Contributions to graph-based hierarchical analysis for images and 3D point clouds

Leonardo Gigli

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES ParisTech

Contributions to graph-based hierarchical analysis for images and 3D point clouds

Contributions à l'analyse hiérarchique basée sur des graphes pour les images et les nuages de poins 3D

Soutenue par

Leonardo Gigli

Le 23 Mars 2021

École doctorale n°621

Ingénierie des Systèmes, Matériaux, Mécanique, Energétique

Spécialité

Morphologie mathématique

Composition du jury :

Sébastien LEFEVRE Professeur, Université Bretagne-Sud	*Président*
Pierre SOILLE Scientific Officer HDR, Joint Research Centre, European Commission	*Rapporteur*
Antonio PLAZA Full Professor, Universidad de Extremadura	*Rapporteur*
Jesus ANGULO Directeur de Recherche, MINES ParisTech	*Examinateur*
Ravi KIRAN Docteur Ingénieur, Navya	*Examinateur*
Benjamin PERRET Professeur Associé, ESIEE Paris	*Examinateur*
Santiago VELASCO-FORERO Chargé de Recherche, MINES ParisTech	*Examinateur*
Beatriz MARCOTEGUI Professeur, MINES ParisTech	*Directeur de thèse*

MINES ParisTech | PSL

Ph.D. Thesis

Contributions to graph-based hierarchical analysis for images and 3D point clouds.

Ph.D. candidate: Leonardo Gigli

Advisors: Ph.D. Santiago Velasco-Forero
Prof. Beatriz Marcotegui

Acknowledgements

First and foremost, I want to thank my research advisors, Dr. Santiago Velasco-Forero and Prof. Beatriz Marcotegui for having accepting me as a student and for the incredible opportunity they gave me to work with them and to pursue the Ph.D. They guided me over the years on different topics of the research with a lot of patience, trying to help me to move forward when I was stuck. I would also like to thank Dr. B Ravi Kiran for the time spent helping me with numerous inspiring discussions and endless number of papers to read, making me understand the importance of applications in our work. Furthermore, I want to express my gratitude to Dr. Andrés Serna, who advised me in my first steps on the world of 3D point clouds during my master and without whom I would never have known about the CMM, Center for Mathematical Morphology.

Being part of the CMM has been an incredible, enriching experience. I would like to thank all the people I met during these years: Samy, Etienne, Petr, Bruno, Francois, Jose, Michel, Jesus. A special thanks goes to Anne-Marie for her help in dealing with the dreadful French bureaucracy and for organizing great moments of conviviality at CMM every time. I want also want to thank all the students in Fontainebleau with whom I shared this though but beautiful path: Eric, David, Amin, Theo, Elodie, Francois, Albane, Robin, Kaiwen, Sebastian, Martin, Mateus, Tarek, Tristan, Zyad, Aurelien, Laure, Marine, Jean, Mike et Ricardo.

Living abroad sometimes can be difficult and the distance from home can make us feel homesick. Luckily, in these moments I could count on the support of a small Italian community that helped to feel less missing from home. Thank you, Carlo, Sam, Sara, Paola, Marco, Ele, Yasmine, Annina, Fabio, Cinzia and Cecilia.

Finally, I am grateful to my parents and my family because they are my first supporters. They push me to do my best and grow as a man, even though this means moving away from them.

Last but not the least, I would like to thank my love Zoe, who supported me in this choice and moved with me to Paris. Much of this achievement is thanks to your encouragement, especially in the hard times.

Abstract

Graphs are powerful mathematical structures representing a set of objects and the underlying links between pairs of objects somehow related. They are becoming increasingly popular in data science in general and in particular in image or 3D point cloud analysis. Among the wide spectra of applications, they are involved in most of the hierarchical approaches. Hierarchies are particularly important because they allow us to efficiently organize the information required and to analyse the problems at different levels of detail. In this thesis, we address the following topics.

Many morphological hierarchical approaches rely on the Minimum Spanning Tree (MST). We propose an algorithm for MST computation in streaming based on a graph decomposition strategy. Thanks to this decomposition, larger images can be processed or can benefit from partial reliable information while the whole image is not completely available.

Thanks to recent developments, LiDAR can acquire large-scale and precise 3D point clouds. Many applications, such as infrastructure monitoring, urban planning, autonomous driving, precision forestry, environmental assessment, to cite a few, are under development nowadays. We introduce a ground-detection algorithm and compare it with the state of the art. Moreover, we study the impact of reducing the point cloud density with low-cost scanners, in the context of an autonomous driving application.

In many hierarchical methods, similarities between points are given as input. However, the metric used to compute similarities influences the quality of the results. Metric learning is a complementary tool that helps to improve the quality of hierarchies. We demonstrate the capabilities of these methods in two contexts. The first one, a texture classification of 3D surfaces, a task organized by SHREC'20 international challenge (ranked second). The second one learning the similarity function together with the optimal hierarchical clustering, in a continuous feature-based hierarchical clustering formulation.

We conclude this thesis proposing a Graph Convolutional Layer in Max-Plus algebra that aims to be a first step towards Morphological Convolutions on Graphs. The proposed convolution is also benchmarked against state-of-the-art Graph Convolutional Layers. Results obtained are promising and prove that it is worth investigating further.

Keywords: Graph Theory, Hierarchical clustering, Segmentation, Image processing, Machine Learning, Point Clouds, Graph Neural Networks

Contents

List of Figures

List of Tables

Introduction

Motivation

Data are ubiquitous. Industries as social media, telecommunication, health-care to mention a few, every day produce a gargantuan amount of data. In 2018, the annual report of cloud software firm DOMO estimated that 2.5 quintillion (10^{18}) bytes of data are created each day and that the 90% had been created in less than two years. Thus, finding new ways on how to treat and analyze this great amount of information is a major challenge for many years now. When dealing with a collection of objects or things, the first step to achieve an understanding is to categorize them into classes/groups based on their similarities. For this reason, graphs are particularly interesting because they allow modeling relationships between different items. Indeed, graphs are the tool used throughout the thesis. Namely, this thesis proposes to investigate the following areas of applications.

