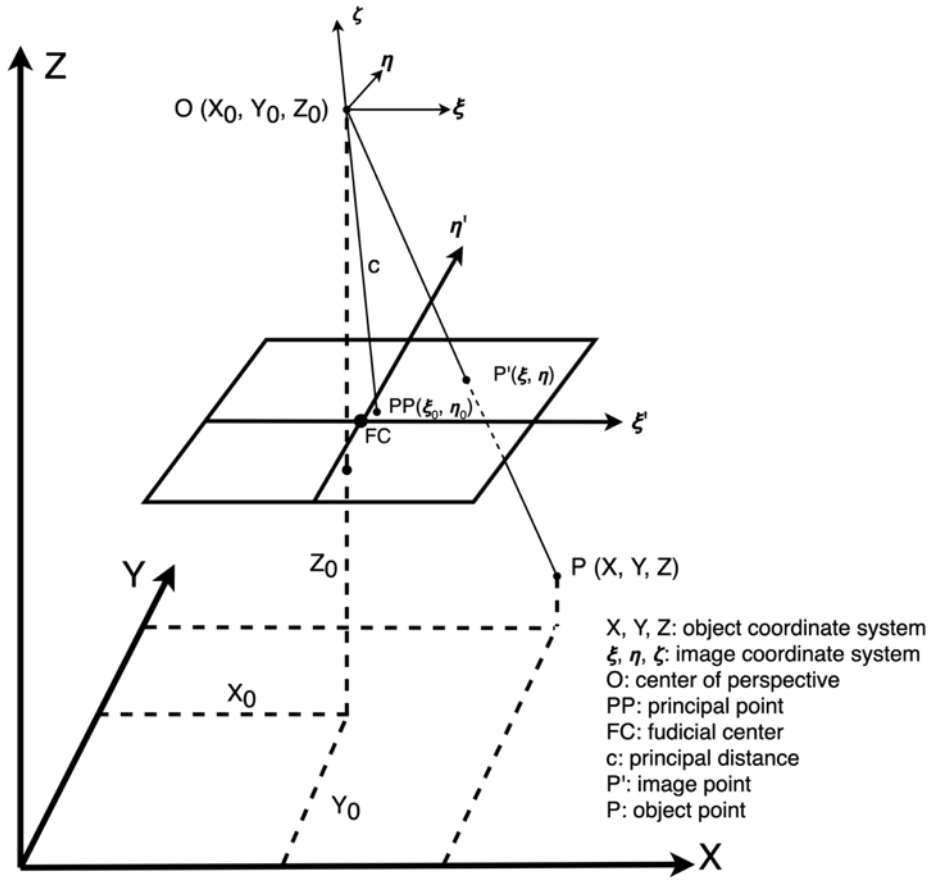


Figure 1-10. Image and object coordinates relations adapted from Kraus (2007).

The relationship shown in Figure 1-10 is explained using Equation **(**1**)**, where r_{ij} are the elements of the rotation matrix of the image calculated based on the exterior orientation parameters defining the angular position of the image around the *X*, *Y*, *Z* axes. Due to content integrity, rotation matrix and further calculations are not introduced here, yet, readers may refer to a photogrammetry book, such as Kraus (2007) or McGlone (2013).

$$\xi = \xi_0 - c\frac{r_{11}(X - X_0) + r_{21}(Y - Y_0) + r_{31}(Z - Z_0)}{r_{13}(X - X_0) + r_{23}(Y - Y_0) + r_{33}(Z - Z_0)}$$
$$\eta = \eta_0 - c\frac{r_{12}(X - X_0) + r_{22}(Y - Y_0) + r_{32}(Z - Z_0)}{r_{13}(X - X_0) + r_{23}(Y - Y_0) + r_{33}(Z - Z_0)} \quad \textbf{(1)}$$

One of the most recent trends in airborne photogrammetry is the adoption of oblique imagery. In airborne photogrammetric applications, the typical data acquisition approach used to be based on nadir-only imagery, in which a single image is acquired with a camera fixed at nadir direction (-*Z* in Figure 1-10). With the oblique image acquisition (Figure 1-11) technique, five images are acquired simultaneously: one in the nadir direction and four in the left, right, front and back directions (i.e., ±45º from *ZX* and *ZY* planes). The additional oblique images provide visibility on the facades of the buildings and other objects, increasing the completeness and the captured details of the acquired data.

Figure 1-11. Examples for nadir (left) and oblique (right) images from the 3DOMCity benchmark (Özdemir et al., 2019b).

Spaceborne photogrammetry can be seen as an extension to airborne photogrammetry, where the significant changes are in the data acquisition. These changes

can be summarized as image acquisition equipment, acquisition network, acquisition time interval, and atmospheric effects (Qin, 2019). In airborne systems, commonly, a camera captures a photograph with a central perspective, as shown in Figure 1-8. In comparison to these systems, in spaceborne photogrammetry, remote sensing satellite systems acquire the images. Such systems do not have a camera with a central perspective geometry. Instead, they commonly use multiple central perspective types of sensors (push broom or whisk broom). The acquisition network (positions of the camera centers for image acquisition) for airborne photogrammetry is designed by the experts during flight planning, where the flight altitude and overlap ratios of images are considered. In spaceborne, however, the satellites follow their orbit at a certain speed which does not allow image acquisition planning by experts. Due to the same reason, the experts cannot decide the image acquisition time intervals either.

Last but not least, another significant difference is the atmospheric effects. As the satellites used in spaceborne systems *fly* at a much higher altitude than airborne systems, the atmospheric effects will affect the images more. For further discussions on spaceborne photogrammetric systems, readers may refer to the articles such as Poli et al. (2015), Qin (2019), or Han et al. (2020), while for further information about spaceborne systems, readers may refer to a remote sensing book, such as the one from Konecny (2014).

1.3.2. Airborne LiDAR

As mentioned in the previous section, LiDAR utilizes laser light for distance measuring (ranging). LiDAR systems use one of the two common methods; time-of-flight (TOF) or phase measurement.

TOF method follows these simplified steps:

(i) emitting laser radiation and starting the time counter;

(ii) counting the time till the laser radiation returns;

(iii) stopping the time counter once the sent laser radiation is returned (detection);

(iv) distance covered by the laser radiation is calculated based on how much time it spent during the travel, as the speed is known a priori.

The alternative approach, named phase measurement, shares the same steps for emitting laser radiation and detecting its return. The difference is that the receiving sensor measures the phase difference between the sent laser radiation and the received one. The measured phase difference is then used for calculating the distance the laser radiation has traveled.

Among these two methods, the acquired data reflect the differences in the working mechanisms as well. The few major differences in the acquired data can be summarized as TOF systems expected to produce fewer points per second and operate at higher ranges than phase difference measuring systems. For further details, readers may refer to books on this topic, such as Vosselman and Maas (2010) or Shan and Toth (2018).

1.4. Point Cloud Classification

Point clouds are one of the most common products of geospatial studies. However, they are not necessarily *the final product*. Point clouds can provide a highly detailed and high-resolution 3D representation, yet, they are not very feasible when using them for map-making. One of the main reasons for that is a point cloud can give only the answer to the question 'where?', but not to 'what?'. Another reason is that a point cloud per se does not include information such as faces or edges of the represented geometry. In order to get an answer to the question of 'what?', a point cloud should be enriched with semantic information, which is possible by semantic segmentation (also known as classification). For generating further geometric information, a mesh model should be generated. Example usage of semantic information and model generation from Bergamo / Italy can be seen in Figure 1-12. The building model is generated after corresponding points are extracted from a city-scale point cloud using classification approach (Özdemir and Remondino, 2018).

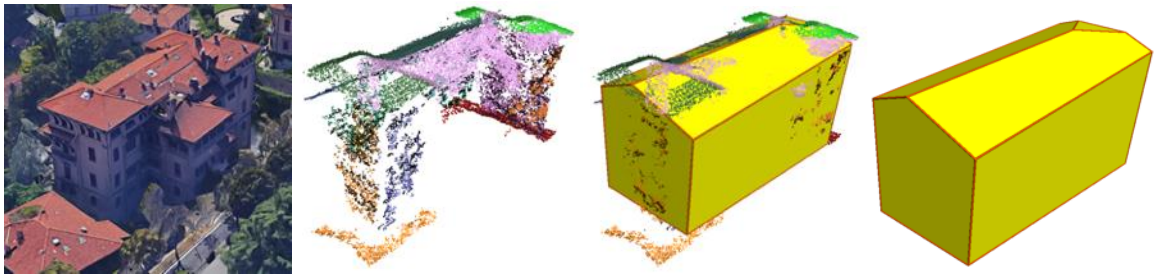
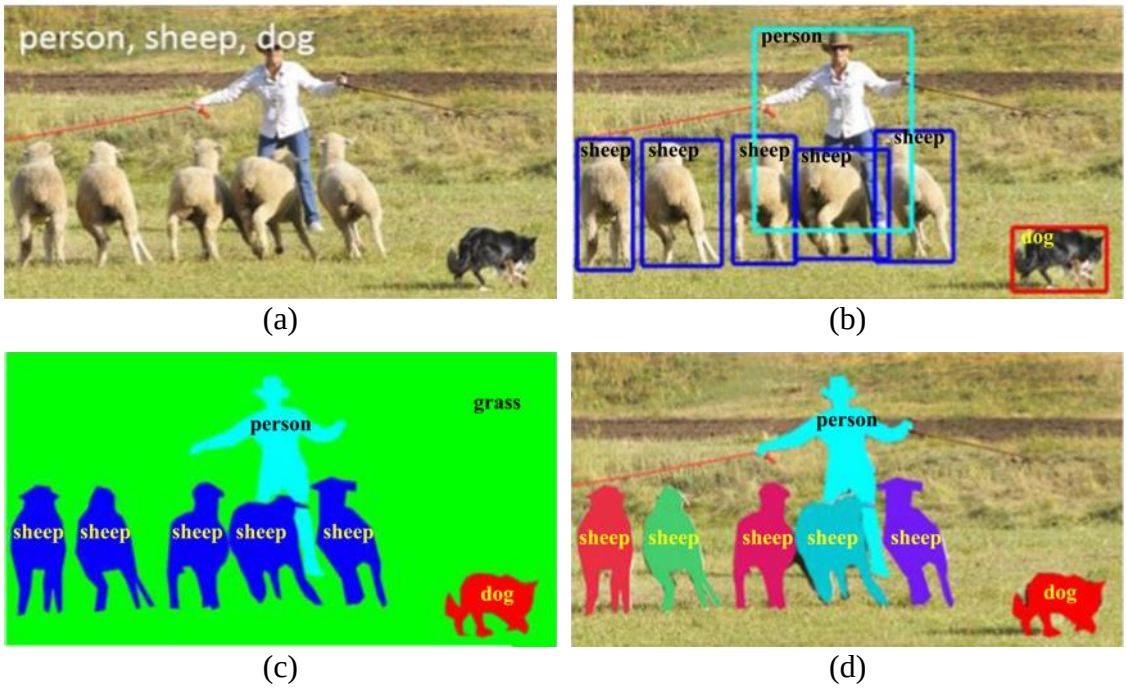


Figure 1-12. Building model generation example. From left to right: image of the building, point cloud extracted with classification and randomly colorized for each extracted planar segment, point cloud and derived 3D model of the building together, 3D model of the building.

Classification can be defined as assigning a class label to each given data point. A data point can be an image, a pixel of an image, a point cloud, or a point in a point cloud. Depending on the task, classification can be referred to with different names. For instance, assigning a class label to a point cloud (Liu et al., 2019) or an image (Liu et al., 2020) can be referred to as object classification. The terminology for the classification tasks is commonly defined as shown in Figure 1-13.



(a) (b)

(c) (d)

Figure 1-13. Classification tasks are shown on the image domain. Figure retrieved from Liu et al. (2020). (a) image object classification, (b) object detection and localization, (c) semantic segmentation, (d) instance segmentation.

As shown in Figure 1-13, image object classification refers to assigning a label (or labels) to a single input (image), where the locations of the object(s) within the input are unknown. Object detection and localization also provide the location of each object within

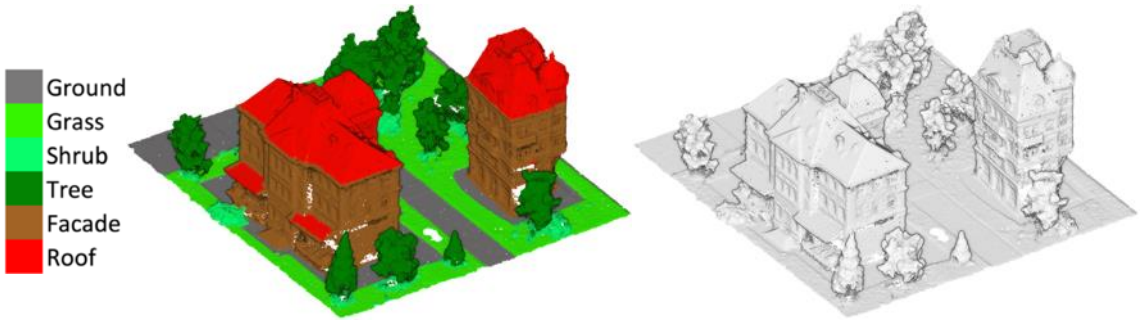
the input using a bounding box. Taking a step further with semantic segmentation, a class label is assigned to each part of the input (pixel in the image), which enables the extraction of the exact location within the input (borders of the objects within the image, instead of a bounding box), yet, the information about individual objects is possible only with the instance segmentation technique.

Various studies focus on different perspectives of deep learning (DL)-based point cloud processing, including 3D shape classification, 3D object detection (Yan et al., 2020), and 3D point cloud classification (Hackel et al., 2017). The main differences among these tasks are similarly represented in Figure 1-13. In order to clarify in the 3D point cloud domain, the definitions in the literature are given below based on Guo et al. (2020) and Liu et al. (2019):

- *3D shape classification* is where the deep neural network (DNN) learns the global shape of the given point cloud objects (i.e., a teapot, a car).
- *3D object detection* is where the DNN is fed with an entire scene and returns the bounding boxes for each one of the detected objects (i.e., pedestrians, trees, and cars in the street).
- *3D point cloud classification* is where the DNN is, again, fed with an entire scene and returns class probabilities for each point. Commonly the class label with the highest probability is assigned to each one of the points. The challenge here is that the network needs to learn both global and local geometric structures in order to succeed.

Keeping the focus on the geospatial point clouds, the term ‘classification’ in this domain is commonly used for the task where a class label is assigned to each point in the point cloud. Therefore, the closest task definition from Figure 1-13 could be the semantic segmentation, which is widely used by the community. Figure 1-14 shows a classified-versus-unclassified point cloud.

For further reading on the topic, readers may refer to a DL book (Goodfellow et al., 2016) or a survey article (Guo et al., 2020).



(a) (b)

Figure 1-14. Example on classified (a) and unclassified (b) point clouds from the 3DOMCity Benchmark (Özdemir et al., 2019b)

1.5. Goal and Research Objectives

Given the motivations (Section 1.1) and the recent developments in the state-of-the-art (Chapter 2) the goal of this study is to develop a point cloud classification framework for geospatial point clouds that achieves better or similar accuracies compared to the state-of-the-art (≳80%) with a more efficient (in terms of computational time and hardware requirements) methodology. In this way, the developed framework can be a

feasible solution for daily applications with large datasets (i.e., city-scale or country-scale). This feasibility is considered as a step forward from research projects, where the datasets are significantly smaller, towards daily applications.

The observed challenges (as mentioned Section 1.1) in the literature (Chapter 2) root the motivations to develop a framework with the following objectives in order to address these challenges more effectively:

(i) Invariant to different data sources: A framework designed for adapting any data source, including photogrammetry (oblique or nadir-only acquisitions), LiDAR, or their combination;

(ii) Generalizable: Point clouds acquired with different sensors in different locations have varying characteristics such as density. Therefore, a high generalization capability is required for being able to handle such dataset variety;

(iii) Invariant to point cloud density variations: A point cloud may have density variations, which changes the local geometry within the dataset and hampers classification;

(iv) Low computational cost and hardware requirement: The DL frameworks commonly require GPUs with high memory, which increases the hardware cost (also energy consumption in many cases). A deep neural network that cannot be used without a high-memory GPU creates a bottleneck for usefulness, especially for large-scale applications (such as nationwide mapping);

(v) Accuracy better or similar than the current state-of-the-art methods (≳80%).

Based on the way the handcrafted features treated (Section 3.1.2) with CNNs, this work can be categorized as architectural innovation. To my knowledge, no other methods use this approach. The patches are generated (a 2D matrix or a 3D array per point) using handcrafted features and coordinates of the neighboring points, and then processes as images aiming to predict a class label per point. Therefore, in terms of innovation, this approach can be considered as a new relationship between geospatial point clouds and CNNs (Henderson and Clark, 1990).

The novelty of this PhD work can be considered as the entire point cloud classification framework which links aerial point clouds with CNN methods. The developed methodology, named TONIC, is able to process aerial point clouds acquired with different sensors and at different densities. It achieves accuracy values in the same order as current state-of-the-art methods. It can be generalized to process unseen datasets and requires low computational resources compared to other existing methods. Therefore, for the first time, a geospatial point cloud classification framework can efficiently achieve high accuracies with a proven generalization capability, which supports the method towards being a feasible solution for daily applications as it would not need separate training dataset for each distinct dataset.

1.6. Thesis Structure

The thesis is structured in six chapters starting with this introductory chapter, followed by a literature review, proposed frameworks, results, validation of the proposed framework, and finalized with discussions and conclusion.

The introduction chapter is focused on motivations, geospatial data acquisition methods, point cloud generation, and processing methods, as well as discussing the goal and research objectives of the study.

The literature review chapter discusses the classification and semantic segmentation methods used for geospatial data in the state-of-the-art. Therefore, the artificial intelligence methods will be addressed initially, then, various ML and DL approaches will be reviewed, focusing on geospatial point cloud data. The section will be ended with a brief look to the state-of-the-art.

In the proposed framework chapter, geospatial point cloud classification framework will be discussed in detail, including point cloud preprocessing, feature extraction, DL approach, and post-processing for instance segmentation.

In the results chapter, the datasets to be used for validating the developed methodology will be explained and the proposed framework will be evaluated on these datasets.

In the validation chapter, the developed methodology will be evaluated for generalization. Furthermore, it will be compared to the state-of-the-art in terms of accuracy as well as hardware requirements and computation time.

In the discussions and conclusion chapter, the results will be further discussed along with the limitations of the proposed frameworks and the future works, and a conclusion will be made.

In the appendices, some codes related to the thesis are shared.

Chapter 2.

Literature Review

In this section, firstly, artificial intelligence (AI) will be discussed with the phases it has been through over the decades, along with some basic descriptions. Afterward, the literature will be reviewed for AI studies focused on geospatial point cloud classification and a summary of the state-of-the-art will be given.

2.1. An Overview of Artificial Intelligence

AI can be described as techniques allowing computers to simulate the intelligence of the human. These techniques can be as simple as a hard-coded knowledge base implementation (i.e., early AI studies with rule-based systems) or complex artificial neural networks (i.e., more recent AI studies with DL). The first studies in the AI domain date back to the early 1900s, when manually designed basic statistical methods were used for identification (Garson, 1900). Throughout the decades, AI methods evolved into different

phases of rule-based systems, classic machine learning, representation learning, and -most recently- DL:

- *Rule-based systems* depends on a knowledge base, which can be seen as a collection of hard-coded information. Received data is then evaluated based on these set of rules.
- *Classic Machine Learning* (ML) is a sort of capability, which enables the machines to gain knowledge by learning the patterns from the data itself. In this case, the data machine learns is typically the representation of the data (rather than the raw data), which is formulated by the experts and commonly named as handcrafted features.
- *Representation Learning* is the method in which the machine learns the representations from the data itself. In this way, the machine learns connections not only between the data and its representation but also between the representation and the output.
- *Deep Learning* (DL) method, like *representation learning*, can derive representations from the data. Moreover, this method is also capable of deriving representations of the representations, which makes it *deep*.

DL method being the most recent among these, is more complex. However, this complexity enables solving more difficult tasks with higher accuracies, meaning more useful results. Figure 2-1 shows the AI methods side-by-side in order to compare them in an easier way.

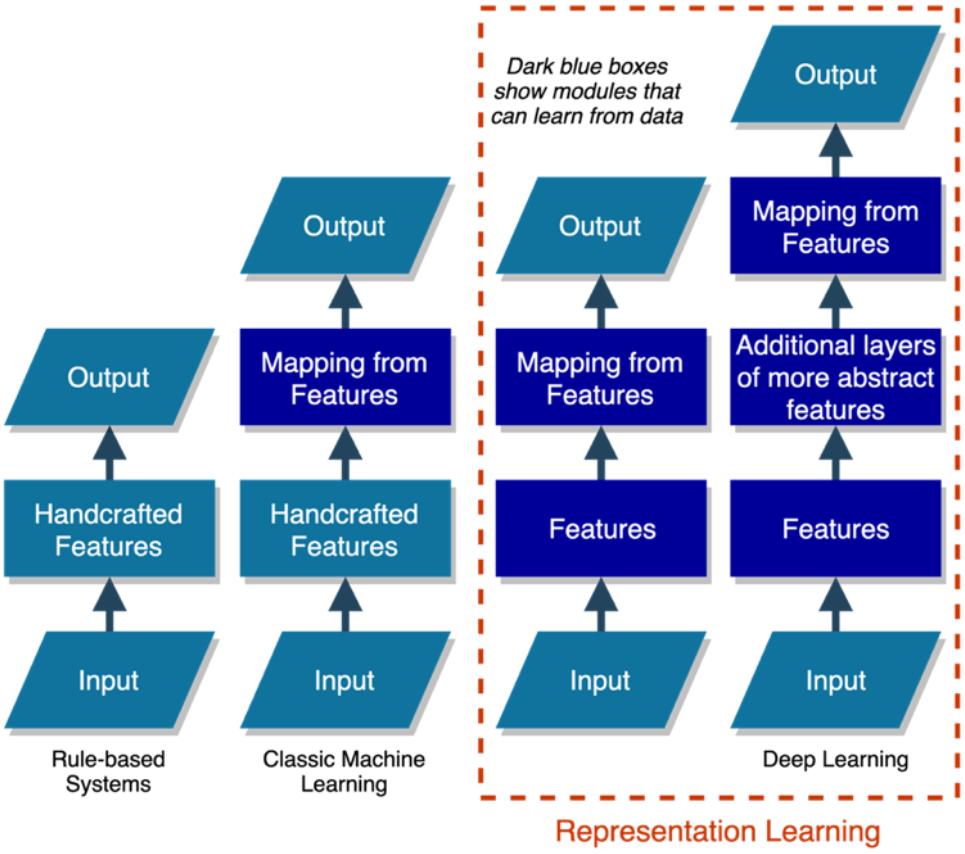


Figure 2-1. Rule-based systems, ML, representation learning, and DL methods are shown. Figure adapted from Goodfellow et al. (2016).

As Figure 2-1 represents the basic differences of these approaches, Figure 2-2 demonstrates the relations between them. It can be seen AI is the broad discipline covering all rule-based systems, ML, representation learning, and DL.

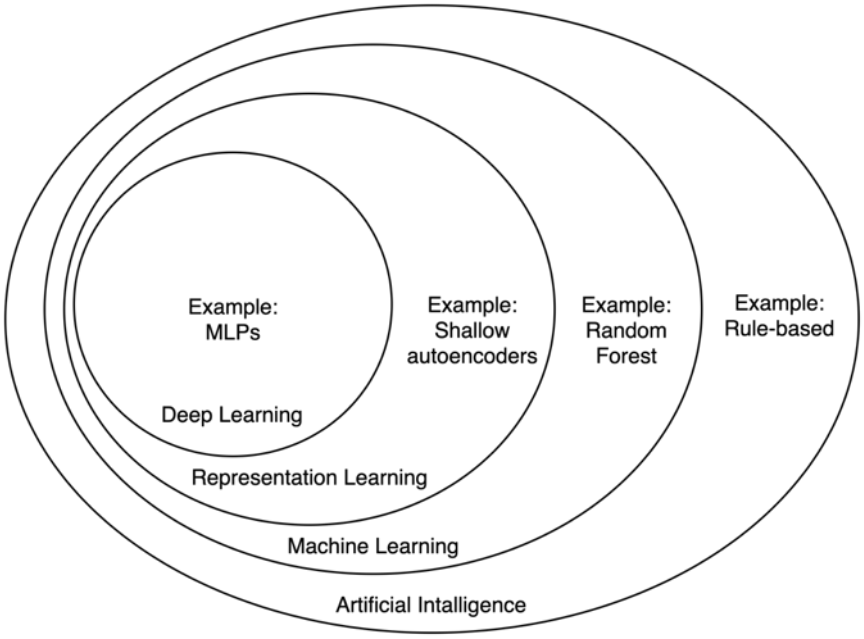


Figure 2-2. Relationship between AI approaches with example algorithms. Adapted from Goodfellow et al. (2016).

To clarify the relationships between AI approaches and better understand them, the example algorithms of random forest, autoencoders, and multi-layer perceptron (MLP) are briefly explained below:

- *Random Forest* (RF) is one of the most popular ML algorithms. The idea behind this algorithm is to utilize multiple decision trees on a randomly generated subspace of the given data.
- *Autoencoder* is a kind of neural network capable of deriving representation (also known as code or feature) from the given input data. This kind of network consists of two functions: encoder and decoder. The encoder function learns the representation generation from the data, while the decoder learns how to create the original data from these representations. This process of mapping is also

called reconstruction. The autoencoders include a hidden layer between the input and output layers, which contains the description of the representation. Earlier applications of autoencoders focused on data reduction and feature learning, while recent studies focused on generative modeling.

- *Multi-Layer Perceptron* is one of the most typical examples of DL. The objective of an MLP is to approximate a mathematical function. Such networks structures do not have any connection from their outputs back to themselves (known as feedback connections), and therefore, MLPs are also called *feedforward neural networks*.

For further reading on autoencoders, MLPs, and DL in general, readers may refer to a DL book such as Goodfellow et al. (2016), and for RF and other topics in ML to a book such as Burkov (2019).

DL is a more recent and popular research field (Figure 2-3), it is supported by researchers with various backgrounds (i.e., engineers, mathematicians, computer scientists, architects, and so on).

Figure 2-3. Interest over time for support vector machine (blue) and convolutional neural networks (red) between January 2004 and June 2021 (Google Trends, 2021).

Typical pipelines for geospatial point cloud classification with DL and ML can be summarized in Figure 2-4. As it can be seen, DNN usage eliminates the need for handcrafted feature extraction, while this is not the case for ML.

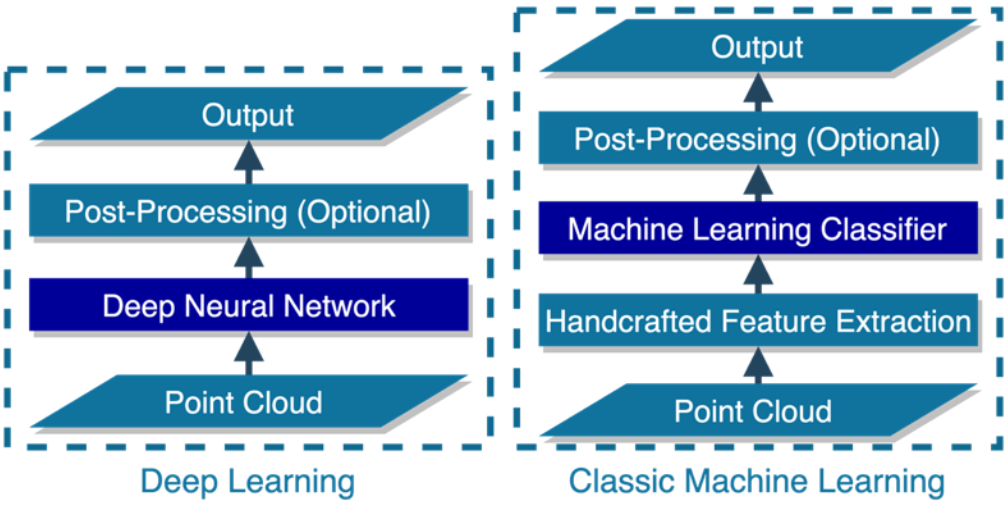


Figure 2-4. Pipelines for DL and ML approaches for geospatial point cloud classification. Dark blue boxes represent modules that can learn.

Since this framework (Chapter 3) can be considered between ML and DL, in the following subsections, the studies in the literature with ML and DL techniques for geospatial point cloud classification will be reviewed.

2.2. Geospatial Point Cloud Classification with Classic Machine Learning

ML methods for geospatial point cloud classification focus on labeling each point individually by processing their features. These features, which are defined by the expert who handcrafts them, are extracted for each one of the points individually. Handcrafted

features typically describe local (i.e., planarity, linearity) or global geometry (i.e., height above ground level, surface normal).

On the one hand, handcrafting features bring a few considerable advantages for point cloud classification with ML:

(i) The expert knowledge can be reflected in the designed approach, which is expected to ensure desired outcomes while making use of the expertise;

(ii) The handcrafted features are case-specific. Therefore, the number of features does not need to be too high (compared to representation learning methods) in most cases;

(iii) As a result of using fewer features, commonly, algorithms used for the task do not need to be very complex. Thus, short processing times are expected.

On the other hand, handcrafting features may also bring some limitations:

(i) Having fewer features can lead to poor accuracy in complex tasks and complex scenes. One cannot formulate too many features (especially compared to DL methods);

(ii) Case-specific feature handcrafting may lead to poor generalization capability;

(iii) For complex problems, features derived from more abstract features (as in the DL, Figure 2-1) can be needed, which are not feasible to be handcrafted.

In many cases, the computational efficiency of the ML algorithms makes it preferable, and some problems can be solved with high accuracy as well (Matrone et al.,

2020). However, as Heipke and Rottensteiner (2020) state, there is a limit to the extent of feature handcrafting, and therefore, how good the results can be.

ML has been a popular research domain with various research studies carried on with different perspectives. Weinmann et al. (2013) analyzed impact of the features in the classification of terrestrial laser scanning (TLS) data. Their findings indicate few and sophisticated features can achieve better performance measures compared to the use of an increased number of features. The performance measures do not only include memory consumption and computational efficiency but also accuracy. This is proving the importance of expert knowledge in handcrafting features as mentioned above. Hackel et al. (2016) suggest point density variations cause difficulties in terms of describing local geometry as well as computational efficiency. Therefore, they present a TLS classification method that can handle varying point densities. Their approach is based on downsampling the entire point cloud and extracting features with a multi-level pyramid approach. They implemented the nearest neighbor search for each level of the pyramid separately. In this way, although the retrieved neighbor points are not exactly within the same radius for each point, it is stated to be efficient for feature extraction. Thomas et al. (2018) focused on utilizing multiscale spherical neighborhoods for indoor and outdoor LiDAR point clouds. The spherical neighborhood, in comparison to the alternative k-nearest neighbors (knn), has the advantage of ensuring the retrieved neighboring points to be within a given radius. In contrast, the knn approach delivers a fixed number of points without any restrictions on the distance. Based on this advantage, the authors indicate their classification framework -using an RF classifier- can achieve accuracies such as 62% mean intersection-over-union

(mIoU) on the Paris-Lille-3D dataset. Pârvu et al. (2020) implemented a region growing algorithm for the neighborhood retrieval of the feature extraction step. Their proposed method utilizes region growing for each point in the given point cloud, and the extracted segment is then used for computing the handcrafted features for representing the local geometry. The proposed method is demonstrated to improve the classification accuracy for ground and vegetation classes. Zhang et al. (2013) developed a classification method that combines surface growing along with support vector machine (SVM) for LiDAR point cloud classification. Their approach uses an object-based approach, which utilizes segments (in other words, clusters) derived with surface growing as objects. Features are extracted at a segment level, then, the SVM classifier is used for the classification of these segments. The authors also implemented connected components-based refinement to cope with noises in classification results. Vosselman et al. (2017) proposed a method implementing multiple segmentation methods which allow context-based classification. The proposed segmentation approach includes planar segmentation using Hough transform (Illingworth and Kittler, 1988) as well as feature-based segmentation. The eliminated points, which do not fit within the segmentation parameters during segmentation, are then assigned to a class based on their neighborhood. Isolated points with no neighbors can be left as unclassified. Li et al. (2019) applied label smoothing as a post-processing step in order to improve classification results. Taking advantage of the graph-structured framework by Landrieu et al. (2017), the authors initially generate an optimal graph using the points' coordinates. Afterward, they apply probabilistic label relaxation to improve the consistency of the labels. In the final step, the data produced in the previous steps are used

as inputs for a graph-structured regularization, which forms the ultimate labels for the points.