The first application we put our interest in is point cloud processing. In particular we focus on the case of scans from road environments. Normally, this kind of scan is characterized by a high variation of point density. A common workflow proposed for Dynamic Object Detection and Tracking is to find and remove the ground from the scene as the first step. Once removed the ground from the scene, other objects can be identified as isolated components of the scene. However, state of the art algorithms for ground detection such as those based on λ-flat zones are conceived to be used on scans with a uniform point density. This lead us to look for simple interpolation methods able to cope with ground detection in the new setting. Another topic that we have explored is related to low-resolution scanners. Recent lidar development has permitted to reduce the costs of production of these sensors. Nonetheless, costs for a high-resolution scanner still remain too high to be employed on a large scale application. Moreover, current research benchmarks rely on datasets acquired using high-resolution scanners. Motivated by this challenge, we considered the problem of road detection as a case study. Our goal is to study the effect of reducing the resolution of the scanner on road detection task.

In Remote sensing applications, most of the classical Image Processing algorithms cannot be applied in context like streaming. In order to treat the great amount of data, methods cannot wait until the stream is complete, but instead it is necessary to decompose the images in strips or tiles. In this context, the decision taken on a given tile may be influenced by information from other tiles, some of them not received yet. The minimum spanning tree (MST) is involved in many remote sensing applications, as a fundamental step of morphological-based image segmentation methods. Therefore, the computation of a

MST in streaming is one of the problems we are going to tackle.

Along with these applications, during the thesis, we put our interest on Hierarchical Clustering (HC). HC is a powerful tool in data analysis mainly because it returns a nested partition of the observed data. This kind of representation has become increasingly popular over the last decades because it allows to model data at different levels of scale and semantics. HC methods have been applied to a wide range of data, such as images, videos, and text. In computer vision for example, these have been employed for object detection, image filtering, multi-scale image segmentation, and image characterization and understanding. The main strategies to implement a HC are to use either a divisive (top-down) approach or an agglomerative (bottom-up) approach. In recent years, a discrete optimization framework for similarity-based hierarchical clustering has been proposed by Dasgupta (2016). Later on, some continuous relaxations of this cost function have been proposed and stochastic gradient descend algorithm can be used to find solutions. The general definition of these problems assumes as input a graph and a similarity-matrix representing relationships between nodes, and the optimization is done on a fixed graph. We investigate a generalization of the problem to a family of data sampled from a fixed distribution. Our idea is to explore if it does exist a family of hierarchies associated to the distribution. To achieve this goal, we also look for an optimal metric function that may measure distances between the points. Hence, we combine metric learning with the continuous optimization framework to extend the formulation to cases in which the number of nodes in the graph is not fixed, and the similarities between points are not known a priori.

Finally, in the last part we focus on Graph Neural Networks (GNN). Recent years a great effort has been put on generalising deep learning architectures, and in particular Convolutional Neural Networks (CNN), beyond the Euclidean domain, and GNN has emerged as new frontier. The basic mechanism behind Graph Convolutional Layers is called Message Passing, that is composed by two steps: aggregate and update. When we apply a convolution on a given node, in the first step, a filter is applied to features of neighboring nodes. The result is then aggregated using a permutation invariant function (*e.g.* maximum, minimum, average, etc.). In the second step the features are updated using an activation function. We propose a novel permutation invariant function defined using morphological dilation/erosion operator.

Thesis overview

The list hereby contains a summary of the different chapters.

Chapter 1 starts with a review of graph theory, recalling the definitions of connectivity and paths on graphs. These concepts are fundamental to introduce the minimum spanning tree (MST) that we will use throughout all the thesis. The second part is a short overview on clustering in which we recall the most important methods that produce either flat or hierarchical clustering.

Chapter 2 contains a general introduction to point clouds analysis. The main challenge with this kind of data is its inherent lack of structure. We describe the principal strategies to acquire and handle point clouds, illustrating also some possible applications. The chapter ends with a description of some

benchmark datasets for the segmentation task.

Chapter 3 is divided into two parts and focuses on two problems related to point cloud processing. Ground detection and Road Detection. In the first, we review and propose two λ-flat zones based algorithms for ground detection and we compare them against the current state of the art. In the second part, we study the effect of subsampling points for the task of road detection. The goal is to understand the reliability of current deep learning methods on point clouds captured with low-resolution scanners. The partial results of this chapter have been presented in the conference paper Gigli et al. (2020a)

Chapter 4 tackles the problem of finding and updating an MST in streaming. We consider the case of an image that arrives via a stream decomposed in blocks of fixed size. The solution proposed, relies on a decomposition of the MST in two parts, *stable* and *unstable*. The stable part is made of edges that we can prove will belong to the final MST. The unstable part is made of edges that are not stable, and we need to keep them in memory to update the MST in the successive iterations. This decomposition reduces the memory footprint and permits to treat images of greater size compared to a naive algorithm that treats the entire image in one iteration. The main results of this chapter are part of the journal paper Gigli et al. (2020b)

Chapter 5 is divided into two parts in which we discuss two applications of deep metric learning. In the first part, we present the solution proposed to the SHREC'20 contest on retrieval of surface patches with similar geometric relief. In particular, our solution uses a Siamese Neural Network to approximate a similarity function between geometric reliefs. The second part concentrates on Hierarchical Clustering (HC). In recent years a theory for objective-based HC has been proposed. In the classical setting, a similarity function is given as an input element of the problems. We put ourselves in a semi-supervised setting and investigate the case in which we learn at the same time a good similarity function between the points and an optimal hierarchical clustering. The main findings of the first part of this chapter have been published in the journal paper Moscoso Thompson et al. (2020)

Chapter 6 is about Graph Neural Networks (GNN). It starts recalling the basic facts on GNN, illustrating the Message Passing mechanism that is the key tool to define Convolutional Neural Networks on Graphs. We use this same mechanism to define Convolutions in Max-Plus algebra that aim to be a first attempt towards Morphological Convolutions on Graphs.