As seen, the studies in the literature focus on different challenges such as features to extract (finding more related features), the scale of the local neighborhood for feature extraction (multi-scale versus single scale), alternative ways to retrieve nearest points (search radius versus knn), and post-processing of the classification results. Although some of these studies present their methodologies on terrestrial datasets, they indicate common issues which are also valid on airborne point clouds: feature selection, neighborhood selection for feature extraction, and multi-scale feature extraction.

The current challenges and limitations of ML methods for geospatial point cloud classification include feature engineering (especially for irregular/noisy point clouds as the local geometry variety is high within the point cloud), solving complex problems (i.e., more classes), being able to exploit more data, generalization, and transfer learning adaptation. These challenges and limitations are considered to be rooted in the nature of ML methods, as the algorithms are typically less complex (i.e., decision trees versus MLPs) than DL methods.

2.3. Geospatial Point Cloud Classification with Deep Learning

As Wang et al. (2020) indicates, DL is becoming increasingly popular for different data processing necessities. The two major reasons reported are the recent developments in technology and the increase in the available datasets. The recent developments in technology enabling realizing *deeper* (i.e., more layers) and *wider* (layers with more

parameters/filters) neural networks, while accessing more data for training and validation helps to exploit these algorithms. Unlike ML methods, which are incapable of handling (Wang, 2015). Being able to learn -the representation- from input data (Figure 2-1) and achieve better results with better generalization ability (Wang et al., 2020), DL methods are the current state-of-the-art in many applications, including geospatial point cloud classification.

Point cloud classification methods have been developed with different approaches based on convolutional neural networks (CNN), graph recursive neural networks (RNN), point convolutions, point-wise MLPs, and so on (Guo et al., 2020). In this section, the studies in the current state-of-the-art about geospatial point cloud classification with DL will be detailed.

Geospatial point cloud classification has been studied by many researchers with varying approaches. Charles et al. (2017) proposed PointNet, which consumes 3D point clouds directly (*nx3* input, where *n* is the number of points), and their proposed method is capable of semantic segmentation. The proposed network utilizes several MLP networks to extract global and local features from the given point set. Qi et al. (2017) developed PointNet++ based on the previously mentioned PointNet. The network processes the point cloud with varying neighborhoods for learning the local geometry better. The major difference between PointNet and PointNet++ can be seen as the varying neighborhood implementation for feature extraction. PointNet++, with this improvement, increased the classification performance and has been a benchmark network since then. Yousefhussien et al. (2018) developed a 1D-CNN-based method that can learn global and local geometric

features from a given point cloud for classification. Their method implements a multi-scale approach. The network is capable of consuming the available spectral data (i.e., color data from photogrammetric point clouds, or intensity LiDAR features) in addition to the 3D coordinates. Özdemir et al. (2019a) proposed a method combining handcrafted features with DL. In the proposed method, handcrafted features (namely covariance features and height above ground using DEM, which are extracted within the framework) are used as supplementary data along with the 3D point coordinates to boost the classification process. Their results indicate that ML methods are not capable of exploiting all the additional information, while DL is. The shared results also indicate the inclusion of handcrafted features enables designing a *shallower* DNN, which allows faster computations. Li et al. (2020a) developed a geometry-attentional DNN for airborne laser scanning (ALS). The proposed method is based on geometry-aware convolutions with a dense hierarchical architecture, including elevation-attention. Geometry-awareness can be described as the extraction of the local pattern for a given point neighborhood using convolutional layers as well as MLPs. The dense hierarchical architecture is based on implementing several skip connections (the output of one layer is passed as the input of at least two layers: the next layer and another further layer) between downsampling and upsampling blocks. Elevation-attention is handled by feeding an MLP structure with the input points' *z* coordinates as a vector. Wen et al. (2021) developed a graph-attention-based approach that includes graph local and graph global attentions. The graph local attention consists of edge and density attentions using MLPs. Similarly, graph global attention also uses MLPs. However, the graph here takes advantage of the Euclidean distances between every point, which is not

taken into consideration by other methods, according to the authors. Huang et al. (2020) developed their DL method based on PointNet++. Their proposed approach includes hierarchical data augmentation, which is not implemented in the original PointNet++ framework. The framework uses a nonlinear manifold-based joint learning approach and removes redundant and disruptive information. The learned hierarchical deep features are globally optimized and embedded into a low-dimensional space. In order to achieve global optimization of the initial classification results, a graph-structured optimization based on the Markov random fields approach is used. Li et al. (2020b) proposed Dance-Net, which introduces a density-aware convolution module that approximates typical convolutions on irregular 3D point clouds. The proposed density-aware convolution module reweights the learnable weights of the convolution kernels based on the point-wise density. The module approximates to a continuous convolution and is implemented in downsampling and upsampling blocks, also called as the multi-scale approach. Winiwarter et al. (2019) developed their method based on PointNet++ implementing a batching framework, which enables processing larger (i.e., geospatial) point clouds. Another structural modification they implemented is the ability to include additional features of the point cloud (i.e., colors, LiDAR features) in computation. Chen et al. (2021) proposed other modifications to PointNet++. The first proposed modification is about the treatment of the local neighborhood points. The authors suggest the way the original network handles the local neighborhood ignores the centroids of the local neighborhoods at different scales. In this way, the authors report, irrelevant information may be learned by the network. Therefore, they propose an alternative approach that takes centroids into consideration during the

processing of the local neighborhood points. The second proposed modification is the loss function. The authors implemented a modified version of the focal (Lin et al., 2017) loss function. The original focal loss function, in short, is designed to handle the class imbalance for the classification task by weighting the misclassified samples. The authors' modified implementation explicitly executes computations based on the number of points in each category. Another modification, the authors reported, is focused on increasing the importance of the elevation data along with distance-based interpolation. Increasing the importance of the elevation is expected to increase the ability to derive features representing the geometry better. Laupheimer et al. (2020) proposed an association-based method, where the point cloud and the mesh model are associated in order to transfer features and class labels between them. The proposed method relies on the classification of the mesh model rather than the point cloud. The feature extraction is computed on the mesh model, while LiDAR features (i.e., intensity) are retrieved from the associate points. Thomas et al. (2019) proposed Kernel Point Convolution (KPConv), which processes the point clouds without generating a voxel or other transitional representations. The proposed network uses a search radius (rather than knn) to retrieve local neighborhoods. The kernel is designed with the capability of learning the local shifts by implemented deformation to the kernel itself. The network is designed with grid subsampling and pooling layers. Grid subsampling ensures invariance against density variations while confirming positional stability of the input points, while pooling layers increase the receptive field. Zhang et al. (2020) proposed a framework for instance segmentation of LiDAR point clouds. The developed instance segmentation method relies on horizontal midpoint as well as the height

limits of an object. This information is then used for grouping the points forming candidate objects and noise removal. The candidate objects' midpoints are then compared with a 30cm threshold in order to merge them to a single object or leave them as separate objects. The authors report their method achieves higher accuracy than the state-of-the-art on a terrestrial dataset, which they introduce. Zhang et al. (2021) proposed an unsupervised instance segmentation for building extraction from airborne LiDAR point clouds. The proposed method divides building point clouds using a tree structure and computing geometric features. The decision is made whether it is a single building or multi-building instance.

Both ML and DL studies in the current state-of-the-art (on point cloud classification) deal with similar challenges, including data irregularity, density variations, and neighborhood retrieval techniques. Each study focuses on different challenges with different perspectives, trying to improve the state-of-the-art. The challenges and limitations of DL methods for geospatial point cloud classification include density variations, noise/outlier presence, availability of larger data sets, generalization, and explainability. Among these challenges, noise/outlier presence in the point cloud can also be dealt with during the generation step, where many improvements happened (Gruen, 2012; Haala, 2013). The availability of larger data sets for different purposes are also increasing in recent years (Guo et al., 2020). Generalization capabilities are observed to be increasing (Liu et al., 2019; Xie et al., 2020b), as well as studies on explainability are becoming more popular (Xie et al., 2020a).

2.4. Summary of the Chapter

In this section, an overview of artificial intelligence is given along with geospatial point cloud classification studies in the literature. It is seen that although the techniques and approaches may differ, the challenges are similar: irregular structure of the 3D point clouds, density variations within a dataset, and neighborhood definitions (local versus global, multi-scale, knn versus radius neighborhood retrieval). A summarized view of the literature is represented in Figure 2-5, which is clustered by challenges and this work is compared to the current state-of-the-art.

Another challenge in the geospatial point cloud domain is instance segmentation. The instance segmentation techniques with DL require much more data (Zhang et al., 2020) and such datasets are not available in the geospatial domain yet. For this reason, urban-scale studies are focusing on unsupervised, geometric-analysis-based approaches for instance segmentation (Xia et al., 2021; Zhang et al., 2021).

As crucial as the abovementioned challenges are, the following two key factors could be considered as essential while designing a geospatial point cloud classification framework: computational efficiency (in terms of computational power and memory requirements) and generalization (being able to process point clouds acquired with different sensors and at different densities) capability. These two key factors can be critical for implementations and deployment in large-scale applications, such as the daily procedures of mapping agencies.

In the upcoming chapter, the proposed method will be explained in detail, which is aims to cope with all these challenges.

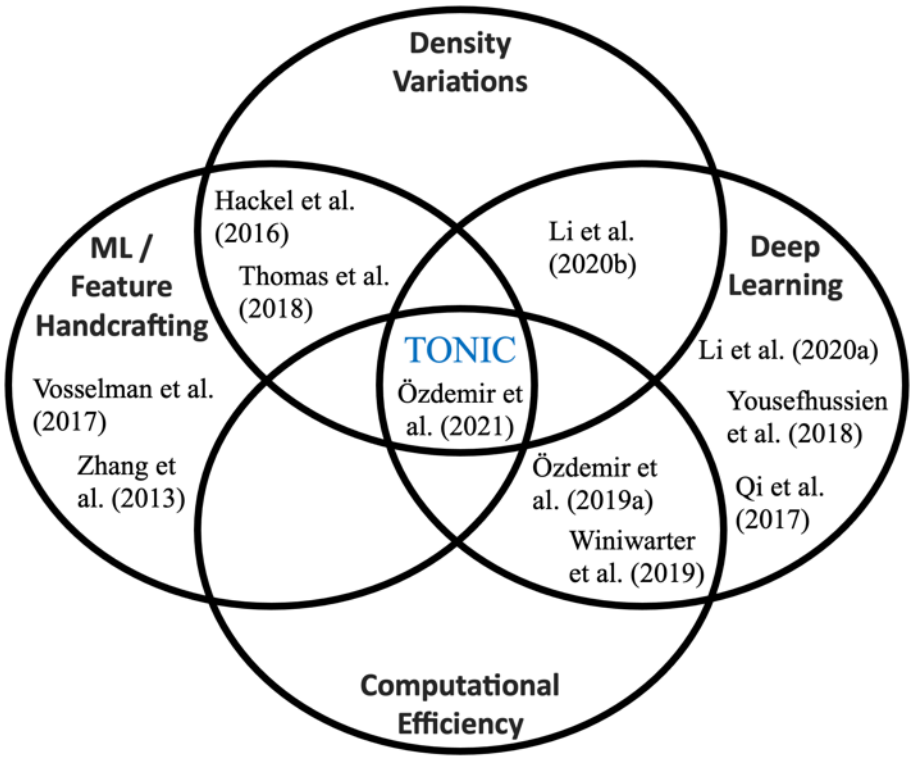


Figure 2-5. A brief look to the literature. TONIC: efficienT classification Of urbaN poInt Clouds framework (Özdemir et al., 2021).

Chapter 3.

Proposed Framework

In this section, first, a brief introduction to the framework will be given. This will be followed by detailed discussions for each step of this framework, including downsampling (Section 3.1), feature extraction (Section 3.2), and classification with DL using image representation (Section 3.3). The overall structure of the framework is shown in Figure 3-1. Besides these main framework steps, instance segmentation for buildings will also be introduced in Section 3.4.

The framework's design started with previous works (Özdemir and Remondino, 2019; Özdemir et al., 2019a), setting the foundations of it. This last iteration is focused on increasing the performance and reliability, computational efficiency, and enhancing generalization capabilities, which are further discussed in the next chapters (Chapters 4 and 5). The feature extraction approach and deep neural network design are discussed in detail in this section.

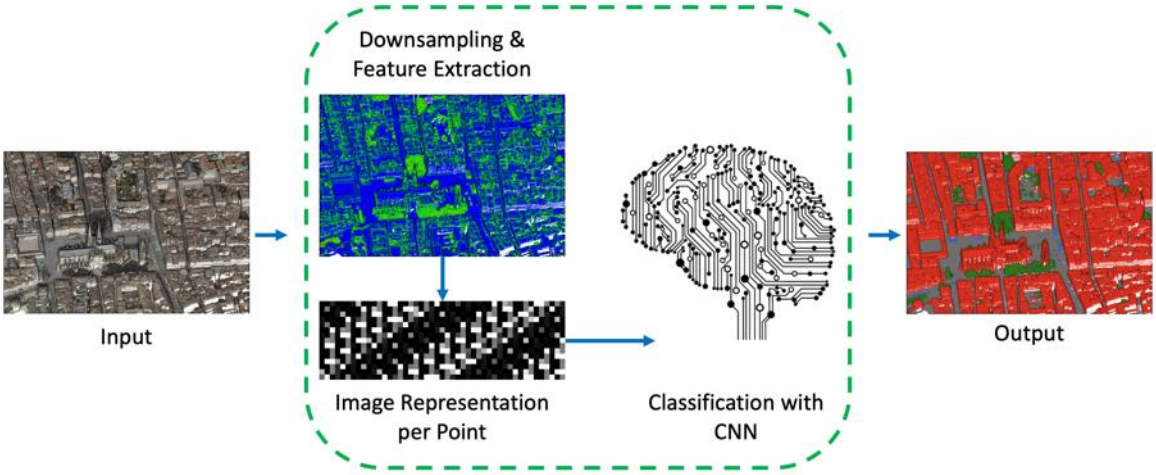


Figure 3-1. TONIC framework.

The framework receives 3D point clouds and outputs class labels per-point. If these point clouds include any radiometric data (i.e., RGB color) or LiDAR features (i.e., intensity, number of returns) these data are exploited, as well.

During development and testing, used open-source libraries are used including C++14 Standard Library (ISO, 2014), libLAS (Butler, 2021), Point Cloud Library (Rusu and Cousins, 2011) for point cloud processing, TensorFlow v2.5.0 (Martín et al., 2015) with Keras (Chollet and others, 2015) for DL implementation, Pandas (McKinney, 2010; Reback et al., 2020) and NumPy (Harris et al., 2020) for data manipulation, and Scikit-Learn (Pedregosa et al., 2011) packages for ML.

3.1. Point Cloud Preprocessing with Downsampling

The overall density of a point cloud is one of its most important characteristics. The deviations in the overall density may cause by data acquisition (i.e., flight height, camera

and lens setup, LiDAR sensor, and so on) as well as data processing options (i.e., parameter settings for dense image matching, filtering, and so on). The overall density of a point cloud can be as low as 1 pts/m^2 or as high as 1000 pts/m^2. Such overall density discrepancies hamper the classification task. For example, at ~1 pts/m^2 overall density, it is unfeasible or at least problematic to extract some high level-of-detail classes (i.e., traffic lights, cars).

Besides the overall density, density variation within a point cloud is another critical characteristic to be considered. Density within a point cloud can be as low as 1 pts/m2 and as high as 1000 pts/m2, which indicates a high density variation. Higher density variations can lead to less consistency in terms of level-of-detail in a point cloud. It also makes it hard to extract local geometry (Özdemir and Remondino, 2019).

As the local geometry (utilizing handcrafted features (Section 3.2) as well as DL (Section 3.3)) needs to be extracted, density variation has a substantial impact on this framework's performance. The handcrafted and learned features become less consistent as the density variety increases. Based on these density-related challenges, downsampling is implemented, expecting the following advantages:

(i) Downsampling reduces the total amount of data. As a result of data reduction, the entire process speeds up significantly;

(ii) Density within the point cloud to become more consistent (low density variation). Thus, the features;

(iii) Ensure that distinct point clouds have similar density characteristics, and therefore maximizing the generalization capabilities;

(iv) Reducing the noise.

Point Cloud Library's voxel-grid filter is used for the downsampling. A critical advantage of this method is that it outputs the coordinates of the centroid for the points falling into the same voxel instead of the center of the voxel (Figure 3-2). In this way, the original geometry is altered less compared to voxelation.

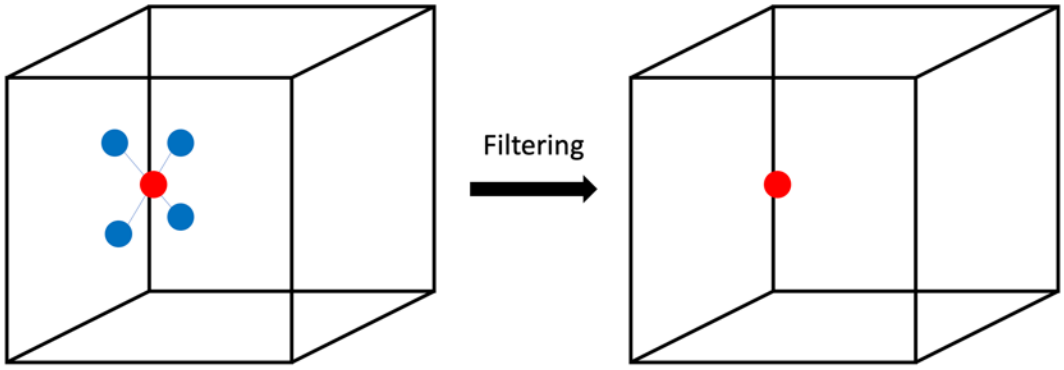


Figure 3-2. Voxel-grid filtering: the unevenly distributed points (blue) in the voxel (black cube); the centroid of the input points (red), which is the output of the filtering.

Using the voxel-grid filtering, also the noise is reduced, as output is the voxel centers instead of the points' centroid (red point in Figure 3-2). As the output of the filtering is not selected among the input points, but rather a new point is generated, the output point itself does not keep the original sensor data (i.e., colors or LiDAR features). These sensor data is then retrieved for each point from their nearest neighbors in the full resolution original input point cloud, after the filtering. The voxel size decision and its impact on the classification results will be further discussed in the next chapter (Section 4.1).

In Figure 3-3, it can be seen the original point cloud's (ISPRS Vaihingen point cloud (Niemeyer et al., 2014)) density variation is very high, while the filtered cloud is

much more homogeneous. Besides the homogeneity, there are fewer points representing the same scene preserving the objects in the scene.

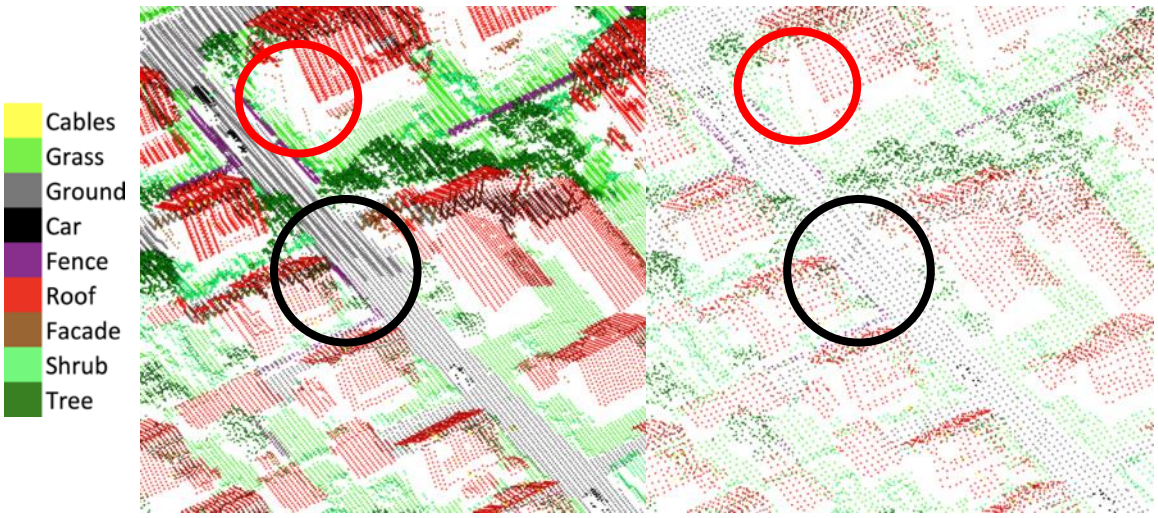


Figure 3-3. Original (left) and after (right) voxel-grid filtering. Black circles highlight the eliminated density variation, red circles highlight the overall data reduction.

3.2. Feature Extraction

The first thing to consider for handcrafted feature extraction is the neighborhood retrieval approach. There are two alternatives as mentioned before: knn and radius search. Knn method retrieves the nearest *k* number of points, which assures there will always be a constant number of points for computing the features. This assurance prevents undoable feature extraction, which happens whenever there are insufficient points (i.e., less than 3 for principal component analysis or plane fitting) in the search neighborhood. However, the problem with this approach is the coverage of the neighborhood points. Considering the density variations within a point cloud, the coverage of the nearest 10 points will vary from region to region. On the contrary, with the radius search, the points falling within a

certain search radius will be retrieved, which assures a more stable geometry representation. The problem in this case occurs:

(i) when there are very few points (i.e., 2 points) within the defined search radius (i.e., 40cm), which prevents the computation of features, or

(ii) when there are too many points (i.e., 195 points), which slows down the computations.

Due to these issues, neighborhood selection has always been a challenging decision when it comes to feature extraction, as also mentioned in Chapter 2.

As mentioned in Section 3.1, one of the advantages of the downsampling implementation is the ensured low density variation within the point cloud, which assures consistency of the geometry in the points' local neighborhood. In this way, the problem with coverage when knn used is overcome. Therefore, knn method is used for local neighborhood queries for the feature extraction. In order to extract the differences in the local geometry better, a multi-scale feature extraction approach is applied with three different scales.

So as to improve computational performance by triggering fewer search queries (of local neighborhood points), the implementation triggers the search with the largest scale only once per point. As the used knn search algorithm (of Point Cloud Library) outputs the points sorted by their distances to the query point, this information is exploited by computing the features starting from the nearest n points, followed by $2n$, and so on, as shown in Algorithm 3-1.

Extracted features include eigenvalues ($\lambda1 > \lambda2 > \lambda3$) derived from the principal component analysis (Wold et al., 1987) implementation of Point Cloud Library, eigenvectors-based surface normal estimations (the last three elements of the eigenvectors), covariance features (linearity, sphericity, omnivariance), as well as geometrically computed features (local elevation change, local planarity, vertical angle, and height above ground).

Algorithm 3-1. Multi-scale knn search implementation.

Input: 3D Point Cloud
Initialization: Not applicable.
Output: Multi-scale features per point.
1: **for** each point in the point cloud **do**
2: retrieve the highest-scale *(3n*, *n* points for *3* scales*)* local neighborhood points
retrieved points are sorted by distance by the search algorithm as default
3: **for** each scale **do**
4: compute the features with the related batch of points (*n*, *2n*, *3n*)
5: **end for**
6: **end for**

The covariance features are computed following the formulation of Hackel et al. (2016). The others are computed based on direct geometrical computations rather than principal component analysis derived eigenvalues or eigenvectors. Formulas for the used features are as shared in Table 3-1.

Table 3-1. Formulas of the handcrafted features.

Linearity	$(\lambda_1 - \lambda_2) \,/\, \lambda_1$
Sphericity	$\lambda_2 \,/\, \lambda_1$
Omnivariance	$(\lambda_1 * \lambda_2 * \lambda_3)^{1/3}$
Local Elevation Change	$z_{max} - z_{min}$
Local Planarity	$\frac{1}{n}\sum_{k=0}^{n} L_2(p_k, \vec{P})$
Vertical Angle	$\cos^{-1}(\frac{\vec{P}_{xy} \cdot \vec{p}}{(\|\|\vec{P}_{xy}\|\|) * (\|\|\vec{p}\|\|)})$
Height Above Ground	$pr_z - pl_z$

The employed non-covariance features are listed below (visualized in Figure 3-4):

(i) *Local elevation change* is the difference in the minimum and maximum z-coordinates in the neighborhood;

(ii) *Local planarity* is the mean distance between the neighboring points (p_k) to the best-fit plane ($\vec{P}$) of these points. Computed with Point Cloud Library's built-in Sample Consensus (SAC) segmentation class using Random Sample Consensus method (RANSAC, (Fischler and Bolles, 1981));

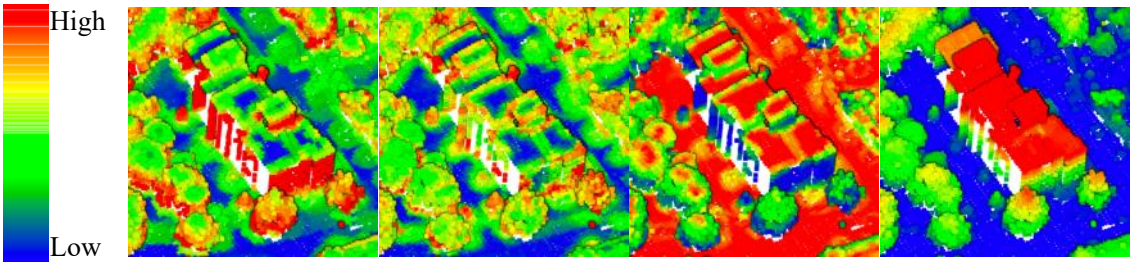
(iii) *Vertical angle* is the angle between the normal vector of a point ($\vec{p}$) and xy-plane ($\vec{P}_{xy}$);

(iv) *Height above ground* is the difference between z-coordinates of the point (pr_z) and the possible lowest point (pl_z). The possible lowest point is a hypothetical point representing the ground level and is extracted as shown in Algorithm 3-2.

The feature extraction tool is built using Point Cloud Library and written in C++ programming language. The implementation is designed with a multi-threading method allowing to utilize as many CPU threads as preferred. Therefore, the feature extraction step can be executed quickly on almost any modern hardware and operating system.

Algorithm 3-2. Identification of the possible lowest point.

Input: 3D Point Cloud
Initialization: Iterate through the input point cloud, get the lowest z-coordinate (z_{min}).
Output: Possible lowest point, to be used for calculating height above ground feature.
1: **for** each point in the cloud (*pr*: {pr_x, pr_y, pr_z}) **do**
2: generate a pseudo point (*pp:* {pr_x, pr_y, z_{min}})
3: retrieve the nearest neighboring point for *pp* from the input cloud
4: retrieve the z-coordinate of the found point (fp_z)
5: change the z-coordinate of the *pp* with fp_z
6: search for the nearest neighbor point for *pp* in the input cloud
the found point is the possible lowest point (*pl: {*pl_x, pl_y, pl_z*}*)
7: **end for**



(a) (b) (c) (d)

Figure 3-4. Local elevation change (a), local planarity (b), vertical angle (c), height above ground (d) are shown on the ISPRS Vaihingen Dataset. Colors scaled as blue-green-yellow-red, from lower to higher values.

3.3. Classification with Deep Learning and Machine Learning

The proposed classification framework (Figure 3-1) relies on the handcrafted features (Section 3.2) as well as self-learned ones. Moreover, if they are available, additional sensor data is also utilized as features (i.e., intensity, number of returns and return number for LiDAR point clouds, color information for photogrammetric ones), and coordinates of the local neighborhood points. The handcrafted feature implementation allows designing a *shallower* DNN. This is because some of the features the network would need to learn are already handcrafted and given. The framework includes both 2DCNN and a 3DCNN which are applied depending on the data and tasks reported in the discussions (Chapter 6). The proposed approach's workflow is shown in Figure 3-5.

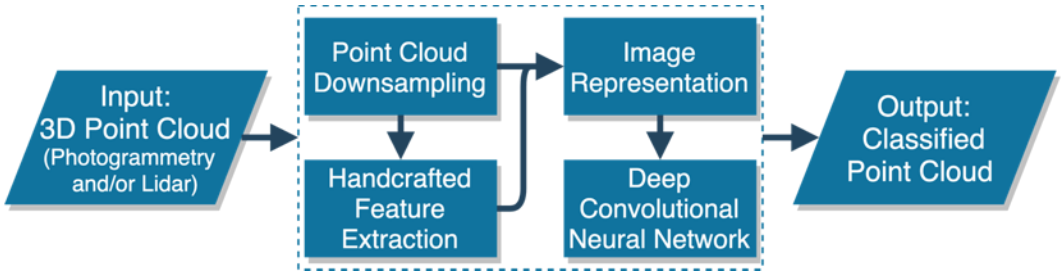


Figure 3-5. Workflow of the proposed classification framework based on DL.

As seen in Figure 3-5, in addition to the handcrafted features, the downsampled point cloud is also fed to the network inside the image representation. The image representation mentioned here is a 2D data table (a matrix) formed for each point with their neighboring points. The matrix includes the features as well as the coordinates of the local neighborhood points (Figure 3-6).

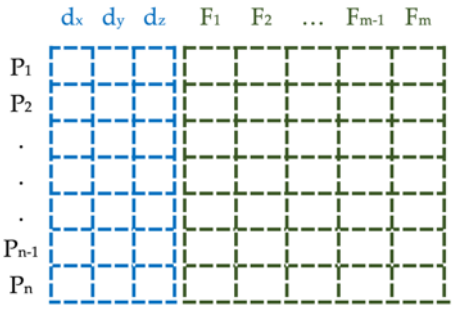


Figure 3-6. The matrix structure generated for each point: P_n denotes points, $d_{x,y,z}$ denotes matrix-wise scaled coordinates (blue cells), and F_m represents the features (green cells).

The coordinates of the local neighborhood points are centered around the point of interest and divided by maximum coordinates values in the matrix for normalization purposes. The coordinates are then clipped to the minimum and maximum values of 0 and 1, respectively. Handcrafted features and sensor data are globally normalized rather than matrix-wise. The matrix is then sorted by x- and z- coordinates respectively, which is observed to provide fractionally better results. The prepared matrix is classified with an image object classification approach by the network.

The network used for the data structure described above is based on 2D convolutions, as shown in Figure 3-7. The network receives the 2D matrices (Figure 3-6) and processes them like an image object classification, outputting the class probabilities. The network is schematized with layer parameter settings, aside from the last Dense layer that has the output dimension set to number classes per dataset.

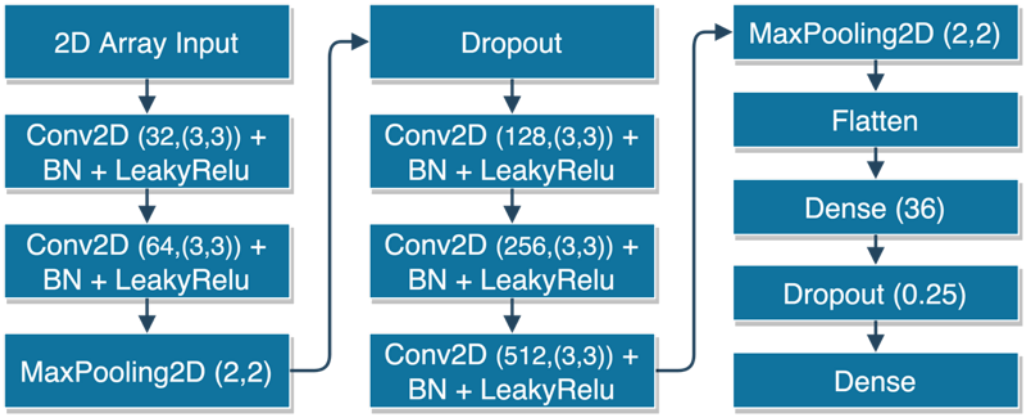


Figure 3-7. Network structure of the 2DCNN (BN: batch normalization).

Unlike rendering-based or voxel-based methods (Guo et al., 2020), TONIC's CNN methods use pseudo images, as shown in Figure 3-8.

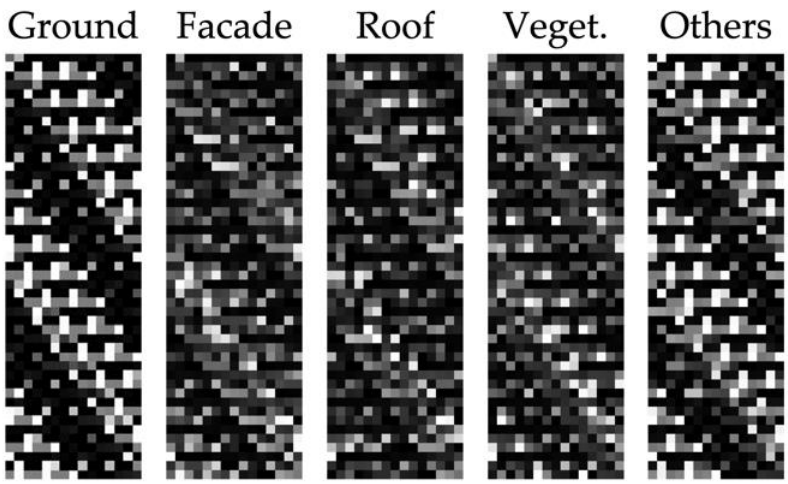


Figure 3-8. Sample matrices, rendered as images for visualization. Matrices are transposed for better illustration.

Besides the 2DCNN, a 3DCNN structure is also implemented, due to their demonstrated performances in the image processing domain. A 3DCNN model has the advantage of exploiting inter-channel correlation as well as spatial correlation (Koundinya

et al., 2018). The 2D matrices are reshaped along the features' axis (vertical axis in Figure 3-6) in order to adapt the abovementioned 2D matrices for a 3DCNN.

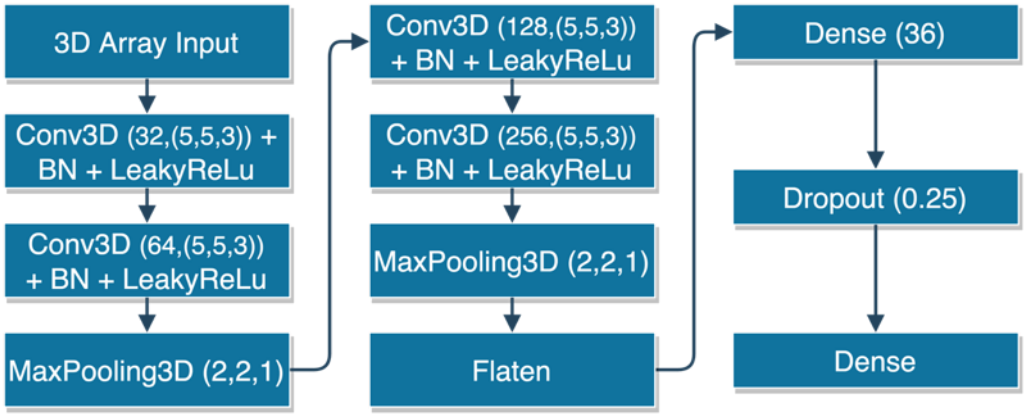


Figure 3-9. Network structure of the 3DCNN (BN: batch normalization).

This reshaping of 2D matrices produces 3D arrays (also known as tensors by the DL community), the shape of input a 3DCNN requires. In this way, a 2D matrix with dimensions of, e.g., 45x15 converted to 45x5x3. The applied 3DCNN architecture (Figure 3-9) is a slightly modified version of the 2DCNN architecture presented in Figure 3-7. Visualizing the 3D patches gives the color images shown in Figure 3-10.

An example of the classification result is shown in Figure 3-11 for qualitative representation. The results will be further discussed in Chapters 4 and 5.

In addition to the developed DL methods, a Random Forests (RF) classifier is also utilized for comparison with ML. The input for the RF classifier is the feature vector for each point, including all the features explained in Section 3.2.

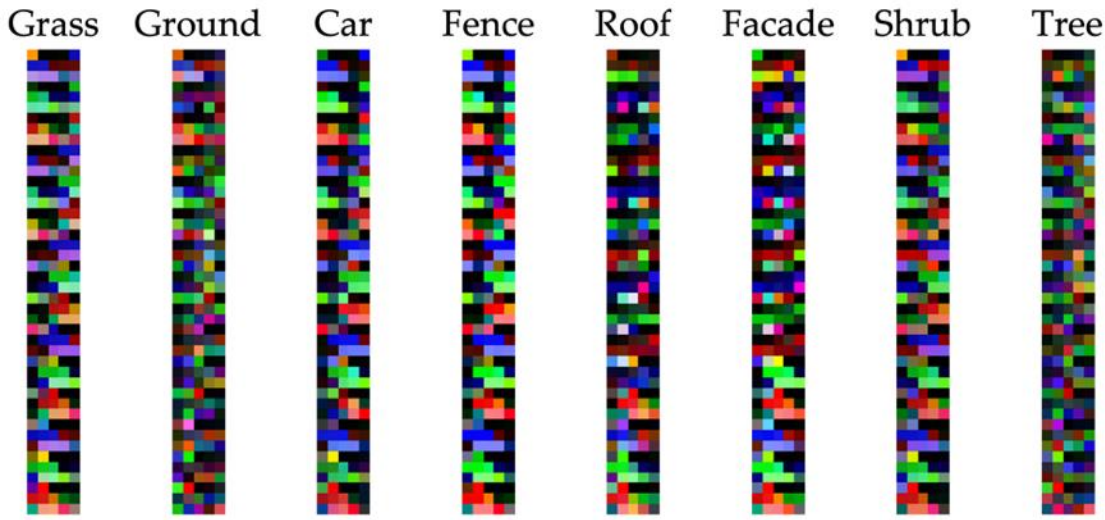


Figure 3-10. Sample tensors, rendered as color images for visualization. Transposed for better illustration.

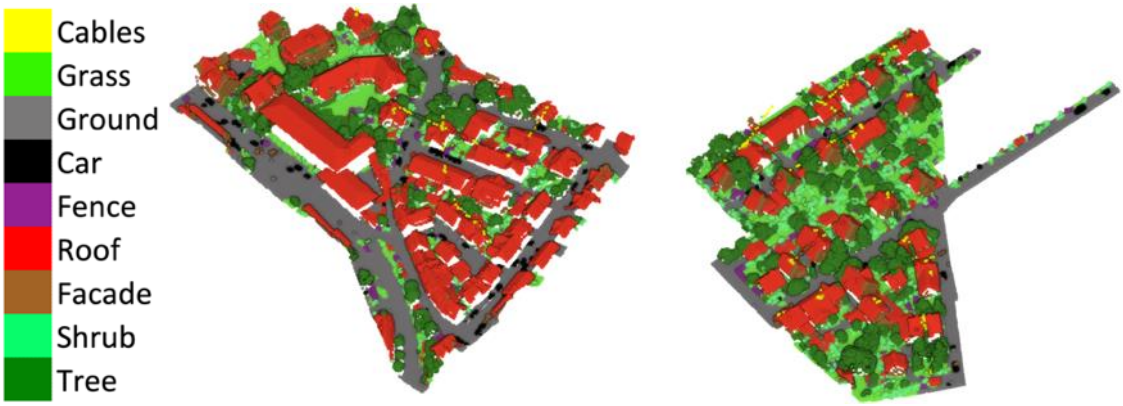


Figure 3-11. An example of classification result, 2DCNN method on the ISPRS Vaihingen dataset.

3.4. Post-Processing for Instance Segmentation of the Buildings

The classification of geospatial point clouds in urban scenes is commonly needed for 3D city or building modeling applications. However, a classified point cloud cannot be

directly used for model generation. Objects need to be separated into instances (i.e., individual buildings) for such applications. Therefore, this step is implemented as post-processing to achieve instance-segmentation results.

The instance segmentation workflow is shown in Figure 3-12. The method relies on clustered roofs instead of facades or their combination. Since the facade points may not exist in all datasets or may not be as complete as roofs (i.e., nadir only acquisitions).

Figure 3-12. Instance segmentation workflow for buildings.

As seen in Figure 3-12, the input of this step is the classified point cloud. The point cloud is first separated by the classes, which outputs a file for each class in the point cloud. This step is followed by Euclidean distance clustering of the roofs, which is applied using the built-in functions of Point Cloud Library. Clustering the roofs delivers separated buildings (building instances). The points in the class of facade (if exists) are then retrieved to the nearest roof cluster, which forms the separated buildings, as shown in Figure 3-13.

For further reading on clustering methods, readers may refer to the overview by Madhulatha (2012).

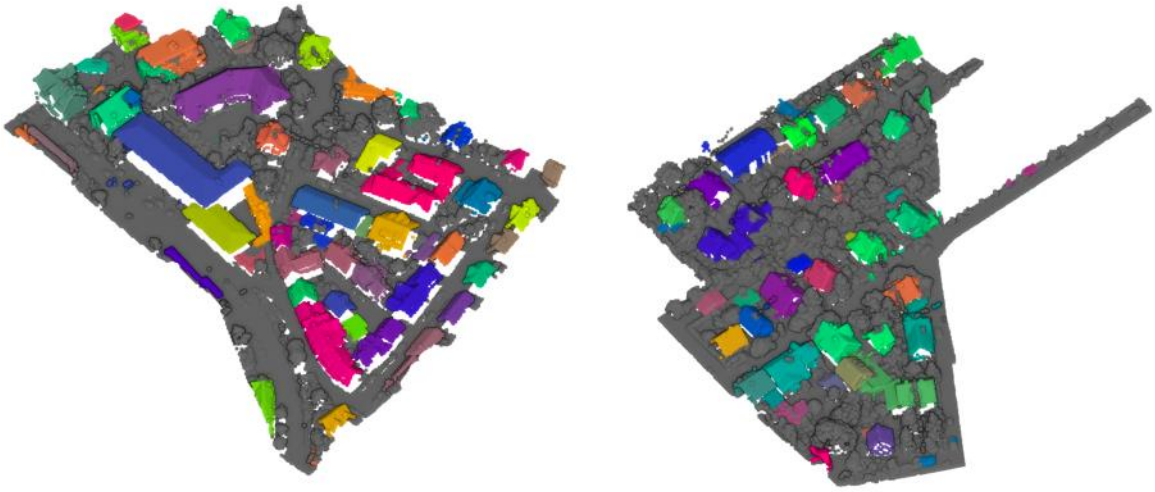


Figure 3-13. Building instances (randomly colored) and the other objects (gray), ISPRS Vaihingen evaluation dataset.

3.5. Summary of the Chapter

In this chapter, the proposed methodology is introduced step-by-step: preprocessing with downsampling, multi-scale handcrafted feature extraction, DL implementation, and post-processing for instance segmentation.

In the next chapter, the accuracy assessment methodology will be explained, datasets used for validation will be introduced, downsampling experiments will be discussed, and classification results will be reported.

Chapter 4.

Results and Accuracy Assessment

In this chapter, the following topics will be discussed: the accuracy assessment methodology, datasets used, downsampling approach and its effects, and quantitative results for the proposed TONIC classification framework.

4.1. Accuracy Assessment Methodology

Accuracy assessment of the classification results are done with the F1 score (also known as Sørensen–Dice coefficient), overall accuracy (OA), and intersection over union (IoU, also known as Jaccard index), along with weighted versions of them with the formulas shown in Table 4-1 (Verma and Aggarwal, 2020). Before these metrics, true positive (TP), true negative (TN), false positive (FP) and false negative (FN) terms are illustrated in Figure 4-1, which are the core elements to calculate the aforementioned accuracy metrics.

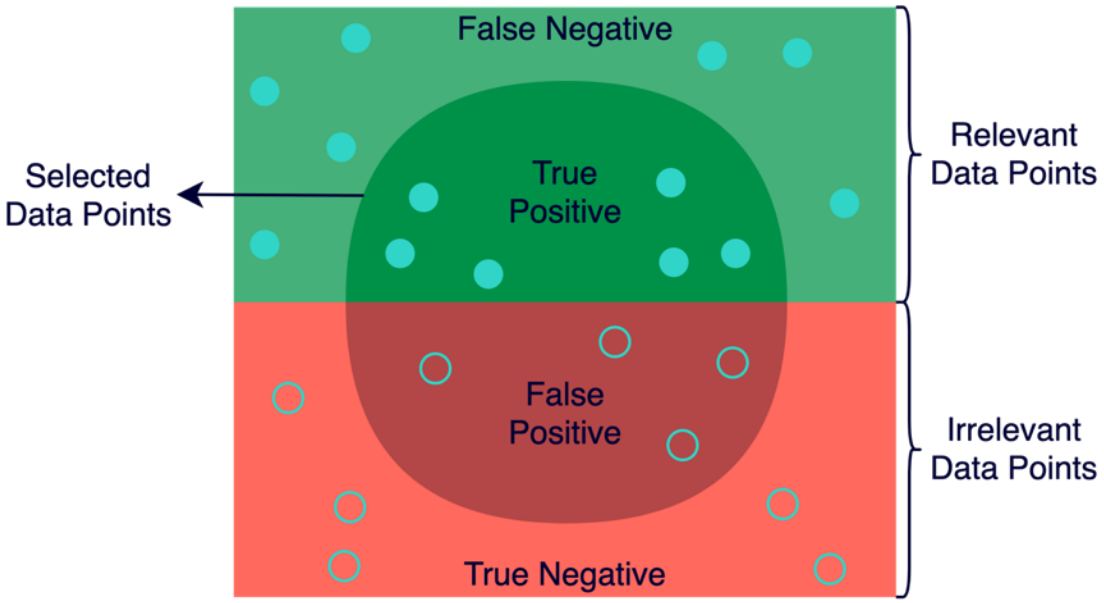


Figure 4-1. Visualization of true positive, true negative, false positive, and false negative.

As it can be seen in Figure 4-1, where the example is a binary classification (i.e., the prediction can indicate either class *true* or class *false*):

- TP is when the classification is *true*, and it is correct;
- FP is when the classification is *true*, and it is wrong;
- TN is when the classification is *false*, and it is correct;
- FN is when the classification is *false*, and it is wrong.

Among those metrics, a visual representation of the IoU can be made for better understanding, while the others are hard to visualize. This is because the IoU metric represents an area, as shown in Figure 4-2.

Table 4-1. Formulas of accuracy assessment metrics (*N*: total number of points in the point cloud; *c*: number of classes; n_i: number of points in class *i*; $F1_i$: F1 score for the class *i*; IoU_i: IoU score for class *i*).

Precision	$\frac{TP}{TP + FP}$	Recall	$\frac{TP}{TP + FN}$
F1 Score	$2 * \frac{precision * recall}{precision + recall}$	IoU	$\frac{TP}{TP + FP + FN}$
Weighted F1 Score	$\frac{1}{N}\sum_{i=0}^{c}(n_i * F1_i)$	Weighted IoU	$\frac{1}{N}\sum_{i=0}^{c}(n_i * IoU_i)$
	Overall Accuracy (OA)	$\frac{TP + TN}{TP + TN + FP + FN}$	

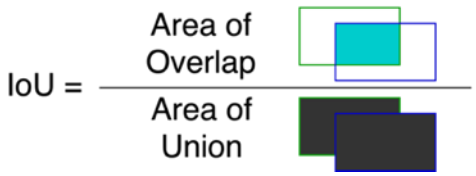


Figure 4-2. Intersection over union visualization, where the blue rectangle shows prediction and the green rectangle shows the ground truth.

As shown in Figure 4-2, IoU represents the ratio of the two areas: the overlapped area between the ground truth and the prediction, and the union of these two areas. Scikit-Learn library (Pedregosa et al., 2011) is used for calculation of all the accuracy metrics.

For the training of DL models, the F1 score is preferred as a loss function as it is commonly used for assessing the performance of a classifier. However, the F1 score is not suitable as a loss function. In fact, the F1 score is based on counted metrics (TP, TN, FP, FN), which prevents its implementation as a loss function. Therefore, an approximation to

F1 score is implemented based on the predicted probabilities, rather than counted classification results. The chosen optimizer is stochastic gradient descent (SGD) due to its performance. The patience is set to 15 epochs for early stopping, observing the validation loss. The learning rate is set to 0.001 and the training is limited to 100 epochs. Besides, in order to handle class imbalance in datasets, the training samples are weighted (i.e., reducing the weight of classes which are represented more in the dataset depending on the occurrences).

4.2. Validation Datasets

Based on the generalization, density, and data-source invariance objectives (Section 1.5), the framework is tested with five different datasets. The datasets include ISPRS Vaihingen, DALES, LASDU, Bordeaux, and 3DOMCity (Table 4-2). Except for the Bordeaux, all datasets are publicly available with ground truth labels. The differences between datasets are density, resolution, source, available sensor data, and the number of classes. The definitions of the terms density and resolution are considered as given below:

- *Density* is the average number of points per m^2 on the ground;
- *Resolution* is the average of the distances between each point and its nearest neighbor.

For further reading in benchmarking in photogrammetry and remote sensing, readers may refer to the review by Bakuła et al. (2019).

Table 4-2. Summary of the validation datasets (L: LiDAR, OP: oblique photogrammetry, Lab: laboratory, Res: Resolution, IR-R-G: Infrared-red-green)

Dataset	Source	Points	Density (pts/m2)	Res. (m)	Area (m x m)	Color	Classes
ISPRS Vaihingen (Cramer, 2010; Niemeyer et al., 2014)	L	1,165,598	4	0.258	383 x 405 + 374 x 402	IR-R-G	9
DALES (Varney et al., 2020)	L	497,632,442	35	0.116	500 x 500 (40 tiles)	No	8
LASDU (Cheng et al., 2013; Ye et al., 2020)	L	3,080,856	3	0.484	1071 x 1285	No	5
Bordeaux (Toschi et al., 2021)	L + OP	10,230,941	25	0.173	704 x 739	RGB	5
3DOMCity (Özdemir et al., 2019b)	OP (Lab)	22,825,024	14000	0.158m m	0.813 x 0.811	RGB	6

4.2.1. ISPRS 3D Semantic Labeling Contest Dataset (ISPRS Vaihingen)

ISPRS 3D Semantic Labeling Contest Dataset of Vaihingen (hereinafter ISPRS Vaihingen for the sake of readability) has been one of the most popular datasets for urban-scale geospatial point cloud classification benchmarking (Cramer, 2010; Niemeyer et al., 2014). The point cloud is acquired with the Leica ALS50 LiDAR scanner over Vaihingen, Germany. In the dataset, the training (753,876 points) and testing (411,722 points) point clouds are labeled for nine classes as follows: powerline, low vegetation (grass), impervious surface (ground), car, fence, roof, facade, shrub, and tree (Figure 4-3). The dataset includes LiDAR points, intensities, the number of returns and return numbers are provided within the classification benchmark dataset. Besides these, IR-R-G orthophotos

(infrared, red, green channels) are also provided, which we exploited in our experiments. Points per class distribution of the dataset are given in Table 4-3.

Figure 4-3. The ISPRS Vaihingen dataset, training data shown alone for better visualization.

Table 4-3. Class distribution for training and validation point clouds of the ISPRS Vaihingen dataset.

Class	**Training**		**Validation**	
	Number of Points	Percentage	Number of Points	Percentage
Cables	527	0.07%	600	0.15%
Grass	180,792	23.98%	98,690	23.97%
Ground	193,824	25.71%	101,986	24.77%
Car	4,610	0.61%	3,708	0.90%
Fence	12,081	1.60%	7,422	1.80%
Roof	152,064	20.17%	109,048	26.49%
Facade	27,192	3.61%	11,224	2.73%
Shrub	47,612	6.32%	24,818	6.03%
Tree	135,174	17.93%	54,226	13.17%
Total	753,876	100.00%	411,722	100.00%

4.2.2. DALES Dataset

Dayton Annotated LiDAR Earth Scan (DALES) dataset is a new, large-scale benchmark dataset for semantic segmentation of point clouds (Varney et al., 2020). The data acquisition is done with a Riegl Q1560 airborne laser scanner with an altitude of 1300 meters over the Surrey City in British Columbia, Canada. Being large-scale, the dataset is distributed with 500m-by-500m tiles, each containing ~12 million points, >500 million points in total. There are 29 tiles for training and 11 for testing. The point cloud is labeled for eight classes -excluding unknown- as follows ground, vegetation, car, truck, cable, fence, pole and building. The dataset includes number of returns and return numbers as LiDAR features, yet it lacks LiDAR intensity and color information. (Figure 4-4). Points per class distribution of the dataset is given in Table 4-4.

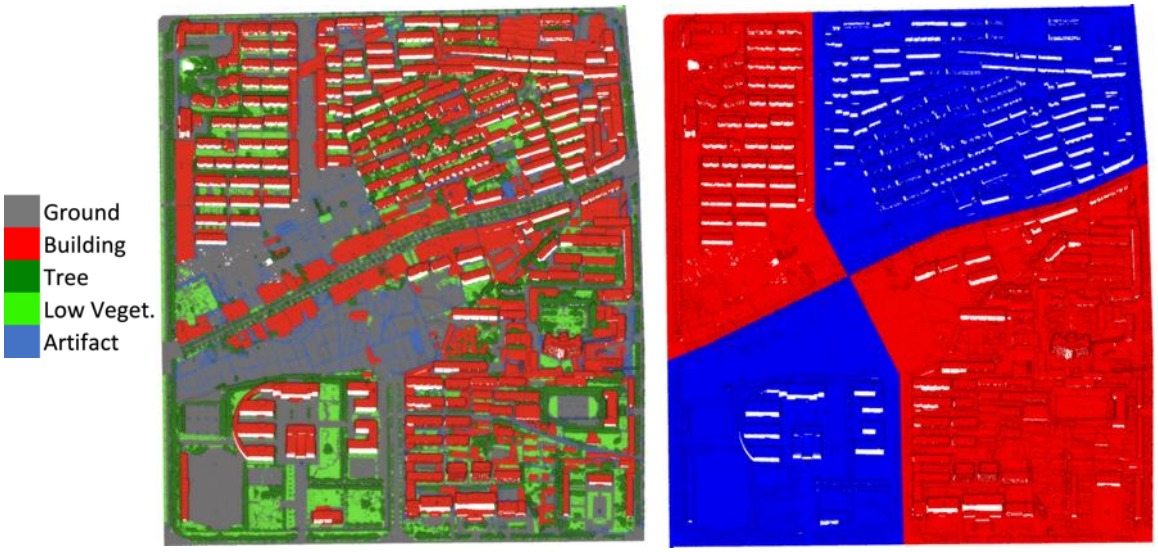
Figure 4-4. The DALES Dataset, a tile from the training set shown.

Table 4-4. Class distribution for training and validation point clouds of the DALES dataset.

Class	Training		Validation	
	Number of Points	Percentage	Number of Points	Percentage
Unknown	6,997,560	1.90%	681,571	0.50%
Ground	178,021,561	48.29%	68,871,897	50.40%
Vegetation	120,818,120	32.77%	41,464,228	30.34%
Car	2,583,281	0.70%	1,070,554	0.78%
Truck	748,890	0.20%	154,142	0.11%
Cable	799,886	0.22%	230,412	0.17%
Fence	1,512,927	0.41%	624,069	0.46%
Pole	276,924	0.08%	92,724	0.07%
Building	56,908,533	15.44%	23,454,294	17.16%
Total	368,667,682	100.00%	136,643,891	100.00%

4.2.3. LASDU Dataset

Large-Scale Aerial LiDAR Point Clouds of Highly-Dense Urban Areas (LASDU) is also a newer classification benchmark and focuses on the urban scenarios (Ye et al., 2020). The data acquisition for this dataset is made with Leica ALS70 over the Heihe River area in the northwest of China. The dataset is divided into four sections by the directions of north-south and east-west and divided equally for training and testing. The point cloud has >3 million points in total, covering >1 km^2 area. The point cloud is labeled for five classes: ground, building, tree, low vegetation, and artifact (Figure 4-5). The dataset provides LiDAR features yet, does not include orthophoto or any color information. Points per class distribution of the dataset are given in Table 4-5.



(a) (b)

Figure 4-5. The LASDU point cloud with five classes (a). The parts for training and testing, red and blue respectively (b).

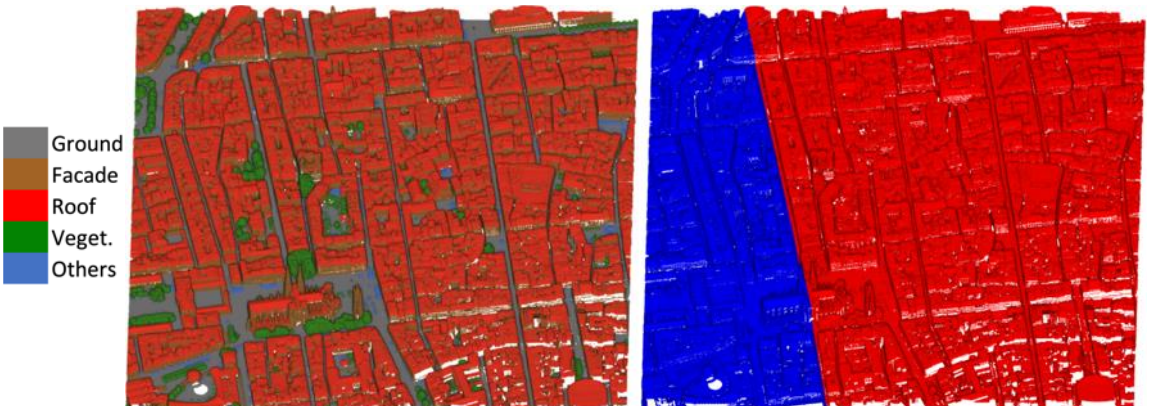
Table 4-5. Class distribution for training and validation point clouds of the LASDU dataset.

Class	**Training**		**Validation**	
	Number of Points	Percentage	Number of Points	Percentage
Ground	704,425	41.56%	637,257	45.98%
Building	508,479	30.00%	395,109	28.51%
Tree	204,775	12.08%	108,466	7.83%
Low Veget.	210,495	12.42%	192,051	13.86%
Artifact	66,738	3.94%	53,061	3.83%
Total	1,694,912	100.00%	1,385,944	100.00%

4.2.4. Bordeaux Dataset

The Bordeaux Dataset is recently produced for another study on point cloud registration refinement (Toschi et al., 2021). Two more classes are added (namely

vegetation and others) to the original classes before using the dataset. The data acquisition is made with a Leica CityMapper hybrid sensor. The sensor is *hybrid* as it contains both a Hyperion LiDAR unit and an oblique photogrammetric multi-camera system. The multi-camera system consists of one nadir and four 45° tilted cameras. The data was acquired over the City of Bordeaux in France. It covers an area of ~700m-by-700m with ~10 million points. The point cloud is labeled for five classes as follows ground, facade, roof, vegetation, and others (cars, bus stops, fences, and other artificial objects). The point cloud is separated for training and testing by 70% and 30%, respectively (Figure 4-6). The dataset includes both LiDAR features and colors. Points per class distribution of the dataset are given in Table 4-6.



(a) (b)

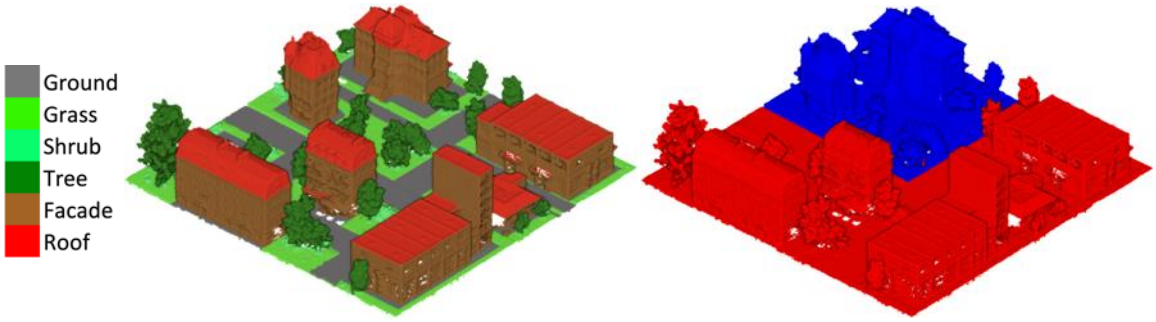
Figure 4-6. The Bordeaux point cloud with five classes (a). The parts for training and testing, red and blue respectively (b).

Table 4-6. Class distribution for training and validation point clouds of the Bordeaux dataset.

Class	Training		Validation	
	Number of Points	Percentage	Number of Points	Percentage
Ground	1,434,895	19.75%	917,446	30.94%
Facade	1,180,663	16.25%	386,237	13.03%
Roof	4,351,748	59.89%	1,445,553	48.76%
Veget.	242,121	3.33%	195,607	6.60%
Others	56,610	0.78%	20,061	0.68%
Total	7,266,037	100.00%	2,964,904	100.00%

4.2.5. 3DOMCity Dataset

3DOMCity is our initiative and includes several benchmark tasks, such as image orientation, dense image matching, and point cloud classification (Özdemir et al., 2019b). The data acquisition is made with a Nikon D750 digital camera with a 50mm focal length lens. Oblique aerial image acquisition is simulated by five different shots at camera stations for nadir (~0.124mm GSD) and oblique (~0.128-0.273mm GSD) views. Initially, only a small portion of the point cloud was labeled for ground truth use. For this study, the entire point cloud is labeled for the classification task. The point cloud is labeled for six classes: ground, grass, shrub, tree, facade, and roof. Dataset is separated ~70-30% for training and testing, respectively (Figure 4-7). Points per class distribution of the dataset are given in Table 4-7.



(a) (b)

Figure 4-7. The 3DOMCity point cloud with six classes (a). The parts for training and testing, red and blue respectively (b).

Table 4-7. Class distribution for training and validation point clouds of the Bordeaux dataset.

Class	Training		Validation	
	Number of Points	Percentage	Number of Points	Percentage
Ground	2,250,516	13.62%	682,212	10.82%
Grass	1,037,156	6.28%	800,733	12.70%
Shrub	628,953	3.81%	307,602	4.88%
Tree	3,414,501	20.67%	1,584,805	25.13%
Facade	6,635,080	40.17%	1,917,940	30.41%
Roof	2,552,885	15.45%	1,012,641	16.06%
Total	16,519,091	100.00%	6,305,933	100.00%

4.3. Point Cloud Preprocessing for Density Analysis

As mentioned in Section 3.1, downsampling the point cloud is expected to have some benefits: the reduced total amount of data for better computing speed; lower density variations for more consistent features; bringing similar overall density characteristics

among distinct point clouds for better generalization; and reduced noise. The downsampling method implemented here is voxel-grid filtering, which outputs the centroid of the points in the same voxel.