List of Pubblications

This thesis has led the following publications

International conferences

- Gigli, Leonardo, Santiago Velasco-Forero, and Beatriz Marcotegui. "On minimum spanning tree streaming for image analysis." 2018 25th IEEE International Conference on Image Processing (ICIP). IEEE, 2018.

- Gigli, Leonardo, et al. "Road Segmentation on low resolution Lidar point clouds for autonomous vehicles." XXIV International Society for Photogrammetry and Remote Sensing Congress. 2020.

Journal Papers

- Gigli, Leonardo, Santiago Velasco-Forero, and Beatriz Marcotegui. "On minimum spanning tree streaming for hierarchical segmentation." Pattern Recognition Letters 138 (2020): 155-162.
- Thompson, Elia Moscoso, et al. "SHREC 2020: Retrieval of digital surfaces with similar geometric reliefs." Computers & Graphics 91 (2020): 199-218.

1 Graph Theory and Clustering

Resumé

Dans ce chapitre, nous introduisons les principaux concepts de la théorie des graphes. Après avoir introduit les définitions de base, nous allons passer à l'introduction d'un objet fondamental tout au long de la thèse, l'arbre couvrant de poids minimal. Cet objet est utilisé dans de nombreux domaines de l'analyse des données et également de l'analyse des images. La deuxième partie du chapitre est consacrée au problème du clustering des données. En particulier, un résumé de l'état de l'art des algorithmes pour le regroupement et le clustering hiérarchique est donné.

1.1 Graph theory

Graphs were firstly introduced by Leonhard Euler in the solution of the notable Königsberg's Seven Bridges problem (Biggs et al., 1986). The city of Könisberg (now Kaliningrad), shown in Figure 1.1, is crossed by the Pregel river, and at that time different part of the city are connected by seven bridges. The question was to prove if it could exist a path through the city crossing all the bridges once and only once. In Euler (1741) the non-existence of such a path is proven, laying the foundations of graph theory. Euler observed that the only important feature of a route was the sequence of the bridge crossed. For this reason, any other path entirely contained in each land mass is irrelevant and can be discarded. Thus, the proposed solution represents each land mass as a single abstract point (or *vertex*) and each bridge connecting two different sizes of the river with an arc (or *edge*). The resulting mathematical structure is a *Graph*.

Thanks to this simple and intuitive definition, graphs are particularly useful mathematical objects that are used to model different kinds of relations and processes in different areas of the sciences such as physics, biology, mathematics and computer science. In the following pages we present the basic objects of graph theory. For a broader introduction to graph theory, please refer to Jungnickel (2013).

Definition 1.1 (Graph)**.** A graph $\mathcal{G}$ is an ordered couple (V, E), where $V = \{v_1, \dots, v_n\}$ is the set of vertices of the graph and $E \subseteq V \times V$ is the set of edges of the graph. We say that $\mathcal{G}$ is undirected when

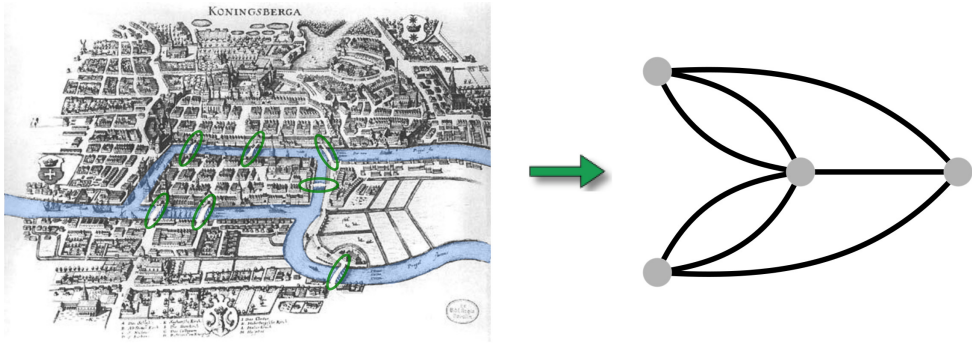


Figure 1.1 (Left) Map of Königsberg showing the seven bridges connecting different areas of the city. (Right) Graph modelling the seven bridges problem. A node is assigned to each district, while the edges correspond to the bridges connecting two areas.

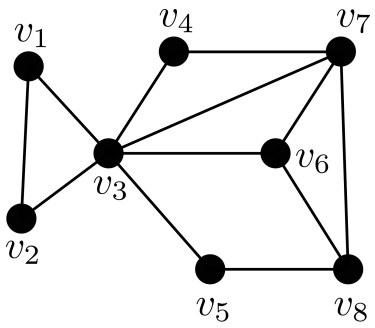


Figure 1.2 An example of undirected graph.

we consider the couples $(v_i, v_j) \in E$ as unordered pairs. Furthermore, given a couple $u, v \in V$ of nodes, we say that v is an adjacent vertex of u if $(u, v) \in E$.

Since in our cases graphs will be undirected, from now on, we consider every graph as undirected. It is possible to give a weight to each edge to model the importance of some connection in comparison to others.

Definition 1.2 (Weighted Graph)**.** A weighted graph $\mathcal{G}$ is a triple (V, E, w) where the couple (V, E) is a graph and $w : E \to \mathbb{R}$ is a weight function defined over the set of edges.

Note that, given a graph $\mathcal{G} = (V, E)$, such that $|E| = m$, the set $\{w : E \to \mathbb{R}\}$ of all possible weight functions over $\mathcal{G}$ is isomorphic to $\mathbb{R}^m$. A weight function can also be seen as a vector of $w = (w_1, \ldots, w_m) \in \mathbb{R}^m$, where w_i is the weight of edge e_i, for all $1 \leq i \leq m$.

Definition 1.3 (Sub-Graph)**.** Let $\mathcal{G} = (V, E)$ be a graph. A subgraph $\mathcal{G}' = (V', E')$ of $\mathcal{G}$, we write $\mathcal{G}' \subseteq \mathcal{G}$, is a graph such that $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. If $V' = V$, we say that $\mathcal{G}'$ *spans all the vertices of* $\mathcal{G}$.