Downsampling, in case overdone, can eliminate too many points. This over-downsampling may cause two major effects: loss of detail and insufficient data for training a network. These effects can lead to low accuracy and not useful results. On the contrary, if downsampling is kept minimal, it may not be supportive for reaching the aforementioned objectives. For these reasons, experiments were held in order to get an understanding of the process and decide optimal downsampling to achieve optimal results.

The voxel dimensions are the critical parameters for voxel-grid downsampling. Here, the voxel dimensions are calculated with a leaf coefficient parameter and the original resolution of the point cloud. As mentioned before (Section 3.5), the resolution here is the average of the distances between each point and their nearest neighbors. However, the resolution is not sufficient alone to represent the density characteristics, as it does not represent the density variations. For this reason, the density characteristics are measured via minimum, mean, median, maximum, and standard deviation of the number of nearest neighboring points. The nearest neighboring points are retrieved with a radius search instead of a knn, as the goal is to measure the geometric consistency (Hermosilla et al., 2018). Influenced by the neighboring distance in the image domain (Figure 4-8), the radius is calculated as 1.45 times the resolution.

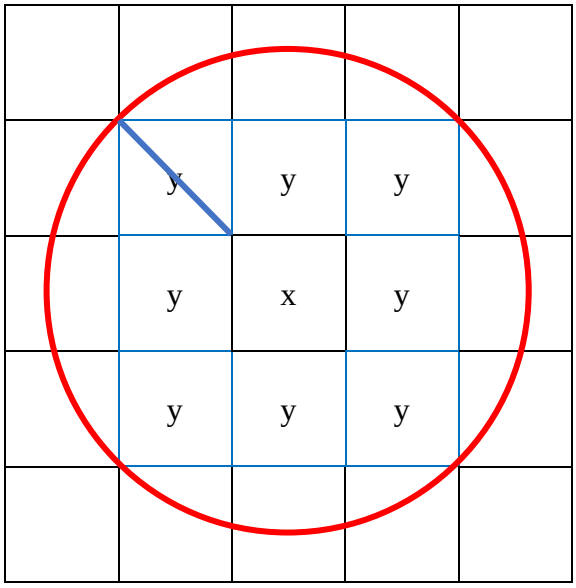


Figure 4-8. Relation between nearest neighborhood and resolution on the image. The pixels *y* are the nearest neighbors of the pixel *x*, at 1-pixel resolution. The red circle is the smallest circle to cover the nearest neighbors; the thick blue line is the distance from the border of pixel *x* to the red circle, which is $sqrt(1^2 + 1^2) \approx 1.41$ *pixels*.

As seen in Figure 4-8, from the borders of a pixel to cover its nearest neighbors, the distance is ~1.41 times the resolution. Considering the irregularities in the point cloud, this value is taken as 1.45 in order to expand the coverage by a small margin without changing it significantly. Here, the distance from the border of pixel *x* is considered, as the points in the point cloud do not have borders.

Instead of theoretically verifying the effects of downsampling via comparing statistics of the computed feature spaces (i.e., minimum, maximum, mean, median, standard deviation, or any other statistical measure), a more practical validation methodology is preferred. Therefore, in order to compare these feature spaces, the

classification results obtained using them are compared. For these experiments, the concentration is on the classification results achieved with feature spaces coming from downsampling with varying voxel dimensions. For classification, a Random Forests (Breiman, 2001) classifier is preferred because of its fast prediction abilities. Moreover, in this way, the DL methods can be compared with an ML method. The classification results are evaluated in terms of weighted F1 score and OA.

ISPRS Vaihingen, LASDU, and Bordeaux datasets are used for experimenting with the downsampling effects. The parameter in focus here is the leaf coefficient. Starting from 1, meaning no downsampling, the parameter is set to multiplies of two. The coefficient is increased till the resolution reaches ~1.0 m in order to keep the level-of-detail represented by the point cloud at a significant level for classification purposes. For this reason, and as the initial resolutions differ among distinct datasets, the number of experiments varies for each dataset.

The ISPRS Vaihingen dataset used for density analysis includes training, validation, and the rest of the tile, as shown in Figure 4-9. This decision is to include a larger area to have a more realistic analysis. It can be seen from the figure the training and validation parts are small areas compared to the tile. Therefore, the density analysis would be affected by the noise easily.

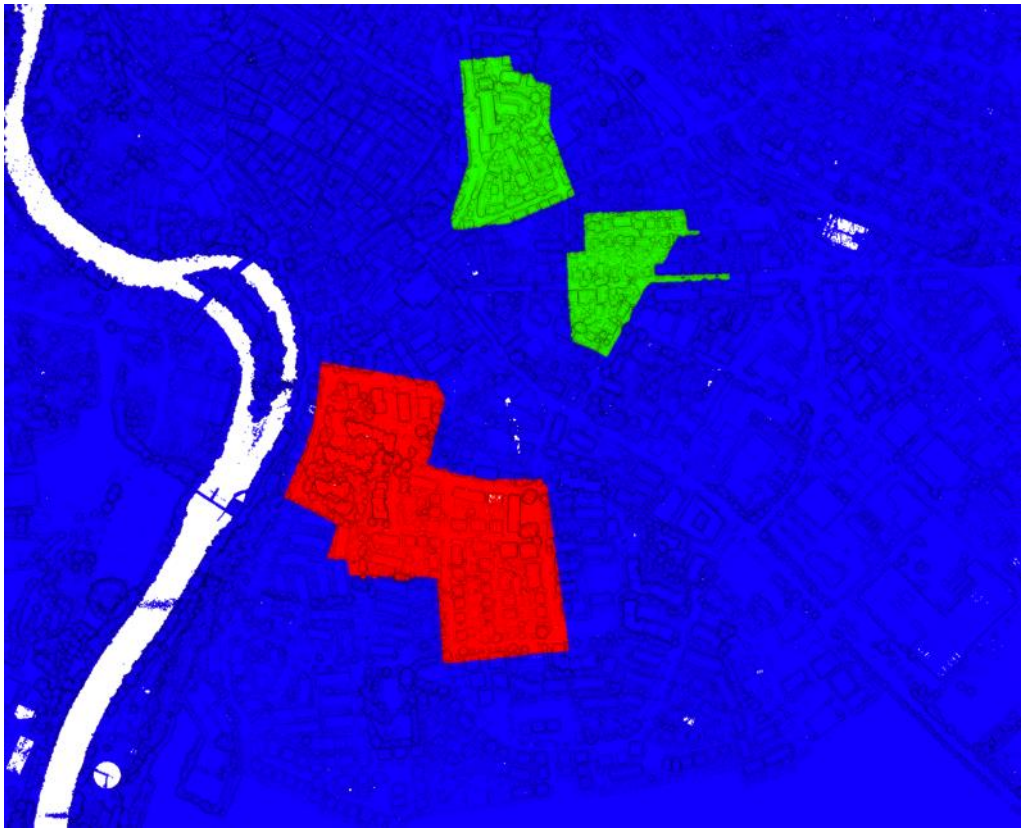


Figure 4-9. The ISPRS Vaihingen dataset as used for density analysis. Training (red), validation (green), and rest (blue) of the dataset.

The test results shown in Table 4-8 indicate the best accuracy is achieved with a leaf coefficient of 2, ending up with 0.796 OA at 0.434 m resolution. However, as it can be seen neither weighted F1 nor OA score shows a significant change between leaf coefficients 2 and 4. On the contrary, the standard deviation is minimum with leaf coefficient 4. The minimum standard deviation means more homogeneous point distribution and less density variation, especially compared to the original point cloud's 12.99s. Regarding data reduction, using leaf coefficient 4, downsampled point cloud contains one-fifth of the original while using leaf coefficient 2 has half of it.

Table 4-8. Density analysis on the ISPRS Vaihingen dataset.

Leaf coefficient	% of points	Reso. (m)	Min knn	Mean knn	Median knn	Max knn	Std. knn	Weigh. F1	OA
1	100	0.258	2	11	5	106	12.99	0.788	0.788
2	50	0.434	2	4	5	15	1.51	**0.798**	**0.796**
4	**20**	**0.732**	2	4	4	13	**1.31**	0.776	0.788
6	11	1.07	2	4	3	13	1.37	0.768	0.770

For the LASDU dataset, the downsampling is performed on the entire point cloud, and the test results are shown in Table 4-9.

Table 4-9. Density analysis on the LASDU dataset.

Leaf coefficient	% of points	Reso. (m)	Min knn	Mean knn	Median knn	Max knn	Std. knn	Weigh. F1	OA
1	100	0.484	2	4	4	83	1.45	**0.826**	0.814
2	**48**	**0.792**	2	5	4	12	**1.24**	0.823	**0.821**
4	15	1.440	2	5	5	14	1.31	0.805	0.816

As seen in Table 4-9, classification results reached with the original dataset fractionally better in terms of weighted F1 score, while it is vice-versa for the OA. Using a leaf coefficient of 2: achieves better OA, reduces the point cloud to less than half of the original, and minimizes the standard deviation.

The downsampling analysis tests for the Bordeaux dataset were also held on the entire dataset. The results are shown in Table 4-10.

As shown in Table 4-10, not only does OA improve significantly after downsampling compared to the original cloud, but the weighted F1 score improves as well. Reducing the data almost to a tenth of the original, density variation is also minimized using a leaf coefficient of 6, resulting in 0.737 m resolution.

Table 4-10. Density Analysis on the Bordeaux dataset.

Leaf coefficient	% of points	Reso. (m)	Min knn	Mean knn	Median knn	Max knn	Std. knn	Weigh. F1	OA
1	100	0.173	2	6	2	61	6.01	0.930	0.928
4	23	0.528	2	4	2	14	1.44	0.922	**0.942**
6	**12**	**0.737**	2	4	4	12	**1.31**	**0.940**	0.940
8	8	0.946	2	4	4	12	1.32	0.935	0.935

Bearing in mind the achieved results of F1 score, the standard deviation for knn, and data reduction represented in the tables above, a resolution of 0.7–0.8m can be considered the most suitable for all datasets. This can be considered as the optimal compromise balancing accuracy, density variation, and data amount. Besides, the generalization capability should be taken into consideration (Section 5.1).

One may worry about the time spent on the downsampling as it is an additional step. However, it can be seen from Table 4-11 that downsampling saves a significant amount of time during feature extraction instead of expanding it. The computations shown in Table 4-11 are held on Intel i9-8950HK Mobile CPU using 4 CPU threads. As seen, the feature extraction step alone takes more than 47 times without downsampling. The downsampling step takes ~11 seconds to complete, yet, it saves ~2858 seconds from feature extraction for a point cloud with ~10,200,000 points. As the downsampling results a point cloud without the sensor data (i.e., color or intensity). Therefore, these data are retrieved from the nearest neighboring points in the original cloud. This retrieval time is also included in the given times. Considering the original point cloud has ~8 times the points, the time differences are understandable.

Table 4-11. Feature extraction and downsampling times for the Bordeaux Dataset times are given in terms of seconds.

	Full Cloud	Downsampled Cloud	Ratio (Full / Downsampled)
Feature Extraction	2919.47	61.70	47.32
Downsampling*	10.82	-	-
# of points	10,230,941	1,264,690	8.09

*Downsampling time includes retrieval of the sensor data.

Based on these experiments, downsampling is applied to all datasets using a fixed 0.75m voxel dimension. In Table 4-12, the number of points and the ratio of kept points before and after downsampling are shown for all the datasets used.

Table 4-12. The number of points in each dataset before and after the downsampling procedure.

Dataset	ISPRS Vaihingen	DALES	LASDU	Bordeaux	3DOMCity
# of original points	1,165,598	497,632,442	3,080,856	10,230,941	22,825,024
# of downsampled points	236,603	27,652,837	1,465,068	1,264,690	2,075,937
Ratio of kept points	0.203	0.056	0.476	0.124	0.091

The ratio of eliminated points reported in Table 4-12 differs between 52-94%. This variety of ratios are due to the differences among the original point clouds' resolutions.

In order to have a correct and fair accuracy assessment, the classification outputs are projected back to the original point clouds based on nearest neighbors.

4.4. Results on Validation Datasets

In this section, the classification results of the framework on the validation datasets will be reported using the accuracy metrics mentioned in Section 4.1.

4.4.1. ISPRS Vaihingen

The accuracy metrics per class and the OA analysis are given in the tables below for quantitative assessment. For the qualitative assessment, the classification results are represented in the figure below. Instance segmentation results are then shared for the qualitative assessment without quantitative assessment due to lack of ground truth.

Table 4-13. Per-class accuracy assessment for the ISPRS Vaihingen dataset. Bold values highlight higher scores among classifiers. LV: Low Vegetation.

	Cables	**LV**	**Ground**	**Car**	**Fence**	**Roof**	**Facade**	**Shrub**	**Tree**
2DCNN - F1	0.000	**0.795**	**0.904**	**0.733**	**0.213**	**0.929**	**0.583**	**0.451**	**0.817**
3DCNN - F1	0.301	0.781	0.896	0.688	0.207	0.902	0.536	0.413	0.802
RF - F1	**0.393**	0.771	0.881	0.547	0.143	0.895	0.574	0.434	0.742
2DCNN - IoU	0.000	**0.660**	**0.825**	**0.579**	**0.119**	**0.867**	**0.411**	**0.291**	**0.690**
3DCNN - IoU	0.177	0.641	0.811	0.525	0.116	0.822	0.367	0.261	0.670
RF - IoU	**0.244**	0.627	0.787	0.376	0.077	0.810	0.403	0.277	0.590

Table 4-14. Average F1, class weighted average F1, and OA for the ISPRS Vaihingen dataset.

	Average F1	**Weigh. Av. F1**	**OA**
2DCNN - F1	**0.603**	**0.822**	**0.826**
3DCNN - F1	0.614	0.804	0.806
RF - F1	0.598	0.788	0.786
2DCNN - IoU	**0.494**	**0.719**	**0.826**
3DCNN - IoU	0.488	0.693	0.806
RF - IoU	0.466	0.670	0.786

As seen in tables above, DL classifiers achieved higher accuracies ($\gtrsim$80% goal) in terms of F1 scores and OA. The RF classifier failed to achieve 80% OA goal by 1.4%.

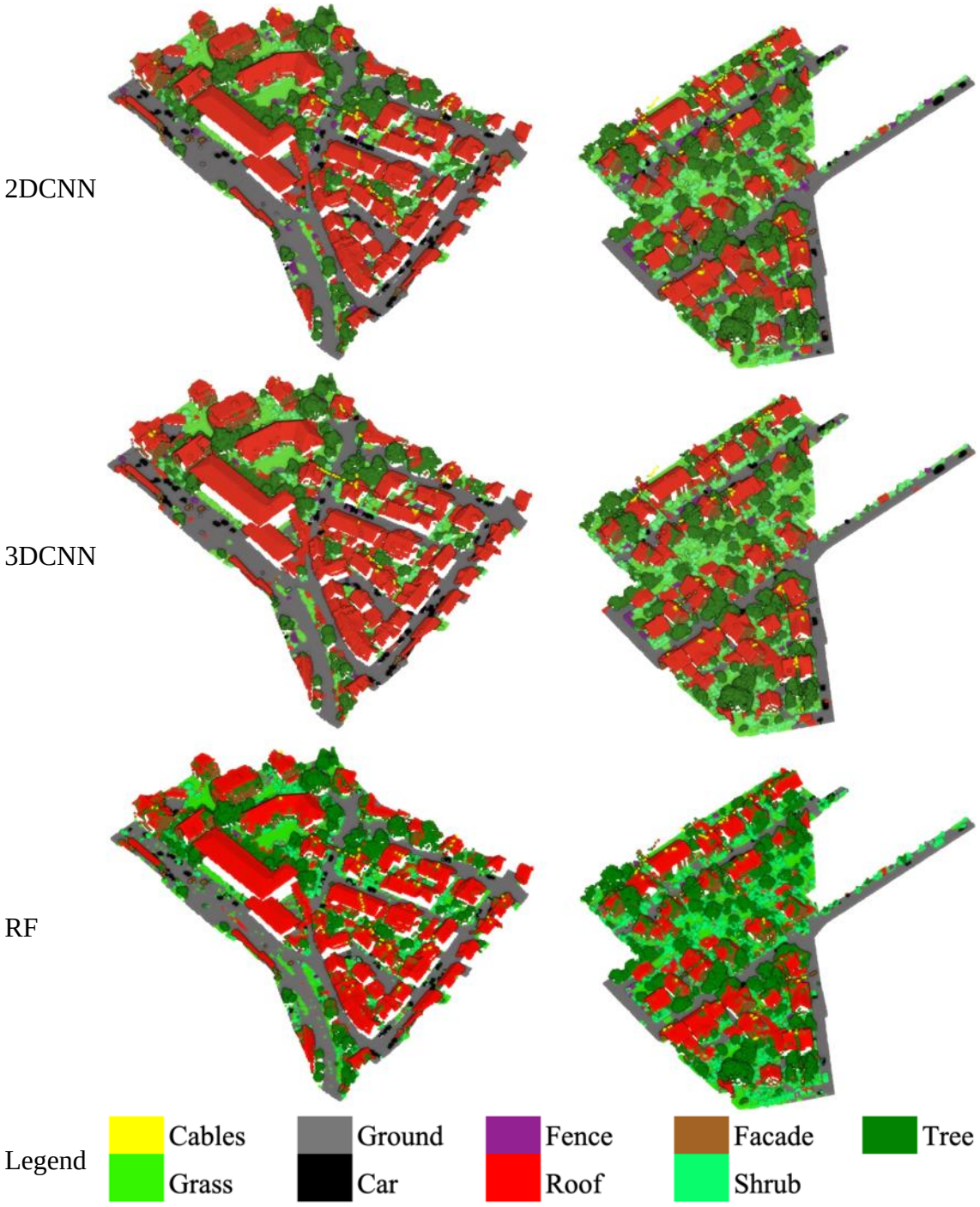


Figure 4-10. Classification results of the ISPRS Vaihingen dataset with proposed DL methods and ML method.

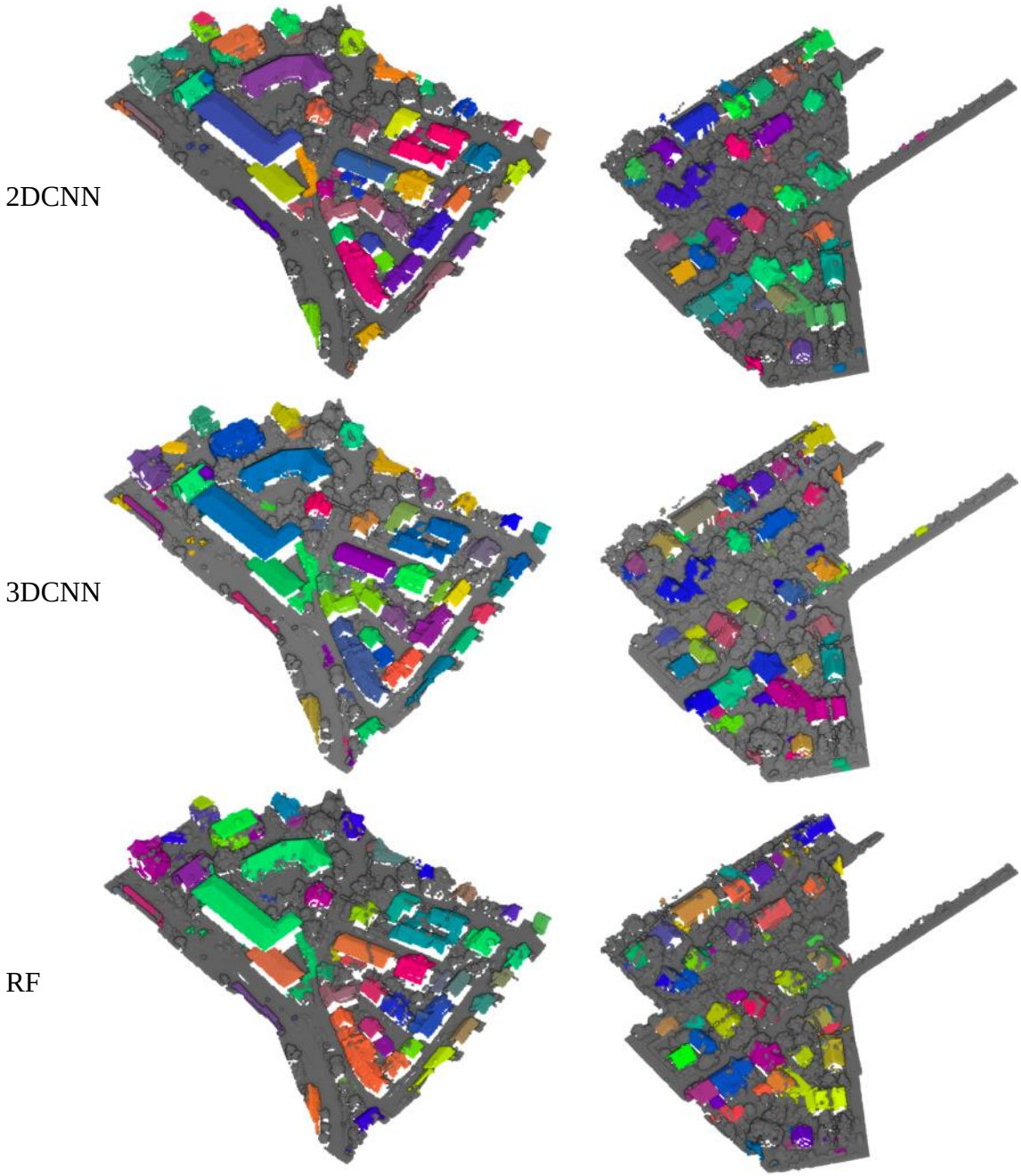


Figure 4-11. Building instances (randomly colored) and the other objects (gray), the ISPRS Vaihingen evaluation dataset.

4.4.2. DALES

The accuracy metrics per class and the OA analysis are given in the tables below for quantitative assessment. For the qualitative assessment, the classification results are represented in the figure below. Instance segmentation results are then shared for the qualitative assessment without quantitative assessment due to lack of ground truth.

Table 4-15. Per-class accuracy assessment for the DALES dataset. Bold values highlight higher scores among classifiers.

	Ground	**Veget.**	**Car**	**Truck**	**Cable**	**Fence**	**Pole**	**Building**
2DCNN - F1	**0.962**	**0.927**	0.666	0.000	0.903	**0.530**	0.468	0.911
3DCNN - F1	0.958	0.923	**0.682**	0.000	**0.914**	0.490	**0.547**	0.905
RF - F1	**0.962**	0.866	0.355	**0.042**	0.812	0.291	0.353	**0.962**
2DCNN - IoU	0.926	**0.863**	0.499	0.000	0.823	**0.360**	0.306	0.837
3DCNN - IoU	0.919	0.857	**0.517**	0.000	**0.841**	0.325	**0.377**	0.826
RF - IoU	**0.928**	0.764	0.216	**0.021**	0.684	0.170	0.214	**0.928**

Table 4-16. Average F1, class weighted average F1, and OA for the DALES dataset.

	Average	**Weigh. Av. F1**	**OA**
2DCNN - F1	0.671	**0.937**	**0.938**
3DCNN - F1	**0.677**	0.932	0.934
RF - F1	0.571	0.884	0.899
2DCNN - IoU	0.577	**0.884**	**0.938**
3DCNN - IoU	**0.583**	0.876	0.934
RF - IoU	0.474	0.843	0.899

As seen in tables above, DL classifiers achieve higher OA as in the previous dataset results. However, for this dataset, the OA gaps between the classifiers are less.

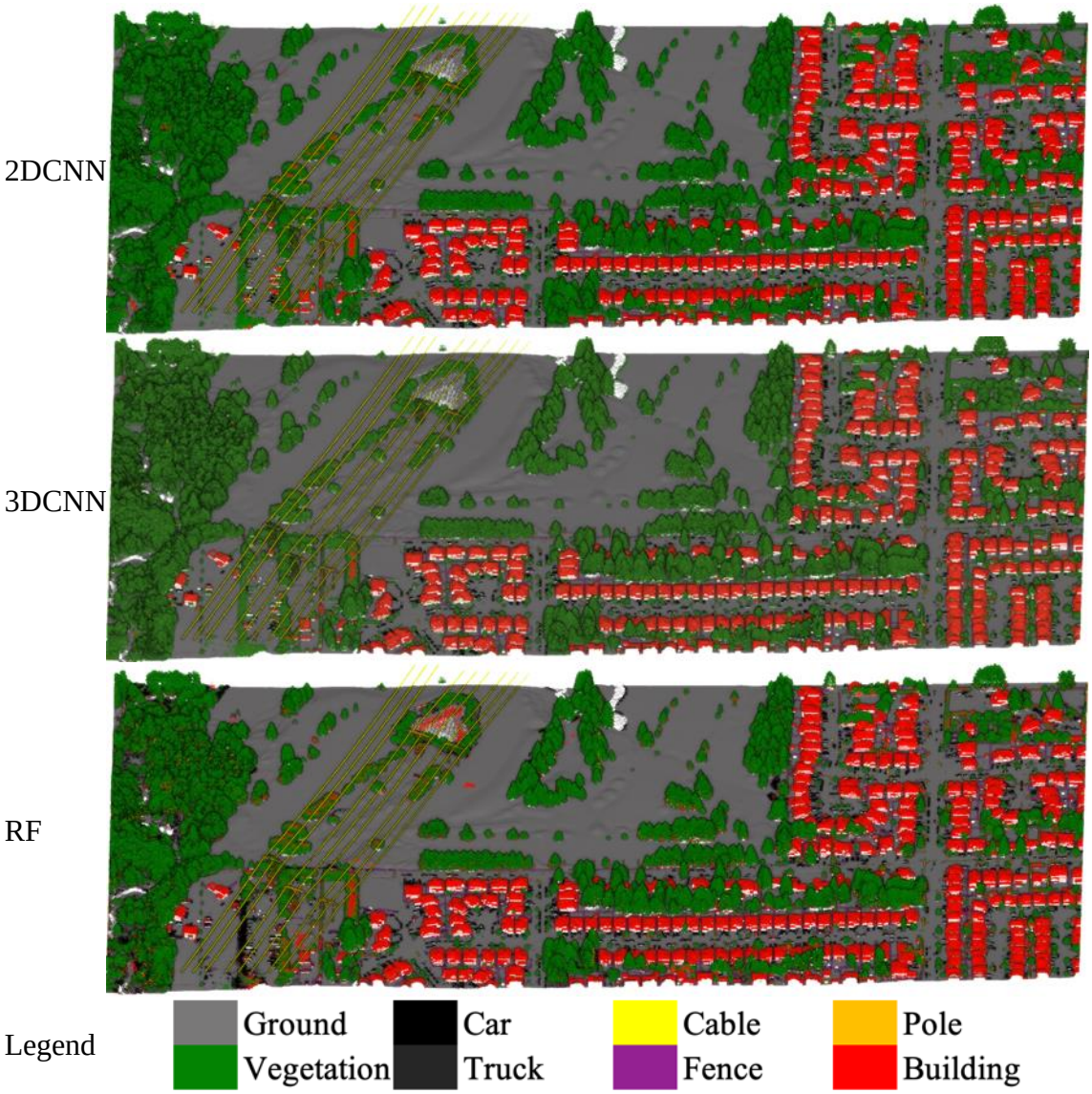


Figure 4-12. Classification results of the DALES dataset with proposed DL methods and ML method. Two of the tiles shown as samples.

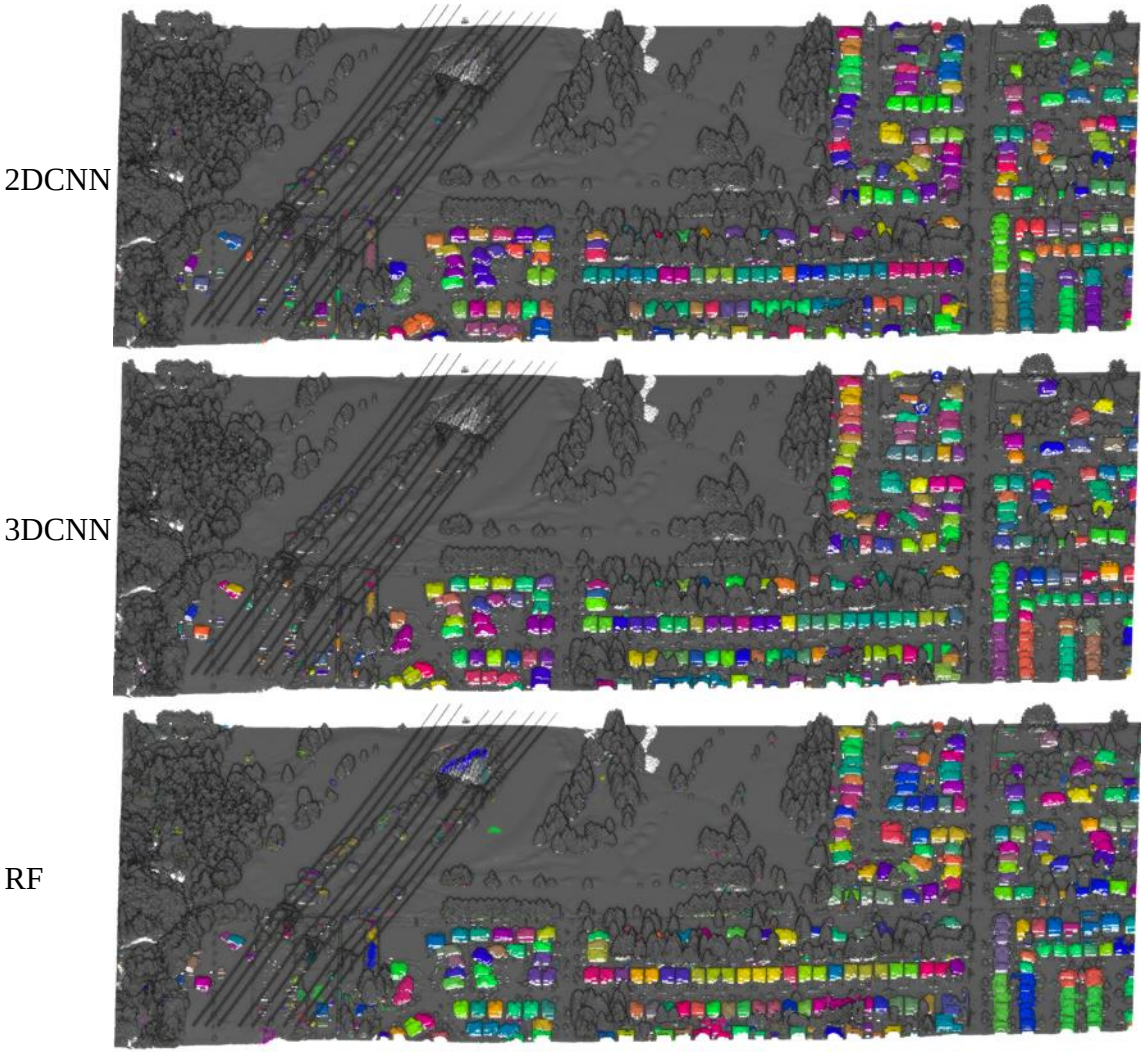


Figure 4-13. Building instances (randomly colored) and the other objects (gray). Two of the tiles shown as samples.

4.4.3. LASDU

The accuracy metrics per class and the OA analysis are given in the tables below for quantitative assessment. For the qualitative assessment, the classification results are represented in the figure below. Instance segmentation results are then shared for the qualitative assessment without quantitative assessment due to lack of ground truth.

Table 4-17. Per-class accuracy assessment for the LASDU dataset. Bold values highlight higher scores among classifiers. LV: Low Vegetation.