Definition 1.4 (Graph Union)**.** Let $\mathcal{G}_1 = (V_1, E_1, w_1)$ and $\mathcal{G}_2 = (V_2, E_2, w_2)$ two weighted undirected graphs, such that

$$w_1\big|_{E_1 \cap E_2} \equiv w_2\big|_{E_1 \cap E_2},$$

where $w\big|_E$ is the restriction of the function w to the set E. We call $\mathcal{G}_1 \cup \mathcal{G}_2$, the weighted undirected graph $\mathcal{G} = (V, E, w)$ with $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, and for all $e \in E$:

$$w(e) = \begin{cases} w_1(e) & \text{if } e \in E_1, \\ w_2(e) & \text{if } e \in E_2. \end{cases}$$

Definition 1.5. Given a weighted graph $\mathcal{G} = (V, E, w)$ and a subset $E' \subseteq E$ of the edges, we call $\mathcal{G} - E'$ the graph $(V, E \setminus E', w)$ obtained by removing the edges E' from $\mathcal{G}$.

We can now move on showing two main ways to represent a graph. The first is a collection of adjacency lists, that is used to represent a finite graph. The second is the adjacency matrix.

Definition 1.6 (Adjacency-list representation)**.** The adjacency-list representation of a graph $\mathcal{G} = (V, E)$, is an array A of length $|V|$. Each element of the array A corresponds to a node $u \in V$ and $A[u]$ is a list of all the vertices adjacent to u.

Definition 1.7 (Adjacent-Matrix representation)**.** The adjacent matrix representation of a graph $\mathcal{G} = (V, E)$, is a matrix $A \in \mathbb{R}^{|V| \times |V|}$, such that

$$A[u][v] = a_{uv} = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

The adjacent-list representation is particularly useful for sparse graphs, that are graphs in which $|E| \ll |V|^2$. In this case, the adjacency list is more space-efficient than an adjacency matrix because its space usage is proportional to the number of edges in the graph, while an adjacency matrix store space is proportional to the square of the number of vertices. However, also compressed representations of sparse matrices exist, and they use adjacency lists in their implementation.

1.1.1 Topological definitions on graphs

Graphs are particularly attractive also because they intrinsically contain topological properties. Namely, it is intuitive to define graph objects as *paths* and *cycles* or determine when a graph is connected.

Definition 1.8 (Path on a graph)**.** We call *walk* a finite or infinite sequence of edges of the graph which joins a sequence of vertices. A walk is a sequence of edges $(e_i)_{i \in \mathcal{I}}$, for which there is a sequence of vertices $(v_i)_{i \in \mathcal{I}}$ such that $e_i = (v_i, v_{i+1})$ for each $i \in \mathcal{I}$. The sequence $(v_i)_{i \in \mathcal{I}}$ is the *vertex sequence* of the walk. If $\mathcal{I}$ is a finite set then the walk is said to be *finite walk*. A *trail* is a walk in which all edges are distinct. A *path* is a trail in which all vertices are distinct.

Definition 1.9 (Cycle)**.** Given a graph $\mathcal{G} = (V, E)$, a *simple cycle* is a finite non-empty trail $(e_1, \ldots, e_n)$ in which the only repeated vertices are the first and the last, that is $(v_1 = v, v_2, \ldots, v_n = v)$.

Definition 1.10 (Connected Graph)**.** We say that a graph $\mathcal{G}$ is connected if for every two vertices $u, v \in V$ it is possible to find a path Π from u to v.

Another equivalent definition of connected graph is the following.

Property 1.11. *A graph $\mathcal{G} = (V, E)$ is connected if and only if for every couple of subsets $V_1, V_2 \subseteq V$, such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$, there must exist at least one edge $e = (u, v) \in E$ such that $u \in V_1$ and $v \in V_2$.*

Definition 1.12 (Connected component)**.** Let $\mathcal{G} = (E, V)$ be a graph, a *connected component* of $\mathcal{G}$ is subgraph $\mathcal{G}'$ that is connected and is maximal for this property. In other words, for any other connected subgraph $\mathcal{F} \subseteq \mathcal{G}$, such that $\mathcal{G}' \subseteq \mathcal{F}$ then $\mathcal{G}' = \mathcal{F}$.

Please remark that any graph $\mathcal{G}$ can always be written as a disjoint union of its connected components.

Definition 1.13 (Graph Cut)**.** Given a graph $\mathcal{G}$, a *graph cut* is a partition $S = \{V_1, V_2\}$ of vertex set V into two disjoint non-empty subsets. We call *cocycle* the set $E(S)$ of edges in E having one endpoint in V_1 and the other in V_2.

Given a path π we are now interested in defining its length. A natural definition could be the number of edges composing the path. In case of a weighted graph, this could be the sum of the weights of the edges in the path. However, there is a more general definition that includes both.

Definition 1.14 (Generalized Path Length)**.** Let $\mathcal{G} = (V, E, w)$ a weighted graph. The *n-generalized length* of a path π is defined as

$$L_n(\pi) = \sqrt[n]{\sum_{e \in \pi} w_e^n}, \tag{1.1}$$

where $w_e = w(e)$ is the weight of edge e. In particular

$$\begin{cases} L_0(\pi) = \sum_{e \in \pi} 1 \\ L_1(\pi) = \sum_{e \in \pi} w_e \\ L_\infty(\pi) = \max_{e \in \pi} w_e = \lim_{n \to \infty} \sqrt[n]{\sum_{e \in \pi} w_e^n} \end{cases} \tag{1.2}$$

Remark 1.15. Remember that as we said before a weight function $w : E \to \mathbb{R}$ can also be seen as a vector of $\mathbb{R}^m$, whose component $w_i = w(e_i)$ is the value of the function at edge e_i. Similarly, a path $\pi = \{v_0, \dots, v_l\}$ can be associated to a projection function $\pi : \mathbb{R}^m \to \mathbb{R}^l$, $\pi(w) = (w_1, \dots, w_l)$ that projects the vector w on the subspace of the edges belonging to the path. With this in mind, let $\|\cdot\|$ be a norm on $\mathbb{R}^m$ and let w be a fixed weight function, then the length of the path π can be defined as the length of the vector $\|\pi(w)\|$ in $\mathbb{R}^l$. Thus, it is possible to exploit this relation to make a correspondence between L_p norms on $\mathbb{R}^m$ and the previously defined path length L_p.

Using the above definition we can define a distance on $\mathcal{G}$.

Definition 1.16 (Distance on graph)**.** Given $u, v \in V$, let Π_u^v be the set of all possible paths from u to v. The general n-distance d_n over $\mathcal{G}$ is defined as

$$d_n(u, v) = \min_{\pi \in \Pi_u^v} L_n(\pi), \tag{1.3}$$

for every $u, v \in V$. For $n = 1$, d_1 is also called *the shortest path distance*, while d_∞ is also known as *lowest path distance*.

1.1.2 Trees, Forests and Spanning Trees

Definition 1.17 (Tree and Forest)**.** A tree is a connected graph $\mathcal{G}$, that does not contain any cycle. A graph $\mathcal{F}$ that contains no cycles is called a *Forest*. Each connected sub-graph of the forest is a tree.

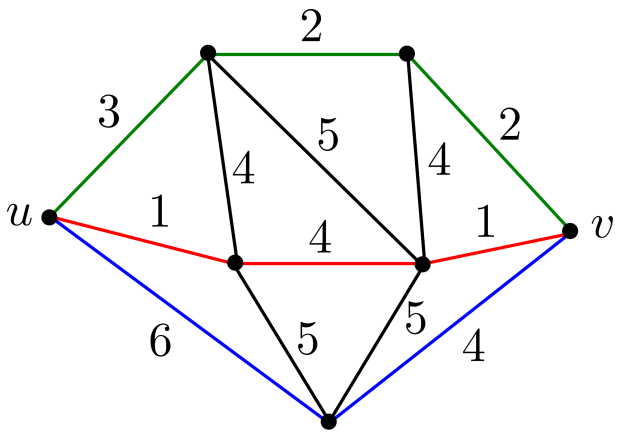


Figure 1.3 Graph distances. Optimal path connecting u and v changes according to the distance that we consider. In blue distance d_0, in red *shortest path* distance d_1 and in green *lowest path* distance d_∞.

We are reminded that Theorem 1.2.8 in (Jungnickel, 2013) proves that any tree $\mathcal{G}$ with n nodes contains exactly $n-1$ edges.

Definition 1.18 (Spanning Tree)**.** Let $\mathcal{G} = (V, E)$ be a connected graph. A subgraph $\mathcal{T} = (V', E') \subseteq \mathcal{G}$ is a spanning tree for $\mathcal{G}$ if $\mathcal{T}$ is a tree and $V' = V$. More generally, a *spanning forest* $\mathcal{F}$ for a graph $\mathcal{G}$ is a forest that spans all the nodes of the graph $\mathcal{G}$, and such that each tree $\mathcal{T} \subseteq \mathcal{F}$ in the forest is a spanning tree for a connected component in $\mathcal{G}$.

Proposition 1.19. *Let $\mathcal{T}_1 = (V_1, E_1)$ and $\mathcal{T}_2 = (V_2, E_2)$ two trees with $n_1 = |V_1|$ and $n_2 = |V_2|$ nodes respectively. Let $\mathcal{G} = \mathcal{T}_1 \cup \mathcal{T}_2$ the union of the two trees, such that $k = |V_1 \cap V_2|$. Then the number of simple cycles in $\mathcal{G}$ is $k - h - 1$, where h is the number of edges in common between the two trees.*

Proof. The number of edges in $\mathcal{T}_1$ is $n_1 - 1$, while the number of edges in $\mathcal{T}_2$ is $n_2 - 1$. Thus, the number of edges in $\mathcal{G}$ is:

$$n_1 - 1 + n_2 - 1 - h = n_1 + n_2 - h - 2.$$

We can observe that $\mathcal{G}$ contains a spanning tree with $n_1 + n_2 - k - 1$ edges. So, the number of cycles in $\mathcal{G}$ corresponds to the number of edges to remove in order to obtain a tree and is

$$(n_1 + n_2 - h - 2) - (n_1 + n_2 - k - 1) = k - h - 1.$$

□

Definition 1.20 (Minimum Spanning Tree)**.** Let $\mathcal{G} = (V, E, w)$ be a connected weighted graph. A Minimum Spanning Tree (MST) of a $\mathcal{G}$ is a subgraph $\mathcal{T} = (V', E', w)$ such that:

i) $\mathcal{T}$ is a tree

ii) $\mathcal{T}$ spans all the vertices of $\mathcal{G}$, i.e. $V = V'$

iii) the sum of its weights $\sum_{e \in E'} w(e)$ is minimum among all the possible spanning trees.

Let now review three well-known characterizations of the minimum spanning tree. The proofs for these theorems can be found in (Jungnickel, 2013, Chapter 4). Before, we remember that given a spanning

tree $\mathcal{T} = (V, E')$ of a connected graph $\mathcal{G} = (V, E)$, each time that we add any edge $e \in E \setminus E'$ to the spanning tree, this generates a cycle. We indicate this cycle as $C_{\mathcal{T}}(e)$.

Theorem 1.21. *Let $\mathcal{G} = (V, E, w)$ a weighted connected graph. A spanning tree $\mathcal{T} = (V, E', w)$ is a minimum spanning tree for $\mathcal{G}$ if and only if for every $e \in E \setminus E'$*

$$w(e) \geq w(e'), \ \ \forall e' \in C_{\mathcal{T}}(e).$$

In words, this first theorem says that if we add an edge to a MST, then this is the heaviest edge of the cycle that it generates. The vice versa is also true. The following characterization of MSTs is based on graph cuts.

Theorem 1.22. *Let $\mathcal{G} = (V, E, w)$ a weighted connected graph. A spanning tree $\mathcal{T}$ of $\mathcal{G}$ is a MST if and only if for each edge e in $\mathcal{T}$*

$$w(e) \leq w(e') \quad \textit{for every edge } e' \in E(S_{\mathcal{T}}(e)),$$

where $S_{\mathcal{T}}(e)$ is the graph cut obtained removing the edge e from $\mathcal{T}$.