	Ground	Building	Tree	LV	Artifact
2DCNN - F1	**0.887**	**0.935**	**0.860**	**0.691**	**0.360**
3DCNN - F1	0.885	0.915	0.858	0.673	0.322
RF - F1	0.869	0.920	0.845	0.597	0.318
2DCNN - IoU	**0.796**	**0.878**	**0.754**	**0.527**	**0.220**
3DCNN - IoU	0.793	0.843	0.751	0.507	0.192
RF - IoU	0.768	0.852	0.731	0.426	0.189

Table 4-18. Average F1, class weighted average F1, and OA for the LASDU dataset.

	Average	Weigh. Av. F1	OA
2DCNN - F1	**0.746**	**0.851**	**0.846**
3DCNN - F1	0.730	0.840	0.837
RF - F1	0.710	0.823	0.821
2DCNN - IoU	**0.635**	**0.757**	**0.846**
3DCNN - IoU	0.617	0.741	0.837
RF - IoU	0.593	0.720	0.821

As seen in the tables above, 2DCNN achieves the highest per-class accuracies, which is also reflected in the OA. Rankings of the per-class accuracies, average F1 scores and OA show similar characteristics to the results represented for ISPR Vaihingen dataset in Table 4-13 and Table 4-14.

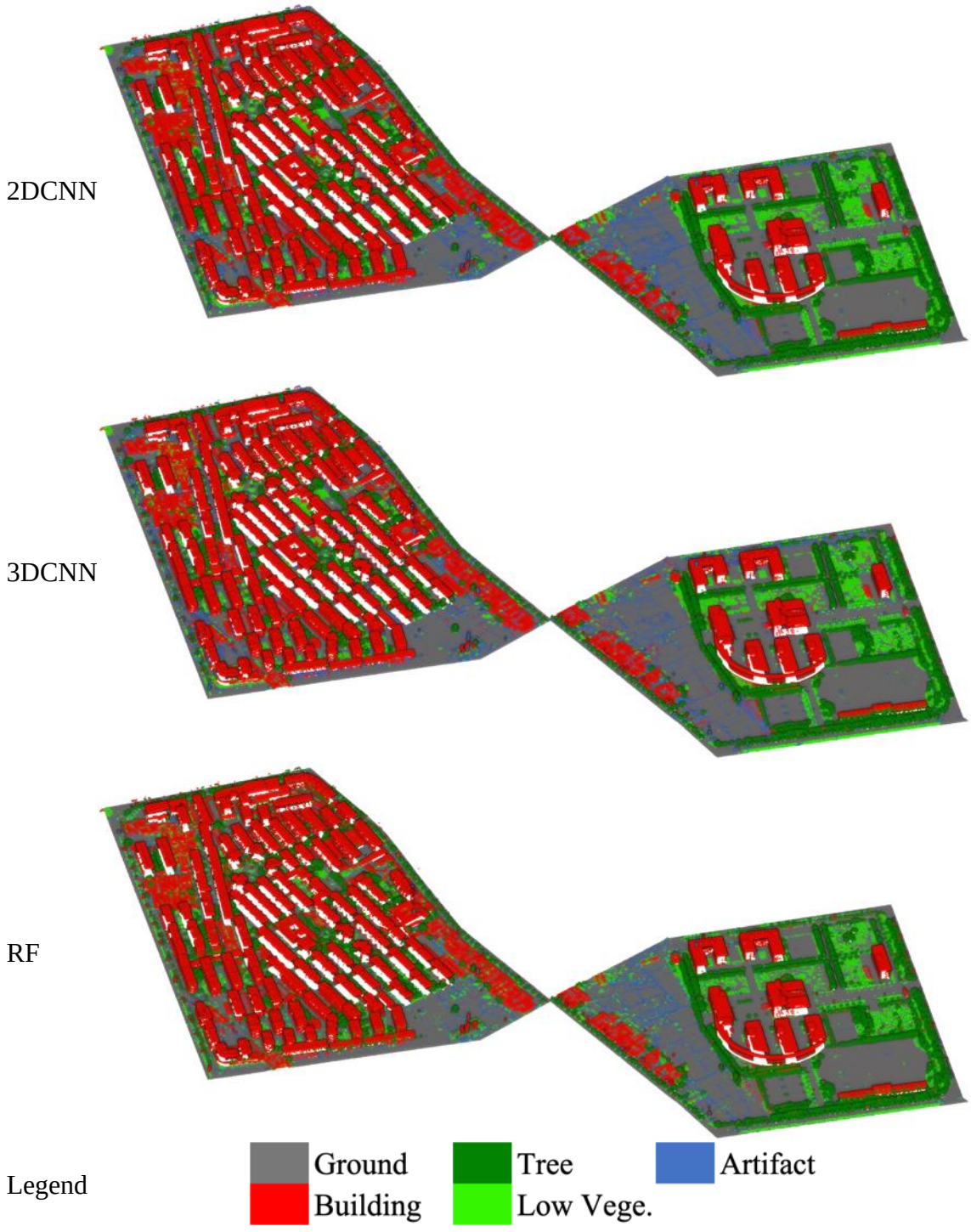
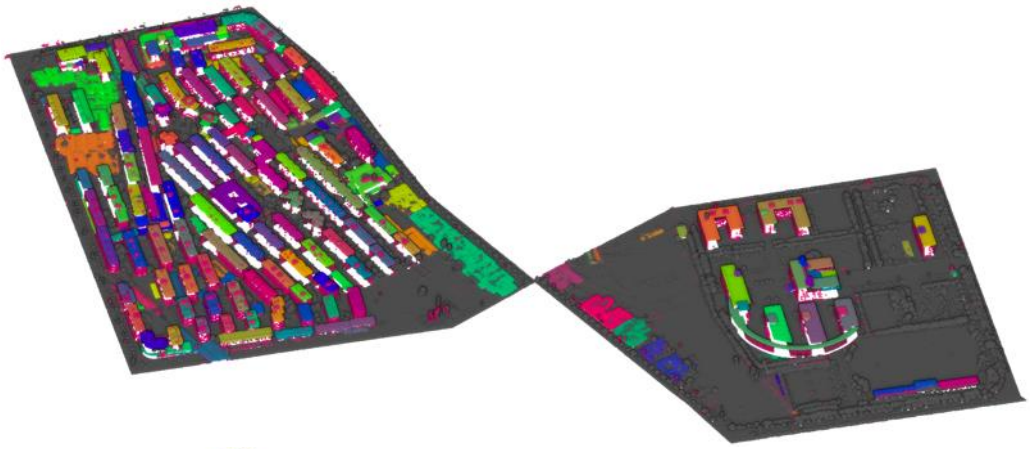
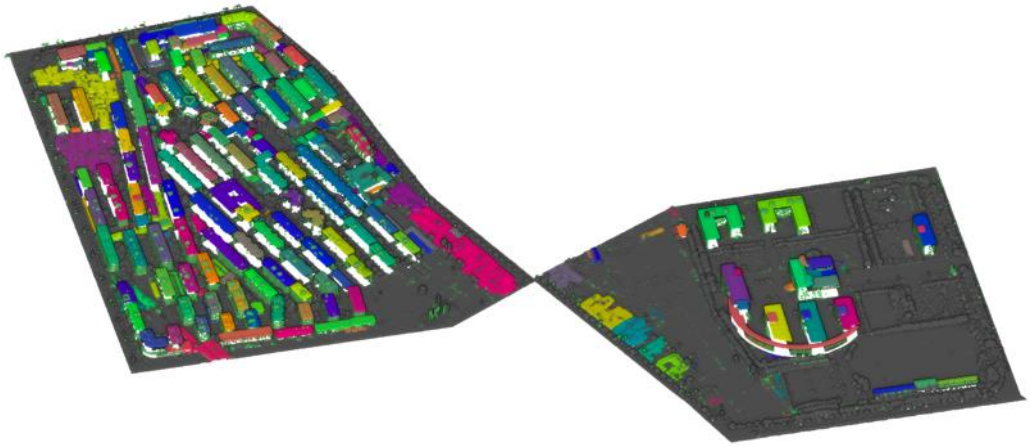


Figure 4-14. Classification results of the LASDU dataset with proposed DL methods and ML method.

2DCNN



3DCNN



RF

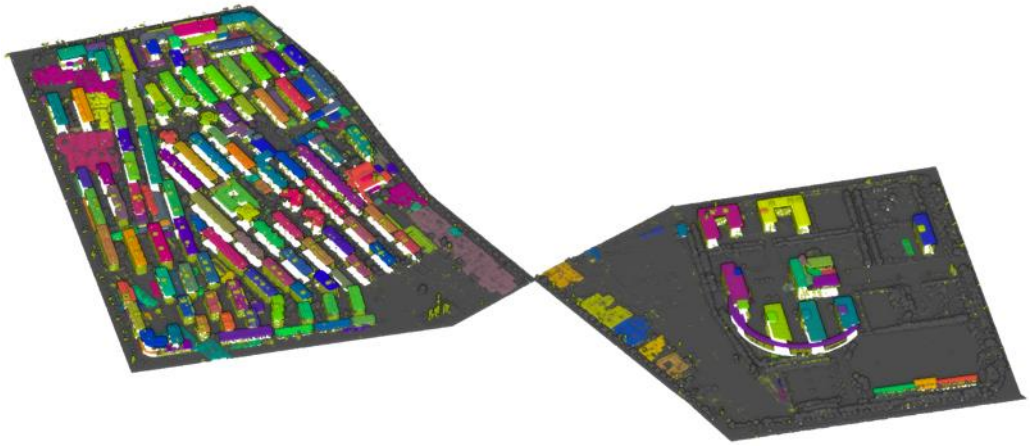


Figure 4-15. Building instances (randomly colored) and the other objects (gray), LASDU evaluation dataset.

4.4.4. Bordeaux

The accuracy metrics per class and the OA analysis are given in the tables below for quantitative assessment. For the qualitative assessment, the classification results are represented in the figure below. Instance segmentation results are then shared for the qualitative assessment without quantitative assessment due to lack of ground truth.

Table 4-19. Per-class accuracy assessment for the Bordeaux dataset. Bold values highlight higher scores among classifiers.

	Ground	**Facade**	**Roof**	**Veget.**	**Others**
2DCNN - F1	**0.972**	0.819	**0.956**	**0.986**	**0.708**
3DCNN - F1	0.966	0.807	0.951	0.985	0.682
RF - F1	0.969	**0.821**	0.954	0.972	0.567
2DCNN - IoU	**0.945**	0.694	**0.916**	**0.972**	**0.548**
3DCNN - IoU	0.934	0.676	0.907	0.970	0.517
RF - IoU	0.940	**0.696**	0.912	0.946	0.396

Table 4-20. Average F1, class weighted average F1, and OA for the Bordeaux dataset.

	Average F1	**Weigh. Av. F1**	**OA**
2DCNN - F1	**0.888**	**0.943**	**0.944**
3DCNN - F1	0.878	0.937	0.938
RF - F1	0.857	0.940	0.940
2DCNN - IoU	**0.808**	**0.897**	**0.944**
3DCNN - IoU	0.801	0.887	0.938
RF - IoU	0.778	0.891	0.940

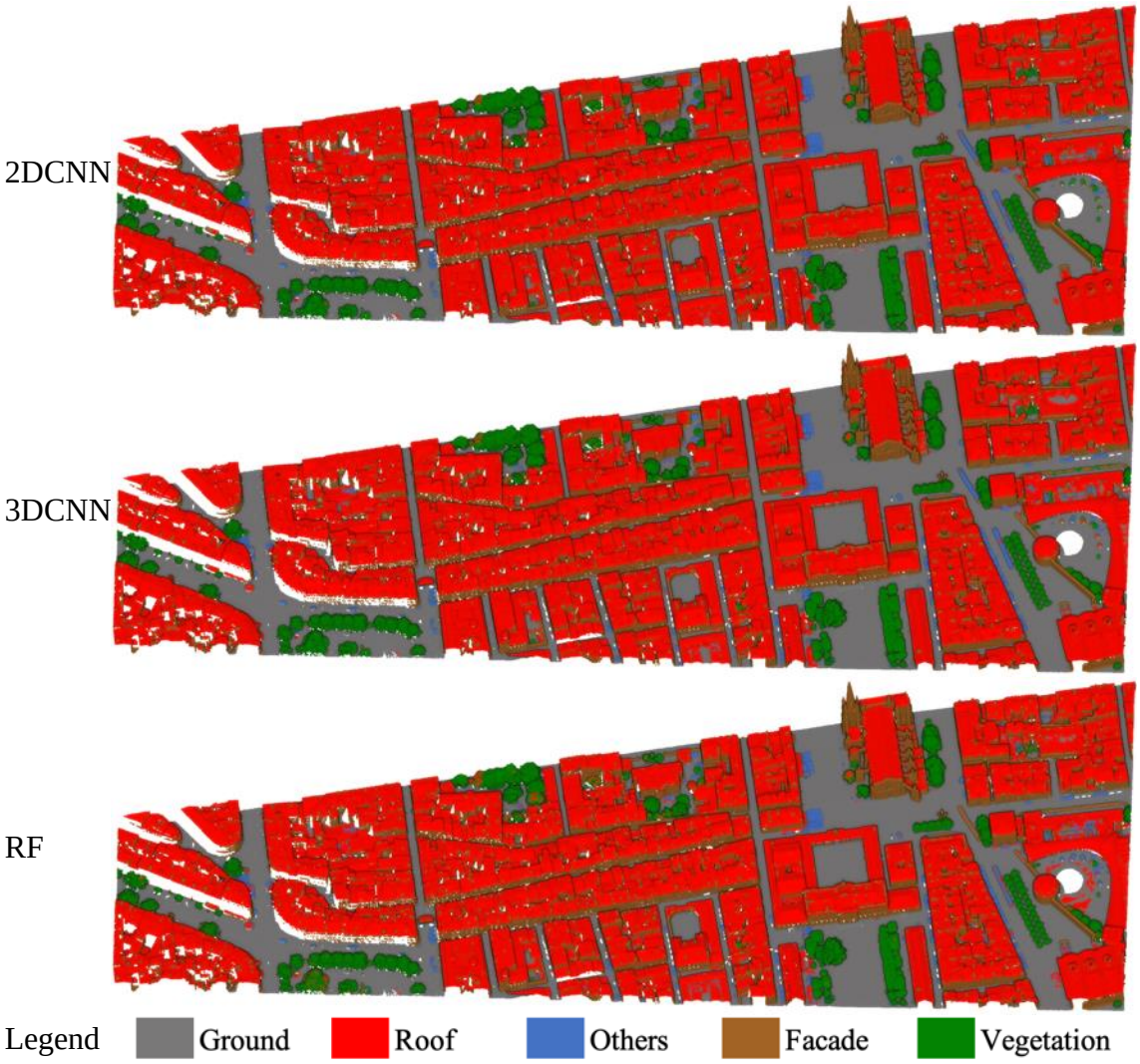


Figure 4-16. Classification results of the Bordeaux dataset with proposed DL methods and ML method.

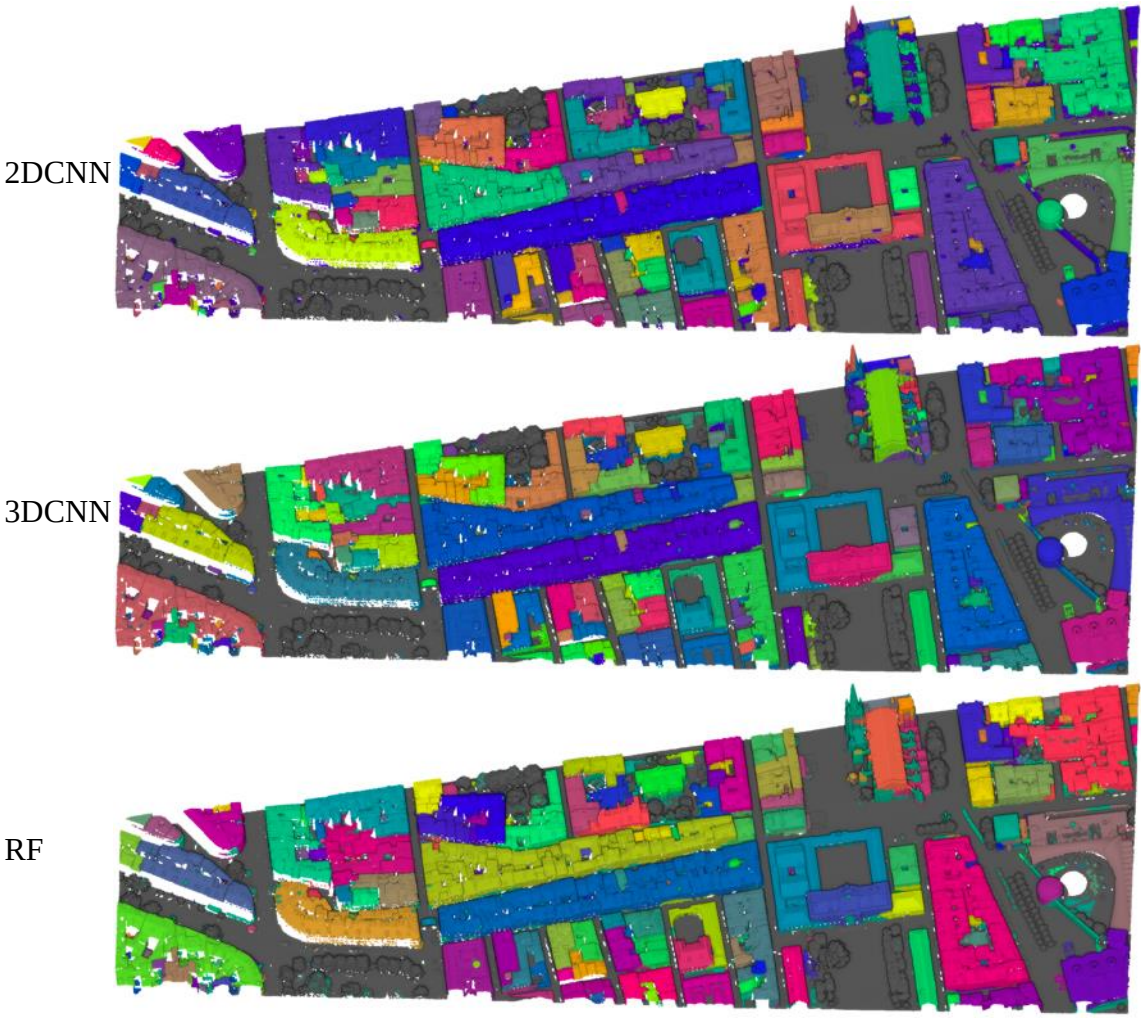


Figure 4-17. Building instances (randomly colored) and the other objects (gray), Bordeaux evaluation dataset.

4.4.5. 3DOMCity

The accuracy metrics per class and the OA analysis are given in the tables below for quantitative assessment. For the qualitative assessment, the classification results are represented in the figure below. Instance segmentation results are not shared for this dataset, as this is in a different scale compared to the geospatial point clouds.

Table 4-21. Per-class accuracy assessment for the 3DOMCity dataset. Bold values highlight higher scores among classifiers.

	Ground	**Grass**	**Shrub**	**Tree**	**Facade**	**Roof**
2DCNN - F1	0.945	0.936	**0.798**	**0.878**	0.864	**0.906**
3DCNN - F1	**0.954**	**0.938**	0.777	0.864	**0.866**	0.887
RF - F1	0.927	0.908	0.725	0.807	0.827	0.848
2DCNN - IoU	0.897	0.880	**0.664**	**0.782**	0.761	**0.828**
3DCNN - IoU	**0.913**	**0.883**	0.635	0.760	**0.763**	0.796
RF - IoU	0.863	0.831	0.569	0.676	0.705	0.737

Table 4-22. Average F1, class weighted average F1, and OA for the 3DOMCity dataset.

	Average F1	**Weigh. Av. F1**	**OA**
2DCNN - F1	**0.888**	**0.889**	**0.889**
3DCNN - F1	0.881	0.883	0.883
RF - F1	0.840	0.841	0.841
2DCNN - IoU	**0.802**	**0.802**	**0.889**
3DCNN - IoU	0.792	0.793	0.883
RF - IoU	0.730	0.729	0.841

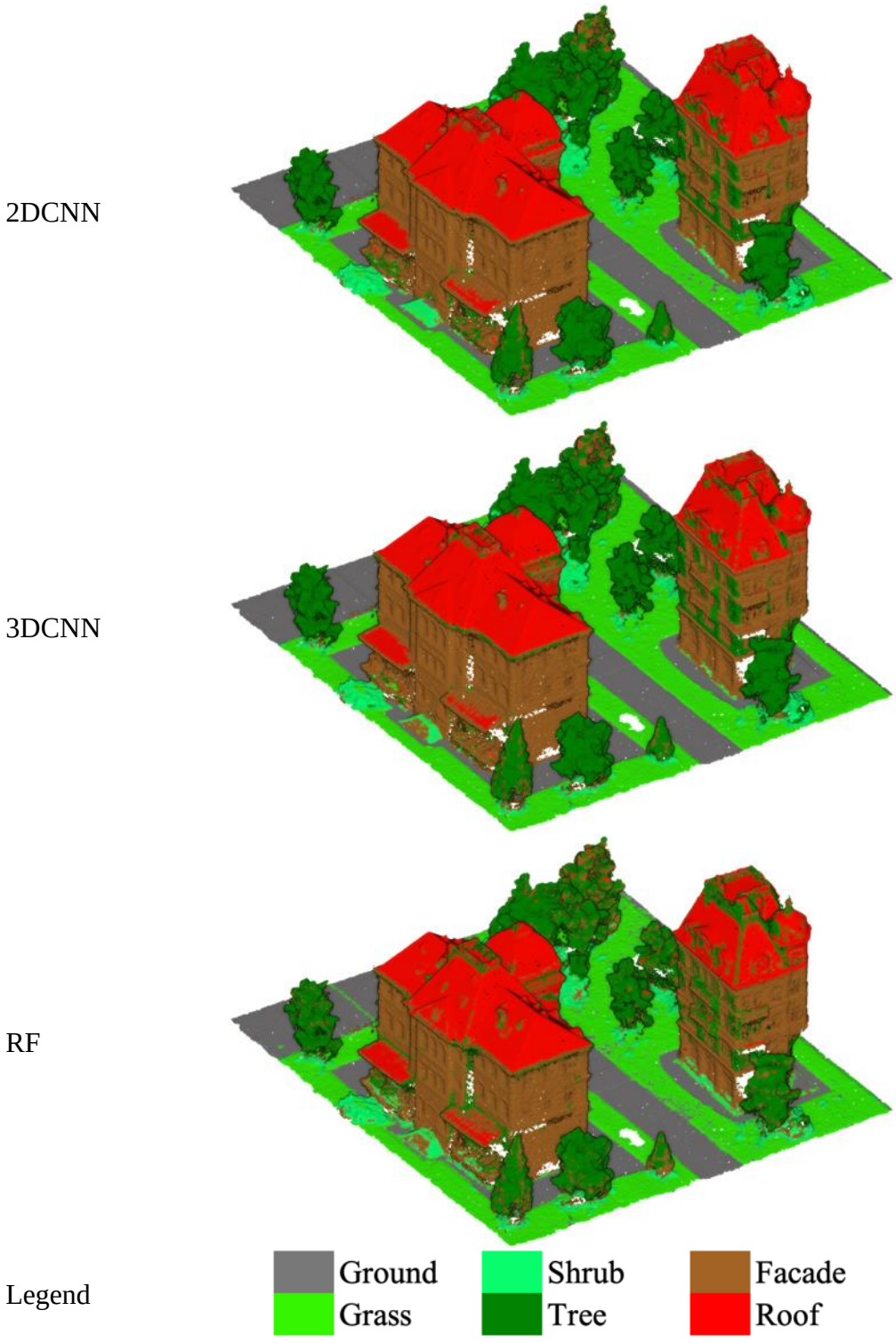


Figure 4-18. Classification results of the 3DOMCity dataset with proposed DL methods and ML method.

4.5. Summary of the Chapter

In this chapter, accuracy assessment methodology, datasets, downsampling approach and experiments, and the results from the TONIC framework are reported for distinct datasets. In the next chapter, validation of the framework will be discussed through generalization experiments and comparisons with the current state-of-the-art methods.

Chapter 5.

Validation

of the Framework

In this chapter, the following topics will be discussed: generalization performance of the framework with further experiments, and comparison of the results with the state-of-the-art methods.

5.1. Generalization Experiments

To test the framework for understanding its generalization capabilities, several experiments performed with training and predicting on separate datasets. Specifically, the classifier models trained for the experiments in Section 4.3 are used for classifying other datasets. In this way, the generalization ability of the framework could be analyzed.

One adaptation needed to be applied here is matching of the feature spaces. For instance, if a model is trained without the number of returns, but the prediction dataset contains this feature, that feature is simply removed from the prediction dataset to match

the feature spaces. This process can be summarized as selecting the feature space of the prediction dataset with respect to the training dataset used for the model. To clarify, a sample is shown in Table 5-1 for the DALES-ISPRS Vaihingen datasets.

Table 5-1. Original feature spaces and modifications for generalization experiments.

Feature	DALES (Train)	ISPRS Vaihingen (Predict)	Intersection
Color	-	IR + R + G	-
Intensity	-	Yes	-
Number of Returns	Yes	Yes	Yes
Return Number	Yes	Yes	Yes
Handcrafted	Yes	Yes	Yes

As seen in Table 5-1, in the case of the DALES-ISPRS Vaihingen experiment, color and intensity features are removed from the ISPRS Vaihingen dataset to use it as prediction data for the model trained on the DALES dataset. The handcrafted features are always kept as they are computed, yet, the feature space matching via intersection needs to be applied for sensor data. Feature space matching modifications will be mentioned for each experiment. For clarification, no computations are performed on the feature spaces. This step is only removing some sensor features to match the spaces, when needed.

Aside from matching the feature spaces, the class structures are needed to be matched. This is due to re-arranging the ground truth for accuracy assessment. An example is shown in Table 5-2. Changes applied to match the classes, along with point distributions within the datasets, will be reported for each generalization experiment. To clarify, the models are not re-trained. The models are used *as is* without any change. Only the class labels are re-arranged to match them for accuracy assessment.

Table 5-2. Corresponding classes between the DALES and the ISPRS Vaihingen datasets, along with their distributions, are shown.

DALES	Points (Train)	ISPRS Vaihingen	Points (Predict)
Cable	29.7%	Powerline	0.2%
Ground	50.0%	Low veget. / Imp. Surface	38.9%
Car / Truck	1.2%	Car	0.8%
Fence	0.6%	Fence	1.9%
Building / Pole	0.9%	Roof / Facade	22.7%
Vegetation	17.6%	Shrub / Tree	35.5%

Table 5-3. F1 and IoU scores for 2DCNN and 3DCNN classifiers trained on the DALES dataset and predicting on the ISPRS Vaihingen dataset.

	Cable	Ground	Car	Fence	Building	Vegetation
2DCNN DALES F1	0.130	0.926	**0.399**	0.001	0.759	0.397
3DCNN DALES F1	**0.230**	**0.929**	0.347	0.000	0.758	0.318
RF DALES F1	0.123	0.916	0.168	**0.039**	**0.767**	**0.429**
2DCNN DALES IoU	0.069	0.862	**0.249**	0.000	0.611	0.247
3DCNN DALES IoU	**0.130**	**0.867**	0.210	0.000	0.610	0.189
RF DALES IoU	0.065	0.846	0.092	**0.020**	**0.622**	**0.273**

Table 5-4. Average F1, class weighted average F1, and OA for 2DCNN and 3DCNN classifiers trained on the DALES dataset and predicting on the ISPRS Vaihingen dataset.

	Average F1	Weigh. Av. F1	OA
2DCNN DALES F1	**0.435**	0.753	**0.779**
3DCNN DALES F1	0.430	0.739	0.774
RF DALES F1	0.407	**0.755**	0.761
2DCNN DALES IoU	**0.340**	**0.649**	**0.779**
3DCNN DALES IoU	0.334	0.639	0.774
RF DALES IoU	0.319	0.647	0.761

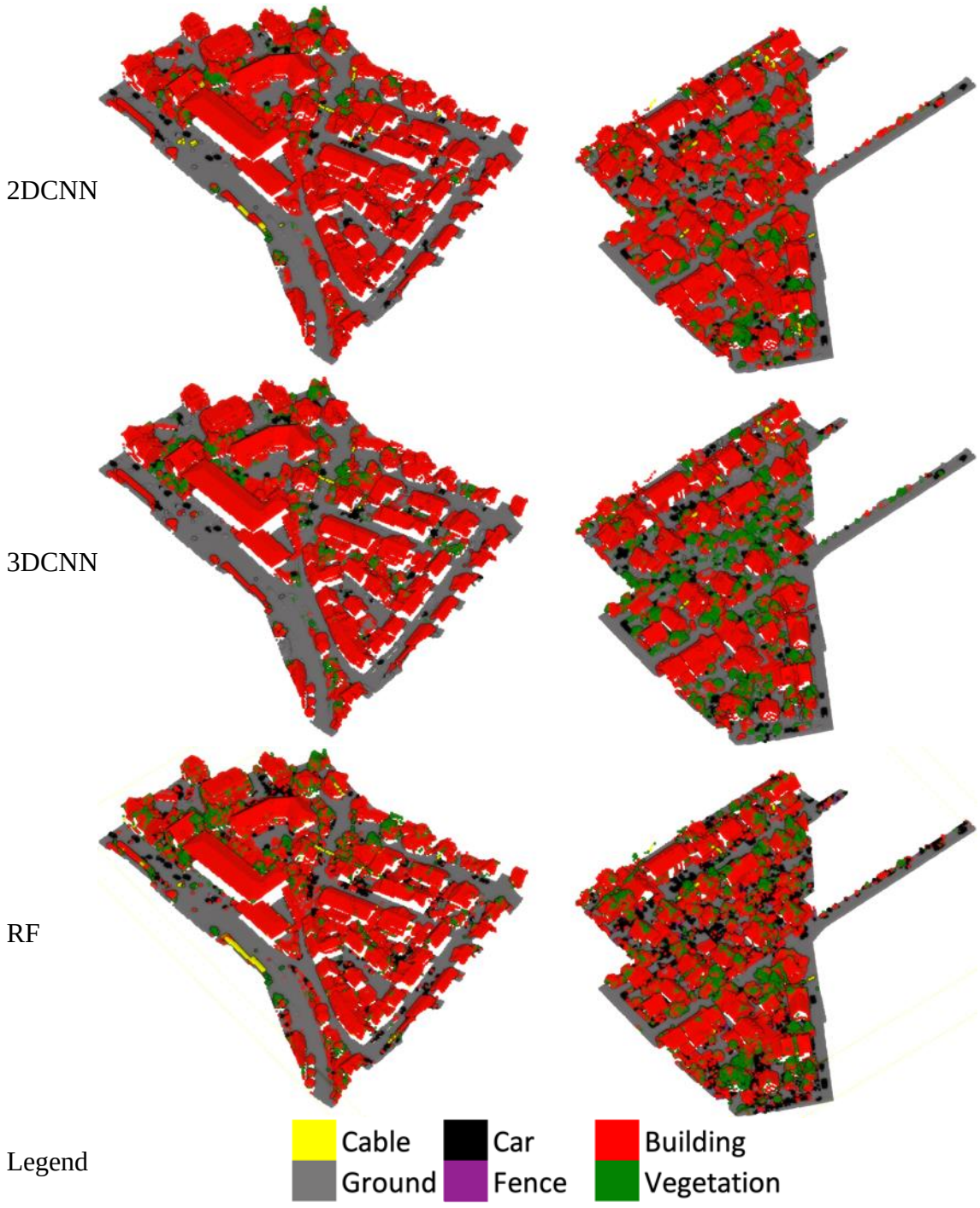


Figure 5-1. Classification results of the ISPRS Vaihingen dataset on models trained with the DALES dataset.