Finally, another characterization that can be derived from Theorem 1.21 states that every path connecting two vertices in the MST is the shortest path in the sense of d_∞ distance previously defined. The proof can be found in (Chao, 1986).

Theorem 1.23. *Minmax path Let $\mathcal{G} = (V, E, w)$ a weighted connected graph. A spanning tree $\mathcal{T} = (V, E', w)$ is minimum if and only if for all couple of vertices $u, v \in V$, the path π^* connecting u, v in $\mathcal{T}$ is such that*

$$\pi^* = \arg\min_{\pi \in \Pi_u^v} L_\infty(\pi), \tag{1.4}$$

where Π_u^v is the set of all paths in $\mathcal{G}$ from u to v.

Remark that in general for a weighted graph $\mathcal{G}$ exists multiple minimum spanning trees, as shown in Figure 1.4. Nonetheless, it is easy to prove that if all the weights of a graph are distinct, then its minimum spanning tree is unique.

1.1.3 Algorithms to compute a Minimum Spanning Tree

We move on recalling the most common algorithms to find a MST of a graph, that are (Kruskal, 1956), Prim [(Dijkstra, 1959; Jarník, 1931; Prim, 1957)] and (Borůvka, 1926).

Borůvka's algorithm

The first algorithm that we present is (Borůvka, 1926) in 1926. This was the first presented algorithm developed to solve the problem of finding a MST of a connected graph. It iteratively constructs the MST using the fact that the smallest weight edge incident on any vertex v must be in the MST. This statement is justified by Theorem 1.22, observing that the minimum edge e incident to v respects $w(e) \leq w(e')$ for all edges in $E(S)$, where S is the partition $S = \{\{v\}, V \setminus \{v\}\}$. At each step the algorithm takes for each node v its smallest edges incident that does not generate cycles and add it to the MST. Successively, the

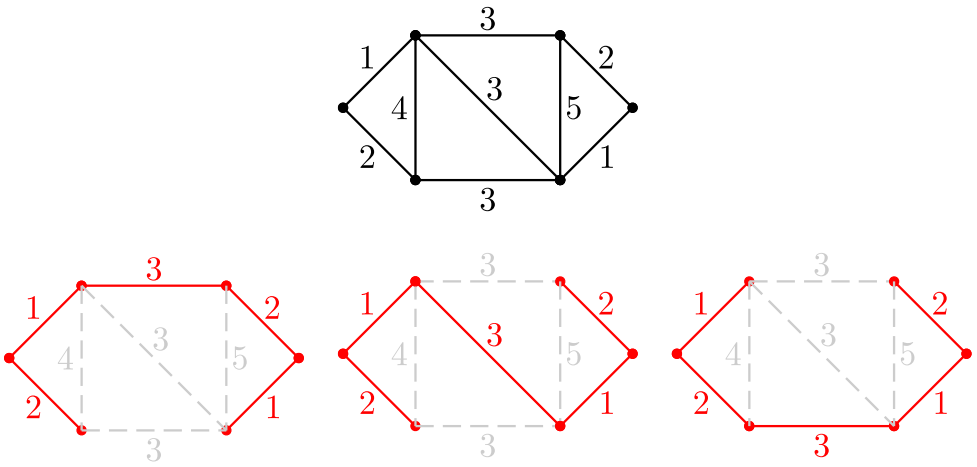


Figure 1.4 An example of graph for which we can find multiple different Minimum Spanning Trees. The graph in the row above is the input graph, while in the row below we illustrate three different MSTs. We color them with red edges belonging to each MST.

connected vertices in the MST are contracted together to form a *supervertex*. Two distinct supervertices are linked together in the graph using the smallest edge in the cocycle between the two supervertices. The process is then iterated on the newly defined graph, until all the nodes in the MST are connected, and only one supervertex remains. Concerning the time complexity of the algorithm, remark that at each step of the loop Borůvka reduces the number of nodes by a factor of at least two. Hence the while loop is executed at most $O(\log n)$ times. In each iteration, all the contraction can be done in $O(m)$ time. In total the method has a running time of $O(m \log n)$. A parallel version of this algorithm has been proposed by (Sun Chung and Condon, 1996).

Procedure 1 Borůvka's Algorithm

Input: A weighted, undirected graph $\mathcal{G} = (V, E, w)$
Output: A minimum spanning tree $\mathcal{T}$

```
1: procedure BORŮVKA
2:     T ← ∅
3:     while |T| < |V| − 1 do:
4:         F ← a forest consisting of the smallest edge incident to each vertex in G
5:         G ← G \ F
6:         T ← T ∪ F
```

Kruskal's Algorithm

Kruskal algorithm creates a forest where each vertex in the graph is initially a separate tree. It then sorts all the edges of the graph in an increasing order. Successively it starts iterating over the sorted edges. For each edge (u, v), it checks if vertices u and v belong to different trees. If so, it adds (u, v) to the forest, combining two trees into a single tree. It proceeds until all the edges have been processed.

Sorting the edges in non-decreasing order takes $O(m \log m)$, where m is the number of edges. The total running time of determining if the edge joins two distinct trees in the forest is $O(m\alpha(m, n))$ time, where α is the functional inverse of Ackermann's function, and n is the number of vertices. Therefore, the

Procedure 2 Kruskal's Algorithm

Input: A weighted, undirected graph $G = (V, E, w)$
Output: A minimum spanning tree $\mathcal{T}$

```
1:  procedure KRUSKAL
2:      Sort the edges in E in non-decreasing order by weight.
3:      T ← ∅
4:      Create one set for each vertex.
5:      for all edge (u, v) in the sorted order do:
6:          x ← FIND(u)
7:          y ← FIND(v)
8:          if x ≠ y then
9:              T ← T ∪ {(u, v)}
10:             UNION(x, y)
```

asymptotic running time of Kruskal's algorithm is $O(m \log m)$, which is the same as $O(m \log n)$ since $\log m = \Theta(\log n)$ by observing that $m \leq \frac{n(n-1)}{2}$, and $m = \Omega(n)$.

Prim's Algorithm

This algorithm has been introduced in (Jarník, 1931) and later rediscovered and republished in (Prim, 1957) and (Dijkstra, 1959). Differently from Kruskal, this algorithm works considering nodes of the graph instead of iterating over edges. In fact, it builds the MST starting from a random vertex v and assigning it to a subset of nodes $V' \subseteq V$. At each step it adds the smallest edge e in the cocycle $E(\{V', V \setminus V'\})$ and adds to V' the end point of the edge e not belonging to it. The method stops when $V' = V$. As in Borůvka, the optimality of the MST is guaranteed by the Theorem 1.22.

Procedure 3 Prim's Algorithm

Input: A weighted, undirected graph $\mathcal{G} = (V, E, w)$
Output: A minimum spanning tree $\mathcal{T}$

```
1:  procedure PRIM
2:      T ← ∅
3:      v an arbitrary vertex in V
4:      V' ← {v}
5:      while |V'| < |V| do:
6:          Find e = (v', u') the smallest edge such that v' ∈ V' and u' ∈ V \ V'
7:          T ← T ∪ (v', u')
8:          V' ← V' ∪ {u'}
```

The most time-consuming step in Prim's algorithm is the research for the smallest edge in the graph cut. A naive implementation at each step finds the smallest edge looking in the adjacency list of the graph. In this way each iteration costs $O(m)$ yielding to a total cost of $O(mn)$. By using Fibonacci heaps, Prim's algorithm can run in $O(m + n \log n)$ time.

1.2 Clustering on graphs

Clustering is an unsupervised learning task in which the goal is to group together similar points of data and separate different ones. Formally, given a set of elements $X = \{x_1, \dots, x_N\}$, a partition of the set X is a family $(X_i)_i$ of disjoint subsets of X that covers the entire set. In other words:

- $X_i \subseteq X$ for every i,
- $X_i \cap X_j = \emptyset$ each time that $i \neq j$,
- $\bigcup_i X_i = X$.

Thus finding a clustering of a set X is to find a partition $S = \{X_i\}_{1 \leq i \leq N}$ such that in each subset X_i elements are homogeneous and dissimilar from elements in other subsets. In (Murphy, 2012, Chapter 25) the author distinguishes clustering problems in two families based on the input data type. The first is called *similarity-based clustering* in which a *distance/similarity matrix* of size $N \times N$ is passed as input to the algorithm. While, the second is called *feature-based clustering* and the input to the algorithm is a $N \times F$ feature matrix, also called *design matrix*. Another distinction made is based on output of clustering algorithms. We distinguish between *flat-clustering* algorithms that return a separation in K groups of the input data and *hierarchical-clustering* algorithms that return a nested-partition of data. Finding a flat clustering is usually faster ($O(NF)$) compared to a hierarchical clustering ($O(N^2 \log N)$). However, finding a flat clustering generally needs to define the number K of clusters to identify. On the contrary, hierarchical clustering is a richer representation that can be viewed as a hierarchy of partitions. Two strategies are used in hierarchical clustering algorithms, that are *agglomerative clustering* and *divisive clustering*. Agglomerative clustering or bottom-up approaches begin assigning each element in a singleton and iteratively merge clusters together until a stop criterion is satisfied. Divisive clustering or *top-down* approaches start assigning all the elements to the same cluster and iteratively split the clusters until a stop criterion is met. Clustering algorithms can be either deterministic, either probabilistic. In the following, we review some classical clustering methods. For further about clustering algorithms, the reader may refer to (Murphy, 2012, Chapter 25) and to Xu and Wunsch (2005) and Xu and Tian (2015).

1.2.1 Flat Clustering

As said before, the aim of flat clustering is to split the input set X into K distinct groups such that similar objects are grouped together while dissimilar are separated. Among different approaches that have been proposed over the years, we divide between probabilistic approaches and deterministic ones. Hereby we summarize the main clustering algorithms, reviewing pros and cons.

Mixture Models

The first algorithm we present uses a probabilistic approach. The method assumes that the observed data are generated by K different probability distributions. For this reason, let us consider hidden variables to model the correlation between data. For example, for each element x_i, let z_i a hidden variable that takes values in the set $[K] = \{1, \dots, K\}$, and use a categorical distribution to model way z_i takes values in $[K]$. That is $p(z_i) = Cat(\pi)$, where $\pi = (\pi_1, \dots, \pi_K)$ and π_k is the probability of seeing element k. Hence the

distribution of observed data can be written as

$$p(x_i|\theta) = \sum_{k=1}^{K} p(x_i|z_i = k, \theta)p(z_i = k) = \sum_{k=1}^{K} \pi_k p_k(x_i|\theta), \tag{1.5}$$

where p_k is the k'th base distribution for the observation, and θ is the vector of parameters of the distributions. Please remark that the quantity $p(x_i|z_i = k, \theta)$ represents the probability that x_i belongs to cluster k. Probably, the most common among mixture models is represented by Gaussian Mixture Model, in which the p_k base distribution are multivariate Gaussians with mean μ_k and covariance matrix Σ_k. Thus the model has the form

$$p(x_i|\mu, \Sigma) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k),$$

The method aims to maximize the log likelihood

$$\ell(\theta) = \sum_{i=1}^{N} \log p(x_i|\theta) = \sum_{i=1}^{N} \log \Big[\sum_{z_i} p(x_i, z_i|\theta) \Big]. \tag{1.6}$$