The second generalization experiment is with the same classifiers as the first one, tested on the Bordeaux dataset. Color data (from photogrammetry), as well as the intensity feature (from LiDAR), is removed for these runs. Firstly the corresponding classes, then the results are reported in the following tables.

Table 5-5. Corresponding classes between the DALES and Bordeaux datasets, along with their distributions, are shown.

DALES	**Points (Train)**	**Bordeaux**	**Points (Predict)**
Ground	29.7%	Ground	13.9%
Building / Cable / Pole	50.0%	Roof / Facade	80.2%
Vegetation	2.1%	Vegetation	5.0%
Car / Truck / Fence	18.2%	Others	1.0%

Table 5-6. F1 and IoU scores for 2DCNN and 3DCNN classifiers trained on the DALES dataset and predicting on Bordeaux dataset.

	Ground	**Building**	**Vegetation**	**Others**
2DCNN DALES F1	**0.965**	**0.978**	**0.947**	0.434
3DCNN DALES F1	0.956	0.972	0.937	**0.462**
RF DALES F1	0.948	0.945	0.618	0.203
2DCNN DALES IoU	**0.931**	**0.958**	**0.900**	0.277
3DCNN DALES IoU	0.916	0.945	0.882	**0.300**
RF DALES IoU	0.901	0.895	0.447	0.113

Table 5-7. Average F1, class weighted average F1, and OA for 2DCNN and 3DCNN classifiers trained on the DALES dataset and predicting on Bordeaux dataset.

	Average F1	Weigh. Av. F1	OA
2DCNN DALES F1	0.831	**0.968**	**0.969**
3DCNN DALES F1	**0.832**	0.961	0.961
RF DALES F1	0.678	0.926	0.914
2DCNN DALES IoU	**0.766**	**0.941**	**0.969**
3DCNN DALES IoU	0.761	0.927	0.961
RF DALES IoU	0.589	0.871	0.914

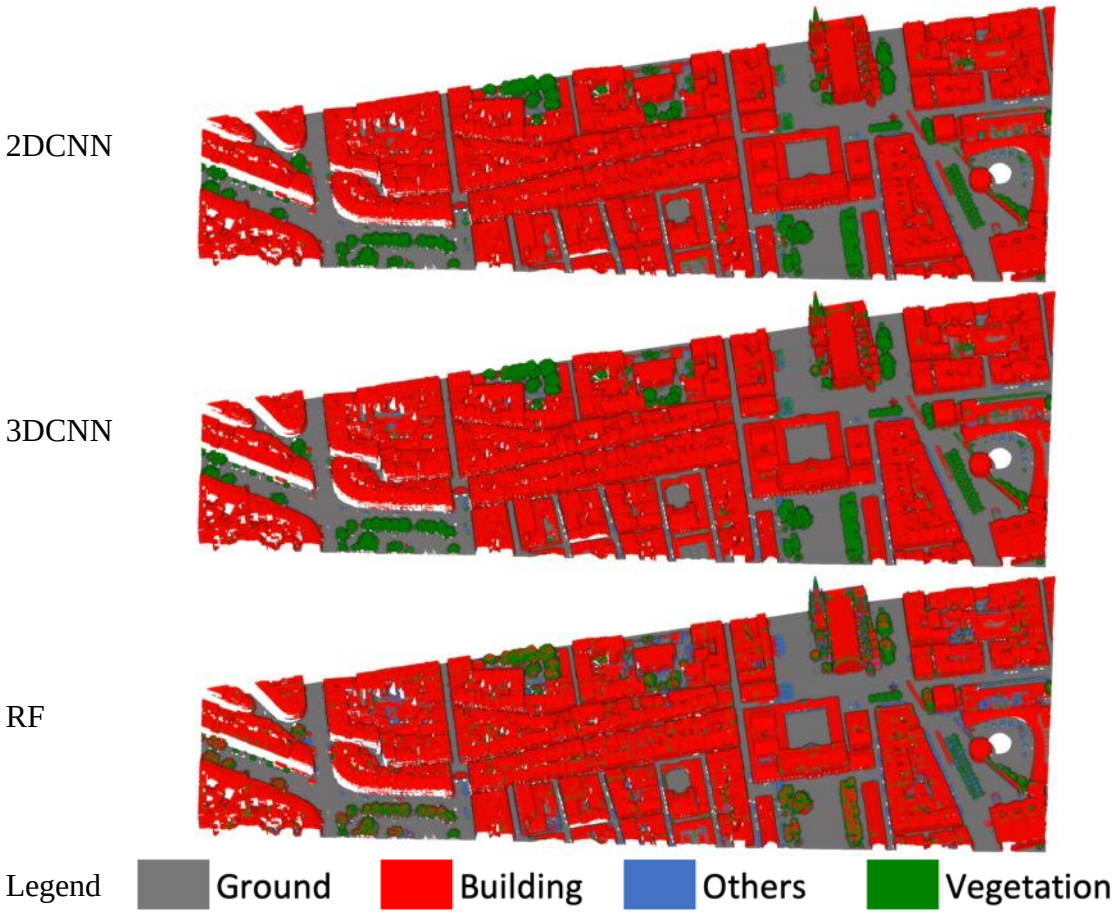


Figure 5-2. Classification results of Bordeaux dataset on models trained with the DALES dataset.

The third generalization experiment is with the ISPRS Vaihingen classifiers, tested on the Bordeaux dataset. For this experiment, the color data of the Bordeaux dataset (RGB) are kept, although the models are trained on the color-space of the ISPRS Vaihingen dataset (IR-R-G). This is due to:

(i) the input dimensions of the models are fixed and cannot be changed;

(ii) experimenting the generalization capability of the models also with some noisy data, assuming the differences in the color spaces as noise.

Corresponding classes and the results are reported in the following tables.

Table 5-8. Corresponding classes between the ISPRS Vaihingen and the Bordeaux datasets, along with their distributions, are shown.

ISPRS Vaihingen	**Points (Train)**	**Bordeaux**	**Points (Predict)**
Ground	18.1%	Ground	13.9%
Roof	26.9%	Roof	43.6%
Facade	5.1%	Facade	36.6%
Low Veget. / Shrub / Tree	46.1%	Vegetation	5.0%
Cables / Car / Fence	3.7%	Others	1.0%

Table 5-9. F1 and IoU scores for 2DCNN and 3DCNN classifiers trained on the ISPRS Vaihingen dataset and predicting on the Bordeaux dataset.

	Ground	**Facade**	**Roof**	**Vegetation**	**Others**
2DCNN ISPRS F1	**0.921**	0.690	**0.923**	0.759	**0.400**
3DCNN ISPRS F1	0.882	**0.743**	0.878	**0.832**	0.331
RF ISPRS F1	0.178	0.589	0.824	0.218	0.000
2DCNN ISPRS IoU	**0.854**	**0.527**	**0.857**	0.612	**0.250**
3DCNN ISPRS IoU	0.789	0.591	0.782	**0.712**	0.198
RF ISPRS IoU	0.098	0.418	0.700	0.123	0.000

Table 5-10. Average F1, class weighted average F1, and OA for 2DCNN and 3DCNN classifiers trained on the ISPRS Vaihingen dataset and predicting on the Bordeaux dataset.

	Average F1	Weigh. Av. F1	OA
2DCNN ISPRS F1	0.739	0.878	**0.882**
3DCNN ISPRS F1	0.733	0.855	0.855
RF ISPRS F1	0.362	0.548	0.506
2DCNN ISPRS IoU	0.620	0.793	**0.882**
3DCNN ISPRS IoU	0.614	0.751	0.855
RF ISPRS IoU	0.268	0.434	0.506

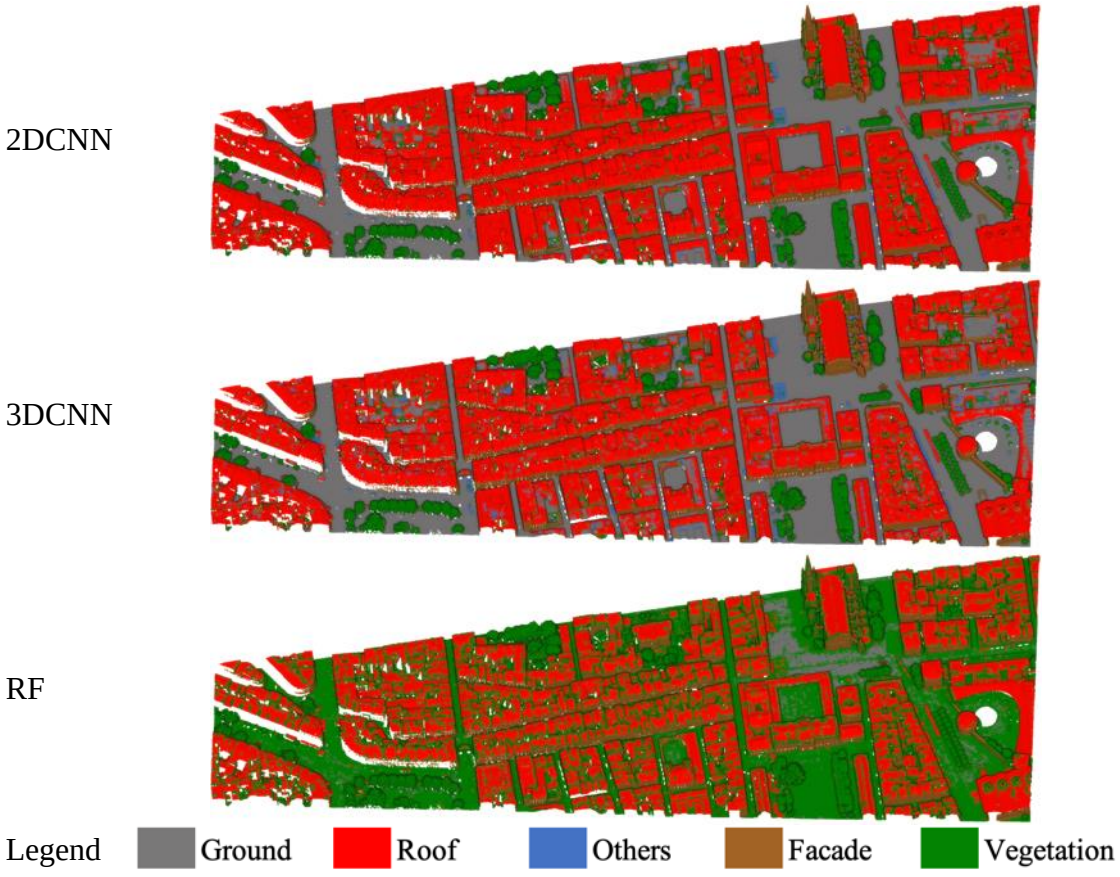


Figure 5-3. Classification results of the Bordeaux dataset on models trained with the ISPRS Vaihingen dataset.

5.2. Comparisons with the State-of-the-Art

In this section, the proposed framework is compared against the current state-of-the-art methods. Besides the accuracy metrics, training timings are also reported, which are acquired from the original research publications.

The state-of-the-art comparisons include not only point cloud classification frameworks but also an alternative CNN to classify using TONIC's data matrix structure (Figure 3-6): *EfficientNet*, B7 version specifically (Tan and Le, 2019). The EfficientNetB7 is reported as the most accurate among the other *EfficientNet*s. Therefore, only this version is used for comparison here.

The network is implemented in the TensorFlow library, and it is applied as shown in Figure 5-4. Due to its network architecture, the EfficientNetB7 cannot be fed with images smaller than 32x32 pixels dimension. For this reason, the matrices are zero-padded in order to fulfill this.

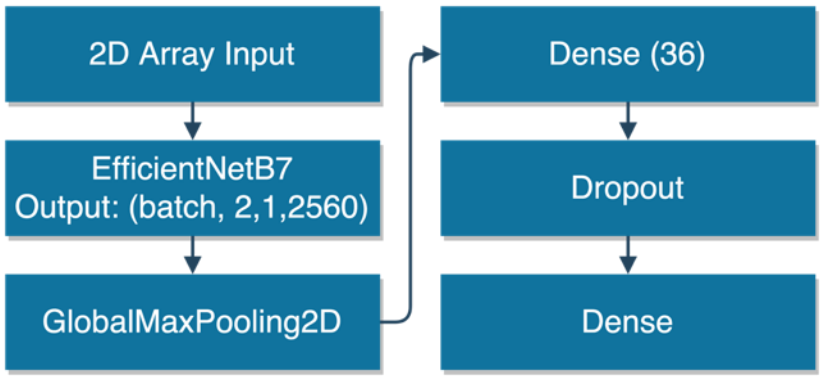


Figure 5-4. EfficientNetB7 implementation via TensorFlow library.

Table 5-11 represents the comparisons for the ISPRS Vaihingen dataset in terms of computational efficiency and accuracy. It can be seen TONIC is faster than the state-of-

the-art methods, on par with them in terms of accuracy, and it consumes less energy and memory.

Table 5-11. Comparison of the performances between TONIC framework and recent papers, ordered by OA. TFLOPS indicates the computational power of the GPU for single-precision floating-point (FP32) operations. Training times are given in hours. Differences from the highest OA score in the table are shown with asterisk.

Method	GPU TFLOPS	Training Time	GPU Watt	GPU Memory	OA	GPU
Li et al. (2020b)	8.73	10	250	24 GB	**0.839**	Tesla K80
Li et al. (2020a)	2 x 12.15	7	2 x 250	2 x 12 GB	0.835	2 x Titan Xp
Wen et al. (2021)	12.15	10	250	12 GB	0.832	Titan Xp
Chen et al. (2021)	14.13	2	250	32 GB	0.832	Tesla V100
EfficientNetB7 Tan and Le (2019)	13.45	1	250	11 GB	0.748 (−9.1%*)	RTX 2080Ti
Ours (2DCNN)	13.45	**0.15**	250	11 GB	0.826 (−1.3%*)	RTX 2080Ti
Ours (3DCNN)	13.45	0.5	250	11 GB	0.806 (−3.3%*)	RTX 2080Ti

In terms of prediction timings performance for the ISPRS Vaihingen dataset, TONIC framework requires ~9.5 seconds on Nvidia RTX 2080Ti GPU. For comparison, EfficientNetB7 takes ~23 seconds for this dataset with the same hardware configuration. Feature extraction for the dataset takes less than 10 seconds on Intel i9-8950HK Mobile CPU using 12 threads. As a result, even with the combination of a mid-level laptop CPU

and a decent GPU, both of the DL models of TONIC are faster and more accurate than the *EfficientNet*, while achieving accuracies on par with the current state-of-the-art methods. The RF model is not reported in the table as it is not a DL-based approach. Yet, it takes less than 1 minute to train the RF model on the same Mobile CPU while consuming less than 2 GB of system memory.

TONIC framework is also implemented for AMD GPUs, using PlaidML (PlaidML, 2019), which is a Python library enables running Keras on OpenCL (Stone et al., 2010), as well as Apple's Metal API (Apple, 2021). The performance comparison of the same 2DCNN model running on different GPUs is given in Table 5-13. In order to fit into a smaller GPU memory, batch size is reduced. The training on Nvidia GPU utilizes CUDA (Nickolls et al., 2008) drivers, while the training on AMD GPU utilizes Metal API.

Table 5-12. Comparison of the performance between different GPUs, running the same model

Method	GPU FP32 TFLOPS	Training Time	GPU Watt	GPU Memory	GPU
Ours (2DCNN)	13.45	**0.15**	250	11 GB	Nvidia RTX 2080Ti
Ours (2DCNN)	2.46	**1.0**	60	4 GB	AMD Radeon Pro 560X Mobile

Other accuracy comparisons are held with the DALES dataset comparing the TONIC's networks with IoU metrics with respect to the current state-of-the-art methods in Table 5-13. Both of the KPConv and PointNet++ outperform the proposed method by 3–4% in terms of OA.

Table 5-13. IoU per class and OA scores on the DALES dataset with respect to current state-of-the-art methods.

Method	Ground	Veget.	Car	Truck	Cable	Fence	Pole	Building	OA
KPConv (Thomas et al.)	0.971	0.941	0.853	0.419	0.955	0.635	0.750	0.966	0.978
PointNet++ (Qi et al., 2017)	0.941	0.912	0.754	0.303	0.799	0.462	0.400	0.891	0.957
Ours (2DCNN)	0.926	0.863	0.499	0.000	0.823	0.360	0.306	0.837	0.938
Ours (3DCNN)	0.919	0.857	0.517	0.000	0.841	0.325	0.377	0.826	0.934

For the LASDU dataset, comparisons of the classification accuracy between the proposed framework and available references are given in Table 5-14. The proposed models outperform the reference methods (PointNet++ and HDA-PointNet++) by 1–4% in average F1 score and 1–3% OA.

Table 5-14. F1 scores and OA scores on the LASDU dataset with respect to current state-of-the-art methods.

Method	Ground	Building	Tree	LV	Artifact	Avg. F1	OA
PointNet++ (Qi et al., 2017)	0.877	0.906	0.820	0.632	0.313	0.710	0.828
HDA-PointNet++ (Huang et al., 2020)	0.887	0.932	0.822	0.652	0.369	0.733	0.844
Ours (2DCNN)	0.887	0.935	0.860	0.691	0.360	**0.746**	**0.846**
Ours (3DCNN)	0.885	0.915	0.858	0.673	0.322	0.730	0.837

5.3. Summary of the Chapter

In this chapter, generalization experiments along with the comparisons with the current state-of-the-art methods are discussed. The comparisons were held not only in

terms of accuracy but also computational efficiency. In the next chapter, these results will be discussed, limitations of the framework will be debated along with future work, and a conclusion will be made.

Chapter 6.

Discussion and Conclusions

In this chapter, the results (Chapters 4 and 5) will be discussed for distinct dataset performances as well as generalization performances. These discussions will be followed by summary of the achievements, limitations, future works, and conclusion, respectively.

6.1. Discussion

The reported results (Chapters 4 and 5) indicated that 2DCNN and 3DCNN models achieve more accurate classification results than RF classifier with the feature space defined in Section 3.2. A summary of these results for each dataset can be seen in Table 6-1. One exception is observed with the Bordeaux dataset where RF outperforms 3DCNN by 0.2% while being outperformed by 2DCNN by 0.4%. The differences in OA in distinct datasets (Table 6-1) are not very high, especially among DL methods. Specifically, the most significant gap observed in distinct datasets is 4.8% in the 3DOMCity dataset between 2DCNN and RF models. Aside from these, in some cases (Table 4-13 cables, Table 4-15

car, pole, building, Table 4-19 facade, Table 4-21 ground, grass), it is observed that some models may outperform the others for a specific class in a specific dataset. These are considered insignificant cases with small margins, as they are not consistent.

Table 6-1. Summarized OA achieved in distinct datasets.

Model	ISPRS Vaihingen	DALES	LASDU	Bordeaux	3DOMCity
2DCNN	**0.826**	**0.938**	**0.846**	**0.944**	**0.889**
3DCNN	0.806	0.934	0.837	0.938	0.883
RF	0.786	0.899	0.821	0.940	0.841

Based on the accuracies of the models in Table 6-1 and the computational performances reported in Table 5-11, the 2DCNN model can be considered as the best performing one for distinct datasets, followed by 3DCNN. Although the RF is the fastest, it falls behind the other two models in terms of accuracy for all the datasets between 0.4-4.8% OA.

As for the density variations, the framework achieved the targeted accuracy on point clouds with varying resolutions between 0.116m to 0.484m (Table 4-2). Based on these results, the framework is expected to operate successfully on geospatial point clouds with resolutions up to ~0.7m, due to the decided downsampling resolution indicated in Section 4.3.

In addition to the distinct dataset experiments, the generalization experiments are similarly very critical. In Table 6-2, these generalization experiments (Section 5.1) are summarized for discussion.

Table 6-2. Summarized OA for the generalization tests.

Trained on	Predicted on	Model	OA
DALES	ISPRS Vaihingen	2DCNN	**0.779**
DALES	ISPRS Vaihingen	3DCNN	0.774
DALES	ISPRS Vaihingen	RF	0.761
DALES	Bordeaux	2DCNN	**0.969**
DALES	Bordeaux	3DCNN	0.961
DALES	Bordeaux	RF	0.914
ISPRS Vaihingen	Bordeaux	2DCNN	**0.882**
ISPRS Vaihingen	Bordeaux	3DCNN	0.855
ISPRS Vaihingen	Bordeaux	RF	0.506

As it can be seen, the 2DCNN method outperforms the others also in the generalization experiments. The RF model is the worst performing in the experiments. Especially in the ISPRS Vaihingen-Bordeaux experiment, the RF classifier had strong confusions between ground and vegetation, which can be seen in Table 5-9 as well as in Figure 5-3.

To consider both distinct dataset experiments and generalization experiments, these results are summarized in Table 6-3 with their averages. As it can be seen, 2DCNN outperforms the others for both usages.

Table 6-3. Average OA per model. DD: Distinct Datasets, Gen.: Generalization

Model	Avg OA on DD	Avg OA on Gen.	Avg OA
2DCNN	**0.889**	**0.877**	**0.883**
3DCNN	0.880	0.863	0.871
RF	0.857	0.727	0.792

The reported results indicate that the TONIC framework can outperform the current state-of-the-art methods by a few percent of OA (Table 5-14), while requiring less memory

and energy consumption due to its design (Table 5-11). These few percent differences are not expected to lead to significant changes in the usefulness of AI methods in terms of daily use. A manual quality control will be required for any of these methods. A few percent OA difference is not expected to change the manual labor needed for the corrections significantly. However, it may not be possible to make similar claims for computational efficiency. Aside from the energy consumption, which might be neglected with the use of environmentally friendly energy sources, the significant impact of the TONIC can be seen in the lower computational power and lower memory requirements reported in Table 5-12.

The main methodological differences between TONIC framework and the current state-of-the-art methods can be seen as the following two:

(i) downsampling with voxel-grid filtering with predefined voxel dimensions,

(ii) multi-scale handcrafted feature extraction before passing the data to DNN.

These pre-processing steps, especially the feature extraction, allow designing a *shallower* DNN compared to the state-of-the-art methods, supporting the efficiency objective. A *shallower* DNN does not only run faster but also requires less memory by its nature. During the experiments, 1024 points-per-batch were fitted using a GPU with 11 GB memory. Needless to say, lower hardware requirements lower the hardware costs and speed up the process, which are favorable characteristics for a method, especially for daily deployments on large (i.e., city-scale or country-scale) datasets.

The reported generalization experiment results (summarized in Table 6-2) are promising, as the proposed methods reach accuracies as high as distinct dataset classification results. A fundamental challenge for generalization is to make the feature

spaces of distinct datasets share similar characteristics. The density variations between distinct datasets cause challenges in the local geometry definitions, as also mentioned in the literature several times (Sections 2.2 and 2.3) and encouraged researchers to develop different approaches to deal with this problem. TONIC framework includes a simple and efficient solution -downsampling- to cope with this while bringing several advantages, including data reduction for efficiency, reducing the density variations for better generalization capability, and noise reduction.

The reported results in Table 6-1 and Table 6-2 indicate the fact that there is a high correlation between the dataset used for prediction and the achieved accuracy. For example, on the ISPRS Vaihingen dataset, TONIC achieved OA 79-83%. Training the models on the DALES dataset, the achieved OAs are 76-78%. Similarly, on the Bordeaux dataset, TONIC achieves OA ~94%, training them on the DALES dataset the achieved OAs are 91-97%, training them on the ISPRS Vaihingen they can (excluding RF) reach up to 88% OA. Comparable observations can also be made with the state-of-the-art methods: PointNet++ achieves 96% OA on the DALES (Table 5-13) dataset while it reaches 83% on the LASDU (Table 5-14) dataset. Based on these results, it can be concluded that there is a high correlation between the used datasets for prediction and the achieved OA. The reason for this can be the noise in the ground truth labels (i.e., wrong labeled points), noise in the point cloud (i.e., isolated points), and structure of classes in the data (more and complex classes versus simpler and fewer classes).

6.2. Summary of the Achievements

The developed methodology's novel approach of implementing, pre-processing with downsampling, multi-scale handcrafted feature extraction, and DL integration defines a new relationship between aerial 3D point clouds and CNNs. This innovative methodology is proven to achieve all the objectives mentioned in Section 1.5:

(i) Process point clouds from different sensors with high accuracy (Sections 4.3 and 4.4 or Table 6-1 for summary);

(ii) Generalization (Section 5.1 or Table 6-2 for summary);

(iii) Being invariant to the density variations: both within distinct datasets (Section 4.4) and among different datasets (Section 5.1);

(iv) Lower computational costs and hardware requirements compared to the state-of-the-art (Tables 5-11 and 5-12);

(v) Achieving better or similar accuracies compared to the current state-of-the-art (Section 5.2, Tables 5-11, 5-13, and 5-14).

6.3. Limitations and Future Work

The main limitation of the developed method is that it is tailored for urban scenarios, where the data acquisition is made using aerial sensors (photogrammetry or LiDAR). Due to the used handcrafted features, TONIC might not be competitive in alternative cases other than geospatial point cloud classification. It can be seen as a purpose-built kind of method rather than a one-fits-for-all kind.

Another limitation is the isolated noise in the point clouds. The framework expects the received point cloud to be free from such isolated points. Although the implemented downsampling is helpful with reducing the noisy points on surfaces, isolated points are yet to be considered.

Several open issues can be addressed in the future:

- UAV point clouds to be examined. The handcrafted features are proven for typical airborne geospatial data (LiDAR or photogrammetry), yet, UAVs are used for geospatial data acquisitions, as well. Due to the use of different platforms with lower flight altitudes, these point clouds are expected to have other characteristics (i.e., geometry, density, visible objects).
- Satellite-based point clouds to be explored. 3D point clouds can be derived with the current state-of-the-art high-resolution multi-view satellite imagery. The exploitation of such data can be helpful, especially for isolated areas, where it is not feasible to plan a flight with an aircraft loaded with photogrammetric equipment due to high costs. An example of satellite derived data is given in Chapter 1 (Figure 1-6), where it is seen that the spaceborne data can come with challenging geometry. Therefore, the TONIC framework could be tested on such data for further development.
- Data augmentation is a de facto standard when it comes to image processing with DL. Various types of augmentation are applied during training in order to increase the data amount as well as improving the overall performance of the networks. The data augmentation is not implemented for TONIC.

- 3D building model generation is an essential task for 3D city models. The use of the classified point clouds coming from TONIC can be examined. The 3D model generation can be seen as the next step of map making with geospatial point clouds. The classification step brings the semantics, so that the 3D data can be used in GIS for more advanced analysis.

6.4. Conclusions

In this thesis, a novel methodology for geospatial point cloud classification is proposed, which takes advantage of multi-scale handcrafted features and DL together. In this way, it not only reaches better or similar accuracies with the current state-of-the-art models but also becomes a more efficient alternative to the existing solutions, requiring less computational resources and computational time (as demonstrated in Table 5-11 and Table 5-12). Besides, the proposed framework is tested on different datasets acquired by various sensors with different resolutions.

The developed methodology experimented for generalization as well. The reported results suggest that TONIC can process data from different sources (photogrammetry or LiDAR) without compromising neither accuracy nor computational efficiency. Considering the amount of data available in the real world -compared to benchmark datasets- and the computational density of the state-of-the-art methods, it will not be surprising to see more research approaching with the efficiency perspective in the future. Based on the accuracies reported in Chapters 4 and 5, TONIC can be helpful for building

extraction (for 3D city modeling), powerline mapping, digital terrain model generation, and other geospatial studies relying on point clouds.

Acknowledgments for the Used Datasets

The Vaihingen data set was provided by the German Society for Photogrammetry, Remote Sensing and Geoinformation (DGPF). The authors would like to acknowledge Hexagon/Leica Geosystem for providing the Bordeaux dataset. The data set presented in this thesis contains information licensed under the Open Government License City of Surrey. Special thanks to the City of Surrey for generously providing the raw data presented in this paper. For more information about the raw data and other data sources like it please see their Open Data Site.

Bibliography

Apple. Metal | Apple Developer Documentation 2021 [Available from: https://developer.apple.com/documentation/metal/.

Bakuła, K, Mills, JP, Remondino, F. A Review of Benchmarking in Photogrammetry and Remote Sensing. *Int Arch Photogramm Remote Sens Spatial Inf Sci*. 2019 XLII-1/W2:1-8.

Becker, C, Rosinskaya, E, Häni, N, D'Angelo, E, Strecha, C. Classification of Aerial Photogrammetric 3D Point Clouds. *Photogrammetric Engineering & Remote Sensing*. 2018 84(5):287-295.

Bello, SA, Yu, S, Wang, C, Adam, JM, Li, J. Review: Deep Learning on 3D Point Clouds. *Remote Sensing*. 2020 12(11).

Bittner, K, d'Angelo, P, Körner, M, Reinartz, P. DSM-to-LoD2: Spaceborne Stereo Digital Surface Model Refinement. *Remote Sensing*. 2018 10(12).

Bossler, JD, Campbell, JB, McMaster, RB, Rizos, C. Manual of Geospatial Science and Technology: CRC Press; 2010.

Breiman, L. Random Forests. *Machine Learning*. 2001 45(1):5-32.

Brinker, RC, Minnick, R. The Surveying Handbook: Springer Science & Business Media; 1995.

Burkov, A. The Hundred-Page machine Learning Book: True Positive Inc.; 2019.

Butler, H. LibLAS LAS 1.0/1.1/1.2 ASPRS LiDAR Data Translation Toolset 2021 [Available from: https://liblas.org/start.html.

Charles, RQ, Su, H, Kaichun, M, Guibas, LJ, editors. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2017 21-26 July 2017.

Chen, Y, Liu, G, Xu, Y, Pan, P, Xing, Y. PointNet++ Network Architecture with Individual Point Level and Global Features on Centroid for ALS Point Cloud Classification. *Remote Sensing*. 2021 13(3).

Cheng, G, Li, X, Liu, S, Xiao, Q, Ma, M, Jin, R, Che, T, Liu, Q, Wang, W, Qi, Y, Wen, J, Li, H, Zhu, G, Guo, J, Ran, Y, Wang, S, Zhu, Z, Zhou, J, Hu, X, Xu, Z. Heihe Watershed Allied Telemetry Experimental Research (HiWATER): Scientific Objectives and Experimental Design. *Bulletin of the American Meteorological Society*. 2013 94(8):1145-1160.

Chollet, F, others. Keras: GitHub; 2015 2015.

Comune di Trento. 2021 [Available from: http://webapps.comune.trento.it/mapaccel/?project=generale&view=base.

Corona (satellite). Wikipedia: The Free Encyclopedia2021 [18.06.2021]. Available from: https://en.wikipedia.org/wiki/Corona_(satellite).

Cramer, M. The DGPF-Test on Digital Airborne Camera Evaluation Overview and Test Design. *Photogrammetrie - Fernerkundung - Geoinformation*. 2010 2010(2):73-82.

Digital Elevation Model. Wikipedia: The Free Encyclopedia2021 [18.06.2021]. Available from: https://en.wikipedia.org/wiki/Digital_elevation_model.

Eisenbeiß, H. UAV Photogrammetry: ETH Zurich; 2009.

Fischler, MA, Bolles, RC. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*. 1981 24(6):381-395.

Garson, JG. The Metric System of Identification of Criminals, as Used in Great Britain and Ireland. *The Journal of the Anthropological Institute of Great Britain and Ireland*. 1900 30:161-198.

Gerke, M, Nex, F, Remondino, F, Jacobsen, K, Kremer, J, Karel, W, Hu, H, Ostrowski, W. Orientation of Oblique Airborne Image Sets-Experiences from the ISPRS/EuroSDR Benchmark on Multi-Platform Photogrammetry. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2016 XLI-B1:185-191.

Gomarasca, MA. Basics of Geomatics: Springer Science & Business Media; 2009.

Goodfellow, I, Bengio, Y, Courville, A. Deep Learning: MIT press; 2016 2016.

Google Trends. 2021 [Available from: https://trends.google.com/trends/explore?date=all&q=support%20vector%20machine,convolutional%20neural%20network.

Gopi, S, Sathikumar, R, Madhu, N. Advanced Surveying: Total Station, GPS, GIS and Remote Sensing: Pearson; 2018.

Grilli, E, Menna, F, Remondino, F. A Review of Point Clouds Segmentation and Classification Algorithms. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2017 XLII-2/W3:339-344.

Ground Control Points for Drone Mapping. 2021 [17.06.2021]. Available from: https://www.groundcontrolpoints.com/mapping-contour-lines-using-gcps.

Gruen, A. Development and Status of Image Matching in Photogrammetry. *The Photogrammetric Record*. 2012 27(137):36-57.

Guo, Y, Wang, H, Hu, Q, Liu, H, Liu, L, Bennamoun, M. Deep Learning for 3D Point Clouds: A Survey. *IEEE Trans Pattern Anal Mach Intell*. 2020 PP:1-1.

Haala, N. The Landscape of Dense Image Matching Algorithms. 54th Photogrammetric Week; 9-13 September 20132013. p. 271-284.

Hackel, T, Savinov, N, Ladicky, L, Wegner, JD, Schindler, K, Pollefeys, M. Semantic3d.Net: A New Large-Scale Point Cloud Classification Benchmark. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2017 IV-1/W1:91-98.

Hackel, T, Wegner, JD, Schindler, K. Fast Semantic Segmentation of 3D Point Clouds with Strongly Varying Density. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2016 III-3:177-184.

Han, Y, Qin, R, Huang, X. Assessment of Dense Image Matchers for Digital Surface Model Generation Using Airborne and Spaceborne Images – An Update. *The Photogrammetric Record*. 2020 35(169):58-80.

Harris, CR, Millman, KJ, van der Walt, SJ, Gommers, R, Virtanen, P, Cournapeau, D, Wieser, E, Taylor, J, Berg, S, Smith, NJ, Kern, R, Picus, M, Hoyer, S, van Kerkwijk, MH, Brett, M, Haldane, A, Del Rio, JF, Wiebe, M, Peterson, P, Gerard-Marchant, P, Sheppard, K, Reddy, T, Weckesser, W, Abbasi, H, Gohlke, C, Oliphant, TE. Array Programming with NumPy. *Nature*. 2020 585(7825):357-362.

Heipke, C, Rottensteiner, F. Deep learning for geometric and semantic tasks in photogrammetry and remote sensing. *Geo-spatial Information Science*. 2020 23(1):10-19.

Henderson, RM, Clark, KB. Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Science Quarterly*. 1990 35(1):9-30.

Hermosilla, P, Ritschel, T, Vázquez, P-P, Vinacua, À, Ropinski, T. Monte Carlo Convolution for Larning on Non-Uniformly Sampled Point Clouds. *ACM Trans Graph*. 2018 37(6):Article 235.

Hofmann-Wellenhof, B, Lichtenegger, H, Collins, J. Global Positioning System: Theory and Practice: Springer Science & Business Media; 2012.

Huang, R, Xu, Y, Hong, D, Yao, W, Ghamisi, P, Stilla, U. Deep point embedding for urban classification using ALS point clouds: A new perspective from local to global. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2020 163:62-81.

Huang, X, Cao, R, Cao, Y. A Density-Based Clustering Method for the Segmentation of Individual Buildings from Filtered Airborne LiDAR Point Clouds. *Journal of the Indian Society of Remote Sensing*. 2018 47(6):907-921.

Illingworth, J, Kittler, J. A Survey of the Hough Transform. *Computer Vision, Graphics, and Image Processing*. 1988 44(1):87-116.

ISO. ISO/IEC 14882-2014: Information Technology–Programming Languages–C++. *ISO Working Group (Dec 2014)(cit on p 76)*. 2014.

Kanezaki, A, Matsushita, Y, Nishida, Y, editors. RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2018 2018/06/18/23.

Konecny, G. Geoinformation: Remote Sensing, Photogrammetry and Geographic Information Systems, Second Edition (2nd ed.): cRc Press; 2014.

Koundinya, S, Sharma, H, Sharma, M, Upadhyay, A, Manekar, R, Mukhopadhyay, R, Karmakar, A, Chaudhury, S, editors. 2D-3D CNN Based Architectures for Spectral Reconstruction from RGB Images. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW); 2018 2018/06/18/22.

Kraus, K. Photogrammetry: de Gruyter; 2007.

Landrieu, L, Raguet, H, Vallet, B, Mallet, C, Weinmann, M. A structured regularization framework for spatially smoothing semantic labelings of 3D point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2017 132:102-118.

Laupheimer, D, Shams Eddin, MH, Haala, N. On the Association of Lidar Point Clouds and Textured Meshes for Multi-Modal Semantic Segmentation. *ISPRS Ann Photogramm Remote Sens Spatial Inf Sci*. 2020 V-2-2020:509-516.

Leica CityMapper-2 Hybrid Airborne Sensor. 2021 [Available from: https://leica-geosystems.com/en-us/products/airborne-systems/leica-citymapper-2.

Li, N, Liu, C, Pfeifer, N. Improving LiDAR classification accuracy by contextual label smoothing in post-processing. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2019 148:13-31.

Li, W, Wang, F-D, Xia, G-S. A Geometry-Attentional Network for ALS Point Cloud Classification. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2020a 164:26-40.

Li, X, Wang, L, Wang, M, Wen, C, Fang, Y. DANCE-NET: Density-aware convolution networks with context encoding for airborne LiDAR point cloud classification. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2020b 166:128-139.

Lin, T, Goyal, P, Girshick, R, He, K, Dollár, P, editors. Focal Loss for Dense Object Detection. 2017 IEEE International Conference on Computer Vision (ICCV); 2017 22-29 Oct. 2017.

Linder, W. Digital Photogrammetry: Springer; 2009.

Liu, L, Ouyang, W, Wang, X, Fieguth, P, Chen, J, Liu, X, Pietikäinen, M. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*. 2020 128(2):261-318.

Liu, W, Sun, J, Li, W, Hu, T, Wang, P. Deep Learning on Point Clouds and Its Application: A Survey. *Sensors (Basel)*. 2019 19(19).

Luhmann, T, Robson, S, Kyle, S, Boehm, J. Close-Range Photogrammetry and 3D Imaging: De Gruyter; 2019.

Madhulatha, TS. An Overview on Clustering Methods. *IOSR Journal of Engineering*. 2012 2:719-725.

Maling, DH. Coordinate Systems and Map Projections: Pergamon Press; 1992.

Maltezos, E, Doulamis, A, Doulamis, N, Ioannidis, C. Building Extraction From LiDAR Data Applying Deep Convolutional Neural Networks. *IEEE Geoscience and Remote Sensing Letters*. 2019 16(1):155-159.

Martín, A, Ashish, A, Paul, B, Eugene, B, Zhifeng, C, Craig, C, Greg, SC, Andy, D, Jeffrey, D, Matthieu, D, Sanjay, G, Ian, G, Andrew, H, Geoffrey, I, Michael, I, Jia, Y, Rafal, J, Lukasz, K, Manjunath, K, Josh, L, Dandelion, M, Rajat, M, Sherry, M, Derek, M, Chris, O, Mike, S, Jonathon, S, Benoit, S, Ilya, S, Kunal, T, Paul, T, Vincent, V, Vijay, V, Fernanda, V, Oriol, V, Pete, W, Martin, W, Martin, W, Yuan, Y, Xiaoqiang, Z. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems2015 2015.

Matrone, F, Grilli, E, Martini, M, Paolanti, M, Pierdicca, R, Remondino, F. Comparing Machine and Deep Learning Methods for Large 3D Heritage Semantic Segmentation. *ISPRS International Journal of Geo-Information*. 2020 9(9).

McGlone, C. Manual of Photogrammetry, Sixth Edition: ASPRS; 2013.

McKinney, W, editor Data Structures for Statistical Computing in Python2010.

Nickolls, J, Buck, I, Garland, M, Skadron, K. Scalable Parallel Programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*. 2008 6(2):40–53.

Niemeyer, J, Rottensteiner, F, Soergel, U. Contextual Classification of Lidar Data and Building Object Detection in Urban Areas. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2014 87:152-165.

Özdemir, E, Remondino, F. Classification of Aerial Point Clouds with Deep Learning. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2019 XLII-2/W13:103-110.

Özdemir, E, Remondino, F. Segmentation of 3D Photogrammetric Point Cloud for 3D Building Modeling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2018 XLII-4/W10:135-142.

Özdemir, E, Remondino, F, Golkar, A. Aerial Point Cloud Classification with Deep Learning and Machine Learning Algorithms. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2019a XLII-4/W18:843-849.

Özdemir, E, Remondino, F, Golkar, A. An Efficient and General Framework for Aerial Point Cloud Classification in Urban Scenarios. *Remote Sensing*. 2021 13(10).

Özdemir, E, Toschi, I, Remondino, F. A Multi-Purpose Benchmark for Photogrammetric Urban 3d Reconstruction in a Controlled Environment. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2019b XLII-1/W2:53-60.

Pârvu, IM, Özdemir, E, Remondino, F. Aerial Point Cloud Classification Using an Alternative Approach for the Dynamic Computation of K-Nearest Neighbors. *Journal of Applied Engineering Sciences*. 2020 10(2):155-162.

Pedregosa, F, Varoquaux, G, Gramfort, A, Michel, V, Thirion, B, Grisel, O, Blondel, M, Prettenhofer, P, Weiss, R, Dubourg, V, Vanderplas, J, Passos, A, Cournapeau, D, Brucher, M, Perrot, M, Duchesnay, É. Scikit-Learn: Machine Learning in Python. *J Mach Learn Res*. 2011 12(null):2825-2830.

Pelton, JN, Madry, S, Camacho-Lara, S. Handbook of Satellite Applications: Springer; 2017.

PlaidML. PlaidML 2019 [updated 2019. Available from: https://github.com/plaidml/plaidml.

Poli, D, Remondino, F, Angiuli, E, Agugiaro, G. Evaluation of Pleiades-1a Triplet on Trento Testfield. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2013 XL-1/W1:287-292.

Poli, D, Remondino, F, Angiuli, E, Agugiaro, G. Radiometric and Geometric Evaluation of GeoEye-1, WorldView-2 and Pléiades-1A Stereo Images for 3D Information Extraction. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2015 100:35-47.

Pulvirenti, L, Squicciarino, G, Fiori, E, Fiorucci, P, Ferraris, L, Negro, D, Gollini, A, Severino, M, Puca, S. An Automatic Processing Chain for Near Real-Time Mapping of Burned Forest Areas Using Sentinel-2 Data. *Remote Sensing*. 2020 12(4):674.

QGIS.org. QGIS Geographic Information System: QGIS Association; 2021 [Available from: http://www.qgis.org.

Qi, CR, Yi, L, Su, H, Guibas, LJ. Pointnet++: Deep Hierarchical Feature Learning On Point Sets In A Metric Space. *arXiv preprint arXiv:170602413*. 2017:5099-5108.

Qin, R. A Critical Analysis of Satellite Stereo Pairs for Digital Surface Model Generation and Matching Quality Prediction Model. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2019 154:139-150.

Qin, R, Tian, J, Reinartz, P. 3D change detection – Approaches and applications. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2016 122:41-56.

Reback, J, jbrockmendel, McKinney, W, Van den Bossche, J, Augspurger, T, Cloud, P, Hawkins, S, gfyoung, Sinhrks, Roeschke, M, Klein, A, Petersen, T, Tratner, J, She, C, Ayd, W, Hoefler, P, Naveh, S, Garcia, M, Schendel, J, Hayden, A, Saxton, D, Gorelli, ME, Shadrach, R, Jancauskas, V, McMaster, A, Li, F, Battiston, P, Seabold, S, attack68, Dong, K. pandas-dev/pandas: Pandas. Zenodo; 2020.

Riegl. RIEGL - Produktdetail 2021 [17.06.2021]. Available from: http://www.riegl.com/nc/products/terrestrial-scanning/produktdetail/product/scanner/30/.

Rottensteiner, F, Sohn, G, Jung, J, Gerke, M, Baillard, C, Benitez, S, Breitkopf, U. The ISPRS Benchmark on Urban Object Classification and 3D Building Reconstruction. *ISPRS Ann Photogramm Remote Sens Spatial Inf Sci*. 2012 I-3:293-298.

Rusu, RB, Cousins, S, editors. 3D is here: Point Cloud Library (PCL). 2011 IEEE International Conference on Robotics and Automation; 2011 2011/05/09/13.

Shan, J, Toth, CK. Topographic Laser Ranging and Scanning: Principles and Processing: CRC Press; 2018.

Stenz, U, Hartmann, J, Paffenholz, J-A, Neumann, I. High-Precision 3D Object Capturing with Static and Kinematic Terrestrial Laser Scanning in Industrial Applications—Approaches of Quality Assessment. *Remote Sensing*. 2020 12(2):290.

Stone, JE, Gohara, D, Shi, G. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science & Engineering*. 2010 12(3):66-73.

Su, H, Maji, S, Kalogerakis, E, Learned-Miller, E, editors. Multi-view Convolutional Neural Networks for 3D Shape Recognition. Proceedings of the IEEE international conference on computer vision; 2015 2015.

Tan, M, Le, Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: Kamalika C, Ruslan S, editors. Proceedings of the 36th International Conference on Machine Learning; Proceedings of Machine Learning Research*: PMLR*; 2019. p. 6105--6114.

Tapete, D, Cigna, F. Appraisal of Opportunities and Perspectives for the Systematic Condition Assessment of Heritage Sites with Copernicus Sentinel-2 High-Resolution Multispectral Imagery. *Remote Sensing*. 2018 10(4):561.

Thomas, H, Goulette, F, Deschaud, J, Marcotegui, B, LeGall, Y, editors. Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods. 2018 International Conference on 3D Vision (3DV); 2018 2018/09/05/8.

Thomas, H, Qi, CR, Deschaud, J-E, Marcotegui, B, Goulette, F, Guibas, LJ, editors. KPConv: Flexible and Deformable Convolution for Point Clouds2019 2019.

Toschi, I, Farella, EM, Welponer, M, Remondino, F. Quality-Based Registration Refinement of Airborne LiDAR and Photogrammetric Point Clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2021 172:160-170.

Trimble. Total Stations | Trimble Geospatial 2021 [17.06.2021]. Available from: https://geospatial.trimble.com/products-and-solutions/total-stations.

Urech, PRW, Dissegna, MA, Girot, C, Grêt-Regamey, A. Point Cloud Modeling as a Bridge Between Landscape Design and Planning. *Landscape and Urban Planning*. 2020 203:103903.

Varney, N, Asari, VK, Graehling, Q, editors. DALES: A Large-Scale Aerial LiDAR Data Set for Semantic Segmentation2020 2020/06//.

Verma, V, Aggarwal, RK. A comparative analysis of similarity measures akin to the Jaccard index in collaborative recommendations: empirical and theoretical perspective. *Social Network Analysis and Mining*. 2020 10(1):43.

Vosselman, G, Coenen, M, Rottensteiner, F. Contextual Segment-Based Classification of Airborne Laser Scanner Data. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2017 128:354-371.

Vosselman, G, Maas, H-G. Airborne and Terrestrial Laser Scanning: CRC Press; 2010.

Wang, X. Learning from Big Data with Uncertainty – editorial. *Journal of Intelligent & Fuzzy Systems*. 2015 28:2329-2330.

Wang, X, Zhao, Y, Pourpanah, F. Recent Advances in Deep Learning. *International Journal of Machine Learning and Cybernetics*. 2020 11(4):747-750.

Wang, Y, Sun, Y, Liu, Z, Sarma, SE, Bronstein, MM, Solomon, JM. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics*. 2019 38(5):1-12.

Weinmann, M, Jutzi, B, Mallet, C. Feature relevance assessment for the semantic interpretation of 3D point cloud data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2013 II-5/W2(W2):313-318.

Wen, C, Li, X, Yao, X, Peng, L, Chi, T. Airborne LiDAR Point Cloud Classification with Global-Local Graph Attention Convolution Neural Network. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2021 173:181-194.

Winiwarter, L, Mandlburger, G, Schmohl, S, Pfeifer, N. Classification of ALS Point Clouds Using End-to-End Deep Learning. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*. 2019 87(3):75-90.

Wold, S, Esbensen, K, Geladi, P. Principal Component Analysis. *Chemometrics and Intelligent Laboratory Systems*. 1987 2(1):37-52.

Xia, S, Xu, S, Wang, R, Li, J, Wang, G. Building Instance Mapping From ALS Point Clouds Aided by Polygonal Maps. *IEEE Transactions on Geoscience and Remote Sensing*. 2021:1-13.

Xie, N, Ras, G, van Gerven, M, Doran, D. Explainable Deep Learning: A Field Guide for the Uninitiated. *arXiv preprint arXiv:200414545*. 2020a.

Xie, Y, Tian, J, Zhu, XX. Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation. *IEEE Geoscience and Remote Sensing Magazine*. 2020b 8(4):38-59.

Yan, Y, Yan, H, Guo, J, Dai, H. Classification and Segmentation of Mining Area Objects in Large-Scale Spares Lidar Point Cloud Using a Novel Rotated Density Network. *ISPRS International Journal of Geo-Information*. 2020 9(3).

Ye, Z, Xu, Y, Huang, R, Tong, X, Li, X, Liu, X, Luan, K, Hoegner, L, Stilla, U. LASDU: A Large-Scale Aerial LiDAR Dataset for Semantic Labeling in Dense Urban Areas. *ISPRS International Journal of Geo-Information*. 2020 9(7).

Yousefhussien, M, Kelbe, DJ, Ientilucci, EJ, Salvaggio, C. A Multi-Scale Fully Convolutional Network for Semantic Labeling of 3D Point Clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2018 143:191-204.

Zhang, F, Guan, C, Fang, J, Bai, S, Yang, R, Torr, PHS, Prisacariu, V, editors. Instance Segmentation of LiDAR Point Clouds. 2020 IEEE International Conference on Robotics and Automation (ICRA); 2020 31 May-31 Aug. 2020.

Zhang, J, Lin, X, Ning, X. SVM-Based Classification of Segmented Airborne LiDAR Point Clouds in Urban Areas. *Remote Sensing*. 2013 5(8):3749-3775.

Zhang, Y, Yang, W, Liu, X, Wan, Y, Zhu, X, Tan, Y. Unsupervised Building Instance Segmentation of Airborne LiDAR Point Clouds for Parallel Reconstruction Analysis. *Remote Sensing*. 2021 13(6):1136.

Appendices A

In this appendix, the codes of the developed framework are shared, including feature extraction tool, deep learning tool, accuracy assessment tool, and instance segmentation tool.

The codes shared in this thesis are not meant to be used for any purpose without an official agreement with Skoltech and Bruno Kessler Foundation.

A.1. Feature Extraction Tool

```
// feature extraction tool (C++)
// author emre özdemir
// multi-threading implementation: alessandro torresani

#define PCL_NO_PRECOMPILE
//#include <boost/make_shared.hpp>
#include <chrono>
#include <cmath>
#include <Eigen/Core>
#include <liblas/liblas.hpp> //laz implementation
#include <pcl/common/pca.h>
#include <pcl/features/normal_3d_omp.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/io/ply_io.h>
#include <pcl/io/pcd_io.h>
#include <pcl/ModelCoefficients.h>
#include <pcl/point_types.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/search/kdtree.h>
```

```
#include <pcl/segmentation/sac_segmentation.h>
#include <thread>
#include <time.h>
#include <vector>
#include <math.h>

// global variables for multi-scale feature extraction
const int scale = 3; //5
const int scale_multiplier = 2; //2
int numpt = 3;
int scale_vals[scale+1] = {0, 6, 20, 38};

//argv 1 input cloud
//argv 2 project name
//argv 3 project path
//argv 4 number of cores

// custom point cloud type, to keep the necessary features
struct Point_custom
{
    PCL_ADD_POINT4D;
    float class_id;
    float is_train; //
    float ei_va_3[scale*3]; //eigen values 1-2-3
    float shp_ft_3[scale*3]; // elevation change, local
planarity, vertical angle
    float ei_ft_3[scale*3]; //eigen features in order:
surface variation, sphericity, omnivariance
```

```
    float ei_nrmls_3[scale*3]; //last 3 eigen vectors from
each scale, the first six are ignored (these are the
normals..)
    float h6; // height above "ground"
    float intensity;
    float number_of_returns;
    float return_num;
    PCL_ADD_NORMAL4D;
    PCL_ADD_UNION_RGB;
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW // make sure our new
allocators are aligned
} EIGEN_ALIGN16;                    // enforce SSE padding
for correct memory alignment

POINT_CLOUD_REGISTER_POINT_STRUCT
(Point_custom,
 (float, x, x)
 (float, y, y)
 (float, z, z)
 (float, class_id, class_id)
 (float, is_train, is_train)
 (float[scale*3], ei_va_3, ei_va_3)
 (float[scale*3], shp_ft_3, shp_ft_3)
 (float[scale*3], ei_ft_3, ei_ft_3)
 (float[scale*3], ei_nrmls_3, ei_nrmls_3)
 (float, h6, h6)
 (float, intensity, intensity)
 (float, number_of_returns, number_of_returns)
 (float, return_num, return_num)
```

```
 (float, normal_x, normal_x) //to receive the original
normals if exists
 (float, normal_y, normal_y)
 (float, normal_z, normal_z)
 (float, rgb, rgb) //to receive the colors
)

// an object to keep feature group names and number of
features in each group
struct fields_struct
{
    const std::string category[4] = {" ei_va_", " shp_ft_",
" ei_ft_", " ei_nrmls_"};
    const int features[4] = {3,3,3,3};
} fields;

// functions //
// to split point cloud indices for multi-threading
std::vector<std::vector<int>>  split_to_indices(const int
num_threds, size_t size_pc, size_t elemPerList);

// compute shape features:
void computeShapeFeatures4(int pt_idx,
pcl::PointCloud<Point_custom>::Ptr pcl_in,
pcl::search::Search<Point_custom>::Ptr kdTree,
pcl::SACSegmentation<Point_custom> plnFitter,
pcl::PCA<Point_custom> PCA);

// compute eigen features (covariance features)
```

```
void ComputeMyEigenFeatures3(int pt_idx,
pcl::PointCloud<Point_custom>::Ptr pcl_in);

// compute height above groundd
void computeHF5 (int pt_idx,
pcl::PointCloud<Point_custom>::Ptr pcl_in,
pcl::search::Search<Point_custom>::Ptr tree, float minz);

// to run all the feature computations, this function is
triggered per-thread
void RunComputeMyFeatures(std::vector<int> pointIndices,
pcl::PointCloud<Point_custom>::Ptr pcl_in,
pcl::search::Search<Point_custom>::Ptr tree, float minz);

// prepare a header line based on the available sensor
feautres
void prepare_head5(pcl::PointCloud<Point_custom>::Ptr
cloud, std::string *heady, bool *rgb, bool *nxyz, bool
*intensiti, bool *numret, bool *retnum);

// brings scalar fields from one point cloud to another
with the nearest neighbor points
void bring_sf(pcl::PointCloud<Point_custom>::Ptr
cloud_missing, pcl::PointCloud<Point_custom>::Ptr
cloud_full, size_t missing_size,
pcl::search::Search<Point_custom>::Ptr tree, bool rgb, bool
nxyz, bool intensiti, bool numret, bool retnum);

//reads laz files into a pcl point cloud object
```

```
void laz2pcl(std::string fname,
pcl::PointCloud<Point_custom>::Ptr cloud);

int main (int argc, char** argv)
{
    std::cout<<"feature extraction tool is
running"<<std::endl;
    std::string infilename = argv[1]; // input point cloud
    std::string suffix = argv[2]; // project name, suffix
    std::string prj_dir = argv[3]; // project directory for
outputs
    const int numberOfThreads = std::stoi(argv[4]);
    // this is the downsampled cloud
    pcl::PointCloud<Point_custom>::Ptr pcl_in (new
pcl::PointCloud<Point_custom>);

    // laz file will be read into this object
    pcl::PointCloud<Point_custom>::Ptr pcl_in0 (new
pcl::PointCloud<Point_custom>);

    // downsampling object
    pcl::VoxelGrid<Point_custom> grid;

    // search kd-tree object, multi-thread safe,
downsampled cloud will be passed to this
    pcl::search::Search<Point_custom>::Ptr tree =
pcl::shared_ptr<pcl::search::Search<Point_custom> > (new
pcl::search::KdTree<Point_custom>);

    // string holding the header line, will be filled
```

```
    std::string heady;

    // bool variables for header object
    bool rgb, nxyz, intensiti, numret, retnum;

    // output file
    std::ofstream outfile;

    // report file
    std::ofstream repout;
    repout.open(prj_dir+"/" + suffix +
"_feature_extraction_report.txt");

    //start chronometer
    std::chrono::steady_clock::time_point begin =
std::chrono::steady_clock::now();

    //read laz file
    laz2pcl(infilename, pcl_in0);

    //report file reading time
    std::chrono::steady_clock::time_point end =
std::chrono::steady_clock::now();
    repout << "file reading time: " <<
std::chrono::duration_cast<std::chrono::microseconds>(end -
begin).count()*0.000001 << "\n";

    //get the original point cloud size (number of points)
and export to report
    size_t pc_size = pcl_in0->points.size();
```

```
    repout << "number of points before downsampling: " <<
pc_size << "\n";

    // header line of the output file prepared here
    prepare_head5(pcl_in0, &heady, &rgb, &nxyz, &intensiti,
&numret, &retnum);

    // reset the chrono before downsampling
    begin = std::chrono::steady_clock::now();

    //gridding (downsampling the data)
    grid.setLeafSize (0.75f, 0.75f, 0.75f);
    grid.setInputCloud (pcl_in0);
    grid.filter (*pcl_in);

    // report time ıf downsampling
    end = std::chrono::steady_clock::now();
    repout << "downsampling time: " <<
std::chrono::duration_cast<std::chrono::microseconds>(end -
begin).count()*0.000001 << "\n";
    // reset the chrono before scalar field bringing
    begin = std::chrono::steady_clock::now();

    //need to get pc_size and build the tree after
gridding, also need to fill missing scalar fields (of
sensor) after downsampling
    pc_size = pcl_in->points.size();
    bring_sf(pcl_in, pcl_in0, pc_size, tree, rgb, nxyz,
intensiti, numret, retnum);
    tree->setInputCloud(pcl_in);
```

```
    //clear the original point cloud
    pcl_in0->clear();

    // prepare a list of lists, each list contains the
point indices, number of lists = number of threads
    size_t elemPerList = pc_size / numberOfThreads;
    std::vector<std::vector<int>> pointThreadIndices;
    pointThreadIndices = split_to_indices(numberOfThreads,
pc_size, elemPerList);

    // report time for scalar field bringing
    end = std::chrono::steady_clock::now();
    repout << "scalar field bringing time: " <<
std::chrono::duration_cast<std::chrono::microseconds>(end -
begin).count()*0.000001 << "\n";
    // reset the chrono before feature extraction
    begin = std::chrono::steady_clock::now();

    // get global min-z for height above ground feature
    float hmin = 9999999.0f;
    for (size_t i=0; i<pc_size; ++i)
    {
        hmin = hmin > pcl_in->points[i].z ? pcl_in-
>points[i].z : hmin;
    }

    // Compute features on multiple threads
    std::vector<std::thread> threads;
    for (int i=0; i<numberOfThreads; ++i)
```

```
        threads.push_back(std::thread(RunComputeMyFeatures,
pointThreadIndices[i], pcl_in, tree, hmin)); //ADD NEW
THINGS HERE

    // wait for all threads to complete
    for (int i=0; i<numberOfThreads; ++i)
        threads[i].join();

    end = std::chrono::steady_clock::now();
    repout << "feature extraction time: " <<
std::chrono::duration_cast<std::chrono::microseconds>(end -
begin).count()*0.000001 << "\n";
    begin = std::chrono::steady_clock::now();

    // export the point cloud to a csv file, checking the
available scalar fields
    outfile.open(prj_dir + "/" + suffix +
"_features_cloud.txt");
    outfile << heady;
    int s=0; // counter of the scales
    int k=0; // counter of feature in each feature group
for each scale
    std::string str; //each line is prepared into this
object, then printed for some speed gain
    for (size_t i = 0; i < pc_size; ++i)
    {
        str =
        (std::to_string(pcl_in->points[i].x)             +"
"+ std::to_string(pcl_in->points[i].y)        +" "+
std::to_string(pcl_in->points[i].z) + " "
```

```
        + std::to_string(pcl_in->points[i].class_id)
+" ");//+ pcl_in->points[i].is_train +" ";
        for (s=0; s<scale; ++s) {
            for (k = 0; k<fields.features[0] ; ++k)  // k
for each feature no under the feature category, there are 3
features under categories:  i=0 and i=3
            str += (std::to_string(pcl_in-
>points[i].ei_va_3[s*fields.features[0]+k]) + " "); //feat
0 is ei_va_3
        }

        for (s=0; s<scale; ++s) {
            for (k = 0; k<fields.features[1] ; ++k)  // k
for each feature no under the feature category, there are 3
features under categories:  i=0 and i=3
            str += (std::to_string(pcl_in-
>points[i].shp_ft_3[s*fields.features[1]+k]) + " "); //feat
1 is shp_ft_3
        }

        for (s=0; s<scale; ++s) {
            for (k = 0; k<fields.features[2] ; ++k)  // k
for each feature no under the feature category, there are 3
features under categories:  i=0 and i=3
            str += (std::to_string(pcl_in-
>points[i].ei_ft_3[s*fields.features[2]+k]) + " "); //feat
2 is ei_ft_3
        }

        for (s=0; s<scale; ++s) {
```

```
            for (k = 0; k<fields.features[3] ; ++k)  // k
for each feature no under the feature category, there are 3
features under categories:  i=0 and i=3
            str += (std::to_string(pcl_in-
>points[i].ei_nrmls_3[s*fields.features[3]+k]) + " ");
//feat 3 is ei_nrmls_3
        }
        str += (std::to_string(pcl_in->points[i].h6) +" ");

        if (intensiti)
            str += (std::to_string(pcl_in-
>points[i].intensity) +" "); // 65536.0f //
normalization&standardization computations will be held in
python
        if (numret)
            str += (std::to_string(pcl_in-
>points[i].number_of_returns) +" ");
        if (retnum)
            str += (std::to_string(pcl_in-
>points[i].return_num) +" ");

        if (nxyz)
            str += (std::to_string(pcl_in-
>points[i].normal_x)   +" "+ std::to_string(pcl_in-
>points[i].normal_y)   +" "+  std::to_string(pcl_in-
>points[i].normal_z)  +" ");
        if (rgb)
            str += (std::to_string(pcl_in->points[i].r)  +"
"+ std::to_string(pcl_in->points[i].g) +" "+
std::to_string(pcl_in->points[i].b) +" ");
```

```
        // normalization&standardization computations will
be held in python /255.f
        str += "\n";
        outfile << str;
        str.clear();
    }
    outfile.close();
    end = std::chrono::steady_clock::now();
    repout << "file export time: " <<
std::chrono::duration_cast<std::chrono::microseconds>(end -
begin).count()*0.000001;
    repout.close();
    std::cout << "Done!" << std::endl;
}

void RunComputeMyFeatures(std::vector<int> pointIndices,
pcl::PointCloud<Point_custom>::Ptr pcl_in,
pcl::search::Search<Point_custom>::Ptr tree, float minz)
{
    // this function is triggered for each thread.
    // plane fitting (sac segmentation) object passed to
each thread defined here
    pcl::SACSegmentation<Point_custom> plnFitter;
    plnFitter.setModelType (pcl::SACMODEL_PLANE);
    plnFitter.setMaxIterations(2);
    plnFitter.setDistanceThreshold(1000);
    plnFitter.setMethodType (pcl::SAC_RANSAC);
    plnFitter.setInputCloud(pcl_in);

    //principal component analysis object defined here
```

```
    pcl::PCA<Point_custom> PCA;
    PCA.setInputCloud(pcl_in);

    // for each thread a point indices vector is passed,
loop below triggered for each point
    for (int i : pointIndices)
    {
        computeShapeFeatures4(i, pcl_in, tree, plnFitter,
PCA);
        ComputeMyEigenFeatures3(i, pcl_in);
        computeHF5(i, pcl_in, tree, minz);
    }
}

void laz2pcl (std::string fname,
pcl::PointCloud<Point_custom>::Ptr cloud)
{
    //to be used as laz has 16-bit color coding.
    float color_bitter = 256.0f/65536.0f;

    // reading laz files into a pcl object
    std::ifstream ifs;
    ifs.open(fname, std::ios::in | std::ios::binary);
    liblas::ReaderFactory f;
    liblas::Reader reader = f.CreateWithStream(ifs);
    liblas::Header const& header = reader.GetHeader();

    // pcl object setting
    cloud->width  = header.GetPointRecordsCount();
    cloud->height = 1;
```

```
    cloud->points.resize(cloud->width);
    int count = 0;

    // any other field to be read from the laz file must be
included in this loop
    // each point is read one by one from the laz object
with GetPoint()
    while (reader.ReadNextPoint())
    {
        liblas::Point const& p = reader.GetPoint();
        cloud->points[count].x =
static_cast<float>(p.GetX());
        cloud->points[count].y =
static_cast<float>(p.GetY());
        cloud->points[count].z =
static_cast<float>(p.GetZ());

        cloud->points[count].r =
static_cast<float>(p.GetColor().GetRed()*color_bitter);
//color_bitter = 256.0f/65536.0f
        cloud->points[count].g =
static_cast<float>(p.GetColor().GetGreen()*color_bitter);
        cloud->points[count].b =
static_cast<float>(p.GetColor().GetBlue()*color_bitter);
        cloud->points[count].number_of_returns =
static_cast<float>(p.GetNumberOfReturns());
        cloud->points[count].return_num =
static_cast<float>(p.GetReturnNumber());
        cloud->points[count].class_id =
static_cast<float>(p.GetClassification().GetClass());
```

```
        cloud->points[count].intensity =
static_cast<float>(p.GetIntensity());
        ++count;
    }
}

void computeShapeFeatures4(int pt_idx,
pcl::PointCloud<Point_custom>::Ptr pcl_in,
pcl::search::Search<Point_custom>::Ptr kdTree,
pcl::SACSegmentation<Point_custom> plnFitter,
pcl::PCA<Point_custom> PCA)
{
    // shape features computations

    // plane coefficients and indices objects
    pcl::ModelCoefficients::Ptr plne_cff (new
pcl::ModelCoefficients);
    pcl::PointIndices::Ptr pln_indc (new
pcl::PointIndices);

    // nearest-neighbor search tool outputs into these
vectors
    std::vector<int> neighborsIndices; // Indices of the
points belonging to the neighborhood
    std::vector<int> neighbors; // Indices of the points
belonging to the neighborhood
    std::vector<float> neighborsDistances; // Distance of
the neighborhood points from srcPoints
```

```
    // planarity feature starts from 0, each point's
distance to plane to be added to this variable
    float tempPLN = 0.0f;

    // pca output objects
    Eigen::Matrix3f eigenvectors; // Eigenvectors
    Eigen::Vector3f eigenvalues; // Eigenvalues

    //hmin&hmax values for local elevation change feature
    float hmint = 999999.0f;
    float hmaxt = -999999.0f;

    // kdtree search is triggered for the largest scale
    kdTree->nearestKSearch(pcl_in->points[pt_idx],
scale_vals[scale], neighborsIndices, neighborsDistances);

    // using the scales vector (global variable), feature
extraction is done from smallest scale to largest
    for (int s=0; s<scale; ++s)
    {
        for(size_t mm = scale_vals[s]; mm <
scale_vals[s+1]; ++mm)
        {
            // indices for each scale is passed to another
vector, this will be passed to plane fitting object and pca
object
            neighbors.push_back(neighborsIndices[mm]);
            hmint = hmint > pcl_in-
>points[neighborsIndices[mm]].z ? pcl_in-
>points[neighborsIndices[mm]].z : hmint;
```

```
            hmaxt = hmaxt < pcl_in-
>points[neighborsIndices[mm]].z ? pcl_in-
>points[neighborsIndices[mm]].z : hmaxt;
        }

        // set indices for pca and plane fitting objects

PCA.setIndices(pcl::make_shared<std::vector<int>>(neighbors
));

plnFitter.setIndices(pcl::make_shared<std::vector<int>>(nei
ghbors));

        //get eigenvalues and eigenvectors from pca
        eigenvalues = PCA.getEigenValues();
        eigenvectors = PCA.getEigenVectors();

        //pass eigenvalues to eigenvalues feature
        pcl_in-
>points[pt_idx].ei_va_3[s*fields.features[0]+0] =
eigenvalues(0);
        pcl_in-
>points[pt_idx].ei_va_3[s*fields.features[0]+1] =
eigenvalues(1);
        pcl_in-
>points[pt_idx].ei_va_3[s*fields.features[0]+2] =
eigenvalues(2);

        // if z-component of the vector is negative, revert
the vector
```

```
        if (  eigenvectors(8) < 0)
        {
            eigenvectors(6) *= -1.0f;
            eigenvectors(7) *= -1.0f;
            eigenvectors(8) *= -1.0f;
        }

        // pass last 3 eigen vectors to eigen normals
feature
        pcl_in-
>points[pt_idx].ei_nrmls_3[s*fields.features[3]+0] =
eigenvectors(6);
        pcl_in-
>points[pt_idx].ei_nrmls_3[s*fields.features[3]+1] =
eigenvectors(7);
        pcl_in-
>points[pt_idx].ei_nrmls_3[s*fields.features[3]+2] =
eigenvectors(8);

        // fit the plane and check if it failed,
        plnFitter.segment(*pln_indc, *plne_cff);
        tempPLN = 0.0f;
        if ( isnan(plne_cff->values[0]) || isnan(plne_cff-
>values[1]) || isnan(plne_cff->values[2]) ||
isnan(plne_cff->values[3]) )
        {
            //check if plane fitting went well, if not, put
0 for plane-based features.
            pcl_in-
>points[pt_idx].shp_ft_3[s*fields.features[1]+1] = 0.0f;
```

```
        }
        // if succeded compute the average of the
distances,
        else
        {
            for(int mm = 0; mm < scale_vals[s+1]; ++mm)
            {
                //sum the absolute distances from each
point to the best fit plane
                tempPLN += (abs(plne_cff->values[0]*pcl_in-
>points[neighborsIndices[mm]].x + plne_cff-
>values[1]*pcl_in->points[neighborsIndices[mm]].y +
plne_cff->values[2]*pcl_in->points[neighborsIndices[mm]].z
+ plne_cff->values[3]) / sqrt(pow(plne_cff->values[0],2.0)
+ pow(plne_cff->values[1],2.0) + pow(plne_cff-
>values[2],2.0)));
            }
            //local planarity shp_ft_3[1], take the average
            pcl_in-
>points[pt_idx].shp_ft_3[s*fields.features[1]+1] =
tempPLN/static_cast<float>(scale_vals[s+1]);
        }
        //elevation change shp_ft_3[0]
        pcl_in-
>points[pt_idx].shp_ft_3[s*fields.features[1]+0] = hmaxt-
hmint;

        //vertical angle calculation
```

```
        float result = acos(pcl_in-
>points[pt_idx].ei_nrmls_3[s*fields.features[3]+2]) *
200.0f / M_PI;
        // check if nan, make it 0 if it is, then pass the
value
        result = isnan(result) ? 0.0f : result;
        pcl_in-
>points[pt_idx].shp_ft_3[s*fields.features[1]+2] = result;
    }
}

void ComputeMyEigenFeatures3(int pt_idx,
pcl::PointCloud<Point_custom>::Ptr pcl_in)
{
    // compute eigen features based on the eigenvalues
    // eigen features in order: surface variation,
sphericity, omnivariance
    for (int i = 0; i<scale; ++i)
    {
        pcl_in-
>points[pt_idx].ei_ft_3[i*fields.features[2]] = (pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+0] - pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+1]) / pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+0];
        pcl_in-
>points[pt_idx].ei_ft_3[i*fields.features[2]+1] = pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+2] / pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+0];
        pcl_in-
>points[pt_idx].ei_ft_3[i*fields.features[2]+2] =
```

```
std::cbrt(pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+2] * pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+1] * pcl_in-
>points[pt_idx].ei_va_3[i*fields.features[0]+0]);
    }
}

void computeHF5 (int pt_idx,
pcl::PointCloud<Point_custom>::Ptr pcl_in,
pcl::search::Search<Point_custom>::Ptr tree, float minz)
{
    // Indices of the points belonging to the neighborhood
    std::vector<int> neighborsIndices;
    // Distance of the neighborhood points from srcPoints
    std::vector<float> neighborsDistances;
    //locally lowest point around the point of interest
    float lowest;

    // z stays the same as pre defined, 1 for lowest, 2 for
highest points in the cloud
    Point_custom temp_point1; //from min z
    temp_point1.x = pcl_in->points[pt_idx].x;
    temp_point1.y = pcl_in->points[pt_idx].y;
    temp_point1.z = minz;

    // get the lowest z within the neighbourhood
    tree->nearestKSearch(temp_point1, 1, neighborsIndices,
neighborsDistances);
    temp_point1.z = pcl_in->points[neighborsIndices[0]].z;
```

```
    tree->nearestKSearch(temp_point1, 1, neighborsIndices,
neighborsDistances);
    lowest = pcl_in->points[neighborsIndices[0]].z;

    // pass the difference as height above ground
    pcl_in->points[pt_idx].h6 = pcl_in->points[pt_idx].z -
lowest;
}

std::vector<std::vector<int>> split_to_indices(const int
num_threds, size_t size_pc, size_t elemPerList)
{
    std::vector<std::vector<int>> pointThreadIndices;
    std::vector<int> v;
    for (int i=0; i < size_pc; ++i)
    {
        v.push_back(i);
        if (pointThreadIndices.size() == num_threds - 1)
        {  // Last chunk case (to handle <numberOfPoints> %
numberOfThreads != 0)
            if (i == size_pc - 1)
            {   // Last point case
                pointThreadIndices.push_back(v);
                v.clear();
            }
        }
        else
        {
            if (((i+1) % elemPerList) == 0)
            {   // Split point
```

```
                pointThreadIndices.push_back(v);
                v.clear();
            }
        }
    }
    return pointThreadIndices;
}

void prepare_head5(pcl::PointCloud<Point_custom>::Ptr
cloud, std::string *heady, bool *rgb, bool *nxyz, bool
*intensiti, bool *numret, bool *retnum)
{
    std::string temp;
    *heady = "x y z class_id";// is_train"; //this part is
fixed
    for (int i  = 0; i<4; ++i) // i for each feature
category, there are 4 feature categories.
    {
        for (int j = 0; j<scale; ++j) //j is for scale,
defined globally
        {   // k for each feature no under the feature
category, there are 3 features under categories:  i=0 and
i=3
            for (int k = 0; k<fields.features[i] ; ++k)
            {   //feature_category_$scale_$feature_no
                temp = fields.category[i] +
std::to_string(j+1) + "_" + std::to_string(k+1);
                (*heady) += temp;
            }
        }
```

```
    }
    (*heady) += " height_6p";

    *intensiti = true;
    if (cloud->points[0].intensity == 0 && cloud-
>points[1].intensity == 0 && cloud->points[2].intensity ==
0)
        *intensiti = false;
    if (*intensiti==true)
        (*heady) += " intensity";

    *numret = true;
    if (cloud->points[0].number_of_returns == 0 && cloud-
>points[1].number_of_returns == 0 && cloud-
>points[2].number_of_returns == 0)
        *numret = false;
    if (*numret==true)
        (*heady) += " num_returns";

    *retnum = true;
    if (cloud->points[0].return_num == 0 && cloud-
>points[1].return_num == 0 && cloud->points[2].return_num
== 0)
        *retnum = false;
    if (*retnum==true)
        (*heady) += " retr_numb";

    *nxyz = true;
    if (cloud->points[0].normal_x == 0 && cloud-
>points[0].normal_y == 0 && cloud->points[0].normal_z == 0)
```

```
        *nxyz = false;
    if (*nxyz==true)
        (*heady) += " nx ny nz";

    *rgb = true;
    if ( cloud->points[0].r == 0 && cloud->points[0].g == 0
&& cloud->points[0].b == 0)
        *rgb = false;
    if (*rgb==true)
        (*heady) += " r g b";
    (*heady) += "\n";
}

void bring_sf(pcl::PointCloud<Point_custom>::Ptr
cloud_missing, pcl::PointCloud<Point_custom>::Ptr
cloud_full, size_t missing_size,
pcl::search::Search<Point_custom>::Ptr tree, bool rgb, bool
nxyz, bool intensiti, bool numret, bool retnum)
{
    for (size_t i = 0; i<missing_size; i++)
    {
        // create a temporary point from the point cloud
with missing scalar fields
        Point_custom tempPoint;
        tempPoint.x = cloud_missing->points[i].x;
        tempPoint.y = cloud_missing->points[i].y;
        tempPoint.z = cloud_missing->points[i].z;

        // search returns outputs into these vectors
        std::vector<int> pointIdxSearch;
```

```
        std::vector<float> pointSquaredDistance;

        // run the search, retrieve only the available
features
        tree->nearestKSearch(tempPoint, 1, pointIdxSearch,
pointSquaredDistance);
        {
            cloud_missing->points[i].class_id = cloud_full-
>points[pointIdxSearch[0]].class_id;
            if (rgb)
            {
                cloud_missing->points[i].r = cloud_full-
>points[pointIdxSearch[0]].r;
                cloud_missing->points[i].g = cloud_full-
>points[pointIdxSearch[0]].g;
                cloud_missing->points[i].b = cloud_full-
>points[pointIdxSearch[0]].b;
            }
            if (nxyz)
            {
                cloud_missing->points[i].normal_x =
cloud_full->points[pointIdxSearch[0]].normal_x;
                cloud_missing->points[i].normal_y =
cloud_full->points[pointIdxSearch[0]].normal_y;
                cloud_missing->points[i].normal_z =
cloud_full->points[pointIdxSearch[0]].normal_z;
            }
            if (intensiti)
            {
```

```
                cloud_missing->points[i].intensity =
cloud_full->points[pointIdxSearch[0]].intensity;
            }
            if (numret)
            {
                cloud_missing->points[i].number_of_returns
= cloud_full->points[pointIdxSearch[0]].number_of_returns;
            }
            if (retnum)
            {
                cloud_missing->points[i].return_num =
cloud_full->points[pointIdxSearch[0]].return_num;
            }
        }
    }
}
```

A.2. 2DCNN Classification Tool

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#@author: emrozdmr
# classication tool (Python 3)

# import the used libraries
import os
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.utils import class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
Flatten, LeakyReLU
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint
```

```
from tensorflow.keras.models import model_from_json
import tensorflow.keras.backend as K
from scipy.spatial import cKDTree
from sklearn.metrics import f1_score, accuracy_score,
confusion_matrix
import gc

#tf.function decoration makes it as-if tensorflow
functions, enables to run efficiently inside the model

@tf.function
def f1_loss(true, pred):
    # weighted f1 loss function, to experiment.
    ground_positives = K.sum(true, axis=0)       # = TP +
FN
    pred_positives = K.sum(pred, axis=0)         # = TP +
FP
    true_positives = K.sum(true * pred, axis=0)  # = TP
    precision = (true_positives + K.epsilon()) /
(pred_positives + K.epsilon())
    recall = (true_positives + K.epsilon()) /
(ground_positives + K.epsilon())
    f1 = 2 * (precision * recall) / (precision + recall +
K.epsilon())
    weighted_f1 = f1 * ground_positives /
K.sum(ground_positives)
    weighted_f1 = K.sum(weighted_f1)
    return 1 - weighted_f1 #for metrics, return only
'weighted_f1'
```

```
@tf.function
def f1_lossAVG(true, pred):
    #average f1 loss functions
    ground_positives = K.sum(true, axis=0)       # = TP +
FN
    pred_positives = K.sum(pred, axis=0)         # = TP +
FP
    true_positives = K.sum(true * pred, axis=0)  # = TP
    precision = (true_positives + K.epsilon()) /
(pred_positives + K.epsilon())
    recall = (true_positives + K.epsilon()) /
(ground_positives + K.epsilon())
    f1 = 2 * (precision * recall) / (precision + recall +
K.epsilon())
    f1 = K.mean(f1)
    return 1 - f1

@tf.function
def f1_acc(y_true, y_pred):
    #taken from old keras source code
    #f1 accuracy is used as metric
    true_positives = K.sum(K.round(K.clip(y_true * y_pred,
0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0,
1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0,
1)))
    precision = true_positives / (predicted_positives +
K.epsilon())
```

```
    recall = true_positives / (possible_positives +
K.epsilon())
    f1_val =
2.0*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

def get_data(mydataFrame):
    # parse the input pandas dataframe
    # output separated sub-parts of it
    labels = mydataFrame['class_id'].copy()
    del mydataFrame['class_id']
    data_array = mydataFrame.values
    coordi = data_array[:,0:3]
    return labels, data_array, coordi

def get_patches3_2d(dataframe, knn, num_feat):
    # the 2d patches to be passed to the network
    labels_arr, dataa, coordie = get_data(dataframe)
    #ckdtree is faster
    trie = cKDTree(coordie, leafsize=1)
    big_arr = np.zeros((labels_arr.shape[0], num_feat,
kk,1), dtype=np.float32)
    #labels_arr.shape[0] is number of points
    for x in range(labels_arr.shape[0]):
        #search for each point
        idx = trie.query(coordie[x,:],k=knn)
        #retrieve the neighboring points
        id_arr = idx[1]
        id_arr = np.sort(id_arr)
```

```
        #get the data for these nighboring points into the
2d array
        temp = dataa[id_arr,:]

        #patch-wise coordinates, centered aroind the point
of interest
        temp[:,0:3] = temp[:,0:3] - coordie[x,:]

        #divide coordinates to patch_max, replace nan
values with zeros, which may occur during division
        temp[:,0:3] = temp[:,0:3] / np.amax(temp[:,0:3],
axis=0)
        temp[:,0:3] = np.nan_to_num(temp[:,0:3])

        #clip coordinates  to [0,1], and sort by x and z
        temp[:,0:3] = np.clip(temp[:,0:3],0,1)
        temp = temp[temp[:,0].argsort(kind='mergesort'),:]
        temp = temp[temp[:,2].argsort(kind='mergesort'),:]
        temp = np.reshape(temp,(num_feat,knn,1))
        big_arr[x,:,:,:]=temp
    return big_arr, labels_arr, labels_arr.shape[0]

# input parameters

# root directory: the outputs will be there
root_dir = '/home/emrozdmr/projects/bordo_new_tf2/'

# input file
all_file_name = 'br6_f12p_features_cloud_normd.txt'
os.chdir(root_dir)
```

```
# project name
suffix = 'bordo_noth_2d'

#read csv file into a pandas object, force float32s
all_data = pd.read_csv(root_dir+all_file_name, sep="
",index_col=False, dtype=np.float32)

# sorting is fast and helps with the kdtree search
all_data = all_data.sort_values(by=['x'])

#in case a nan column exists, remote it
try: del all_data['nan']
except: print('no nan col')

#optional, remove some sensor features for experimenting
#del all_data['intensity'];  del all_data['retr_numb'];
del all_data['num_returns']
#del all_data['r'];  del all_data['g'];  del all_data['b']

#split training and validation data based on the is_train
train = all_data[all_data['is_train']==1].copy()
evald = all_data[all_data['is_train']==0].copy()

# remove is_train column and the original dataframe, they
are not needed anymore
del train['is_train'], evald['is_train'], all_data

# 15 points to create 2d matrices
kk=15
```

```
#number of features are dynamic, because of the availabe
sensor features
num_feat = train.shape[1] - 1

# get the tensor holind 2d arrays, class_id ground truths
and number of points for training and validation data
x_train3, my_lbls_arr, num_pts_train =
get_patches3_2d(train, kk, num_feat)  #get training data
x_eval, my_lbls_arr_evl, num_pts_eval =
get_patches3_2d(evald, kk, num_feat) #get validation data

#remove the dataframes, not necessary anymore, and force-
clean the memory
del train, evald
gc.collect()

# get number of classes and class weights
ua,uind=np.unique(my_lbls_arr,return_inverse=True)
counts=np.bincount(uind)
cls_wghs =
class_weight.compute_class_weight(class_weight='balanced',
classes=ua, y=my_lbls_arr)

#print class weights
print(ua)
print("weights: ", cls_wghs)
print("weights max: ",cls_wghs.max())
print("weights max: ",cls_wghs.min())
```

```
# push class weights into a dict, required by newer TF
versions
myclass_weights = {}
for x in range(cls_wghs.shape[0]): myclass_weights[x] =
cls_wghs[x]

# create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
input_shape=(num_feat,kk,1)))
model.add(BatchNormalization())
model.add(LeakyReLU())

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(MaxPooling2D(pool_size=(2, 2), padding='valid'))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(LeakyReLU())

model.add(Conv2D(256, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(LeakyReLU())

model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Flatten())
model.add(Dense(36, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(len(myclass_weights),
activation='softmax'))

# training parameters, optimizer definitions
my_epochs = 100
learning_rate = 0.001
decay_rate = learning_rate / my_epochs      # LearningRate
= LearningRate * 1/(1 + decay * epoch)
my_batch_size = 1024
sgd = SGD(learning_rate=learning_rate, decay=decay_rate,
nesterov=False, momentum=0.95 )

#compile the model with the specified, optimizer, loss, and
metric functions
model.compile(loss=f1_lossAVG, optimizer=sgd,
metrics=[f1_acc])

# export model structure to json
model_json = model.to_json()     # serialize model to JSON
with open(root_dir+suffix+"_2dcnn.json", "w") as json_file:
json_file.write(model_json)

# set callbacks for training, and force-clean memory before
the training
my_callbacks = [EarlyStopping(patience=15,
min_delta=0.0001, restore_best_weights=True),
```

```
ModelCheckpoint(filepath=suffix+'_model.h5',
save_best_only=True, verbose=1)]
gc.collect()

# run the training, export the losses and accuracies for
training and validation into my_history variable
my_history = model.fit(x_train3,
to_categorical(my_lbls_arr,
num_classes=len(myclass_weights)),
batch_size=my_batch_size,

validation_data=(x_eval,to_categorical(my_lbls_arr_evl,
num_classes=len(myclass_weights))), epochs=my_epochs,
                      class_weight=myclass_weights,
shuffle=True, callbacks=my_callbacks)

# export the traning history into a csv file
pd.DataFrame.from_dict(my_history.history).to_csv(root_dir+
suffix+'_2dcnn_history.csv',index=False)

#export model weights, clean history object
model.save_weights(root_dir+suffix+"_2dcnn_weights.h5")
# serialize weights to HDF5
del my_history



"""
## OPTIONAL BITS
```

```
#read validation data, and predict on it,
# as the coordinates are not kept above, it is necessary to
read again to get the coordinates for exporting the
classification results
all_data = pd.read_csv(root_dir+all_file_name, sep="
",index_col=False, dtype=np.float32, usecols=[0,1,2,3,4])
#del all_data['intensity'];  del all_data['retr_numb'];
del all_data['num_returns']
#del all_data['r'];  del all_data['g'];  del all_data['b']
all_data = all_data.sort_values(by=['x'])
evald = all_data[all_data['is_train']==0].copy()
del all_data
coords_eval = evald.iloc[:,0:3].values
del evald
predicted_eval =
model.predict(x_eval,batch_size=my_batch_size,verbose=1)
prd_evl = np.argmax(predicted_eval, axis=1)
res_eval = np.column_stack((coords_eval,prd_evl,
np.amax(predicted_eval, axis=1)))
np.savetxt(root_dir+suffix+'_clsfd_eval_probs.txt',res_eval
,fmt='%1.3f %1.3f %1.3f %d %1.3f')
del predicted_eval, res_eval, coords_eval
gc.collect()


# read another dataset to run prediction on
all_data =
pd.read_csv('/full/path/to/pointcloudwithfeatures.txt',
sep=" ",index_col=False, dtype=np.float32)
```

```
#del all_data['intensity'];  del all_data['retr_numb'];
del all_data['num_returns']
#del all_data['r'];  del all_data['g'];  del all_data['b']
all_data = all_data.sort_values(by=['x'])
coords_eval = all_data.iloc[:,0:3].values
x_eval, my_lbls_arr_evl, num_pts_eval =
get_patches3_2d(all_data, kk, num_feat)
del all_data
predicted_eval =
model.predict(x_eval,batch_size=my_batch_size,verbose=1)
prd_evl = np.argmax(predicted_eval, axis=1)
res_eval = np.column_stack((coords_eval,prd_evl,
np.amax(predicted_eval, axis=1)))
np.savetxt('/full/path/to/exported_classification_result.tx
t',res_eval,fmt='%1.3f %1.3f %1.3f %d %1.3f')
del predicted_eval, res_eval, coords_eval
gc.collect()
"""
```

A.3. Accuracy Assessment Tool

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# @author: emre özdemir

#imports
import sys
import pandas as pd
import numpy as np
import os
from sklearn.metrics import accuracy_score, f1_score,
confusion_matrix, jaccard_score

# accuracy assessment tool
# recieves a single file with ground truth and prediction
class ids, outputs the accuracy assessment report

# control the passed inputs, give a message if something is
wrong
assert len(sys.argv) == 2, print("Please give the file name
to the grount truth and prediction.\n \
          the text file is expected in order of x y z
class_id_gt class_id_predicted, can have a header line\n  \
```

```
          Example use of this tool:\n \
          'python acc_calc.py full/path/to/your_file.txt
'\n \
          Come back with the required file!")

# read the data, print number of points
all_data = pd.read_csv(sys.argv[1], sep="
",index_col=False, usecols=[3, 4])
print("Number of points ",all_data.shape[0])

# compute accuracy metrics
overall_acc = accuracy_score(all_data.iloc[:,0],
all_data.iloc[:,1])
f1_scores = f1_score(all_data.iloc[:,0],
all_data.iloc[:,1], average=None)
f1_score_weighted = f1_score(all_data.iloc[:,0],
all_data.iloc[:,1], average='weighted')
con_mat = confusion_matrix(all_data.iloc[:,0],
all_data.iloc[:,1])
iou_per_class = jaccard_score(all_data.iloc[:,0],
all_data.iloc[:,1], average=None)
iou_global = jaccard_score(all_data.iloc[:,0],
all_data.iloc[:,1], average='micro')
iou_weighted = jaccard_score(all_data.iloc[:,0],
all_data.iloc[:,1], average='weighted')

#export the report
report_fname = os.path.dirname(sys.argv[1]) + "/report_" +
os.path.basename(sys.argv[1])
np.set_printoptions(precision=4)
```

```
file = open(report_fname,"w")
file.write("F1 Scores per class\n\n")
file.write(str(f1_scores))
file.write("\n\nWeighted F1 Score\n\n")
file.write('{:.4f}'.format(f1_score_weighted) )
file.write("\n\nAverage F1 Score\n\n")
file.write('{:.4f}'.format(f1_scores.mean() ) )
file.write("\n\nIOU per class\n\n")
file.write(str(iou_per_class))
file.write("\n\nIOU Global\n\n")
file.write('{:.4f}'.format(iou_global ) )
file.write("\n\nIOU Weighted\n\n")
file.write('{:.4f}'.format(iou_weighted ) )
file.write("\n\nAverage IOU\n\n")
file.write('{:.4f}'.format(iou_per_class.mean() ) )
file.write("\n\nOverall Accuracy\n\n")
file.write('{:.4f}'.format(overall_acc) )
file.write("\n\nConfusion Matrix\n\n")
file.write(str(con_mat))
file.close()
print("Find the report next to the input file.")
```

A.4. Instance Segmentation Tool

```
// clusterings tool (C++)
// author emre Özdemir

#define PCL_NO_PRECOMPILE
#include <pcl/ModelCoefficients.h>
#include <pcl/point_types.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/features/normal_3d.h>
#include <pcl/search/kdtree.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/segmentation/extract_clusters.h>
#include <pcl/io/ply_io.h>

// argv[1] input file
// argv[2] output directory

// custom point cloud type for clusters
struct Point_custom
{
    PCL_ADD_POINT4D;
```

```
    float class_id;
    float cluster_id;
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW // make sure our new
allocators are aligned
} EIGEN_ALIGN16;                    // enforce SSE padding
for correct memory alignment

POINT_CLOUD_REGISTER_POINT_STRUCT
(Point_custom,
 (float, x, x)
 (float, y, y)
 (float, z, z)
 (float, class_id, class_id)
 (float, cluster_id, cluster_id)
)


int main (int argc, char** argv)
{
    std::cout<<"Euclidean clustering is
running"<<std::endl;
    // input file, ply
    std::string indir = argv[1];

    // output directory
    std::string outdir = argv[2];

    // point cloud object definition and reading with
control
```

```
    pcl::PointCloud<Point_custom>::Ptr cloud (new
pcl::PointCloud<Point_custom>);
    if ( pcl::io::loadPLYFile <Point_custom> (indir,
*cloud) == -1)
    {
        std::cout << "Cloud reading failed." << std::endl;
        return (-1);
    }

    std::cout << "File successfully read\nIt is processing,
hold on for a while..." << std::endl;

    // creating the kdtree object for the search method of
the extraction
    pcl::search::Search<Point_custom>::Ptr tree =
pcl::shared_ptr<pcl::search::Search<Point_custom> > (new
pcl::search::KdTree<Point_custom>);
    tree->setInputCloud (cloud);

    // vector for exported cluster indices
    std::vector<pcl::PointIndices> cluster_indices;

    // euclidean distance clustering object definition
    // and parameter setting
    pcl::EuclideanClusterExtraction<Point_custom> ec;
    ec.setClusterTolerance(1.50f); //3.5
    ec.setMinClusterSize(100);
    ec.setMaxClusterSize(1600000);
    ec.setSearchMethod(tree);
    ec.setInputCloud(cloud);
```

```
    ec.extract(cluster_indices);

    // counter to keep cluster_id
    int k = 1;

    //commented lines in the loop are useful for exporting
each cluster in individual filess
    for (std::vector<pcl::PointIndices>::const_iterator it
= cluster_indices.begin (); it != cluster_indices.end ();
++it)
    {
//        pcl::PointCloud<Point_custom>::Ptr cloud_cluster
(new pcl::PointCloud<Point_custom>);
        for (std::vector<int>::const_iterator pit = it-
>indices.begin (); pit != it->indices.end (); ++pit) {
            cloud->points[*pit].cluster_id = k;
//            cloud_cluster->points.push_back (cloud-
>points[*pit]);
        }

        ++k;
//        cloud_cluster->width = cloud_cluster->points.size
();
//        cloud_cluster->height = 1;
//        cloud_cluster->is_dense = true;
//        pcl::io::savePLYFileBinary(outdir + "/cluster_" +
std::to_string(k) + ".ply",*cloud_cluster);
    }
    //export the cloud with cluster_ids
```

```
    pcl::io::savePLYFileBinary(outdir + "/clustered_" +
std::to_string(k) + ".ply",*cloud);
    std::cout << "Done!" << std::endl;
}
```