The problem in the equation above is that the log cannot be pushed inside the sum. The solution has been proposed by the Expectation Maximization algorithm (EM) (Dempster et al., 1977; McLachlan and Krishnan, 2008; Meng and Van Dyk, 1997). The idea of the algorithm is that if the z_i are known then the parameters can be computed maximizing the complete data log-likelihood

$$\ell_c(\theta) = \sum_{i=1}^{N} \log p(x_i, z_i|\theta).$$

Viceversa, if parameters are known the values of z_i can be computed by maximizing the log-likelihood over all the possible values of z_i. Thus EM algorithm is an iterative algorithm, in which each iteration is composed in two steps called *E step* and *M step*. In the *E step* we compute the probabilities of hidden values z_i given the parameters $\theta^{(t)}$, while in the *M step* we use the just computed values of z_i to find a better estimate $\theta^{(t+1)}$ for the parameters of the models. More in details in the expectation step the expected complete data log likelihood is computed using parameters at time t:

$$\begin{aligned} Q(\theta, \theta^{(t)}) =& \mathbb{E}[\ell_c(\theta)|\mathcal{D}, \theta^{(t)}] = \mathbb{E}\Big[\sum_i \log p(x_i, z_i|\theta) \Big] = \sum_i \mathbb{E}\Big[\log \Big[\prod_{k=1}^{K} (\pi_k p_k(x_i|\theta))^{\mathbb{I}(z_i=k)} \Big] \Big] \\ =& \sum_i \sum_k r_{ik} \log \pi_k + \sum_i \sum_k r_{ik} \log p_k(x_i|\theta), \end{aligned} \tag{1.7}$$

where $r_{ik} = p(z_i = k|x_i, \theta^{(t)})$ is the influence of cluster k to point x_i. In the M step, the goal is to find a new estimated value $\theta^{(t+1)}$ optimizing the Q function with respect to θ

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta, \theta^{(t)}).$$

K-means

K-means algorithm is an iterative algorithm that aims to divide data in K clusters. An important assumption made in k-means clustering is that the shape of the clusters is convex. The goal of the method is to find a partition $S = \{X_1, \dots, X_K\}$ such that

$$\arg\min_{S} \sum_{k=0}^{K} \sum_{x \in X_k} \|x - \mu_k\|^2, \tag{1.8}$$

where μ_k is the mean of points in X_k. Basically, the goal is to find the partition that minimizes the intra-cluster variation. Despite its formulation seems simple, the problem is indeed NP-complex. However, the algorithm proposed by Lloyd (1982) finds an approximate solution even though it doesn't guarantee a convergence through the optimal solution. This algorithm, better known as *naive k-means*, starts placing randomly the k centroids $\mu_1^{(0)}, \dots, \mu_K^{(0)}$ on the space, and in each iteration t it executes the two following steps:

1. (Assignement) For each point x_i, it computes the closest mean point $\mu_k^{(t)}$ and it assigns x_i to cluster X_k.
2. (Update) It updates the position of the centroids using the observations assigned to each cluster

$$\mu_k^{(t+1)} = \frac{1}{|X_k|} \sum_{x \in X_k} x.$$

The stop criterion for the method is when assignments no longer changes. The running time of each iteration is $O(NKD)$, but it can be accelerated using the triangular inequality (He et al., 2010). Nonetheless, the initialisation of the centroids plays a key role in the quality of the solutions and several strategies have been developed such as k-means++ proposed by (Arthur and Vassilvitskii, 2007). Finally, we remark that the above algorithm can be seen as a special case of EM algorithm. In fact, if we consider an Isotropic Gaussian Mixture Model in which the distributions have all the same covariance matrix and assume that $\pi_k = 1/K$ is fixed, then the only parameters of the model to estimate are the Gaussian's centers μ_k.

Graph cut approaches

Graph Cuts is a wide family of approaches to clustering. These methods build a weighted undirected graph $\mathcal{G}$ whose vertices are observed data. Connection between vertices of the graph can be either already defined a priori or each point can be connected to its closest neighbors. Moreover edge weights are assigned either using the similarity matrix or using a distance between points. In both cases a $N \times N$ weight matrix $W(w_{ij})_{1 \le i,j \le N}$ is defined. The idea of Graph Cut methods is to find a cut that is minimal in some sense. For example, if the weights represent similarities between data, we can look for a partition $S = \{V_1, \dots, V_K\}$ that minimizes the cut weight

$$\text{cut}(S) = \text{cut}(V_1, \dots, V_K) = \frac{1}{2} \sum_{i=1}^{K} W(V_i, \overline{V}_i), \tag{1.9}$$

where $\overline{V}$ is the complementary of set V and $W(V_i, \overline{V}_i) = \sum_{u \in V_i} \sum_{v \in \overline{V}_i} w(u, v)$. Among all methods belonging to this family of methods, we review Spectral Clustering and Minimum Spanning Trees based graph cuts.

Spectral Clustering Spectral Clustering (von Luxburg, 2007) uses spectral graph analysis to solve the *normalized-cut problem* defined as:

$$\min_S \mathrm{Ncut}(S) = \min_S \frac{1}{2} \sum_{i=1}^{K} \frac{W(V_i, \overline{V}_i)}{\mathrm{vol}(V_i)} = \min_S \sum_{i=1}^{K} \frac{\mathrm{cut}(V_i, \overline{V}_i)}{\mathrm{vol}(V_i)}, \tag{1.10}$$

where $\mathrm{vol}(V) = \sum_{v \in V} d_i$ and $d_i = \sum_{j=1}^{N} w_{ij}$. Note that the quantity to minimize takes small values if the quantities $vol(V_i)$ are not small. Thus the objective function favours "balanced" partitions compared to splittings containing small components. (Jianbo Shi and Malik, 2000) proposed a solution to a relaxed version of the problem above that analyses the eigenvalues of the normalized graph Laplacian. Here, we briefly describe the solution for the case $K = 2$. Let define the degree of a vertex v_i as the quantity $d_i = \sum_{j=1}^{N} w_{ij}$ and let $D = \mathrm{diag}(d_1, \ldots, d_N)$ be the degree matrix. The unnormalized graph Laplacian is the matrix defined as $L = D - W$. Let now consider the cluster indicator vector f defined as

$$f_i = \begin{cases} \sqrt{\frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)}} & \text{if } v_i \in A \\ -\sqrt{\frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})}} & \text{if } v_i \in \overline{A}, \end{cases} \tag{1.11}$$

where the partition is $S = \{A, \overline{A}\}$. It easy to prove that $(Df)^T \mathbf{1}_N = 0$, $f^T D f = \mathrm{vol}(V)$ and $f^T D f = \mathrm{vol}(V)\,\mathrm{Ncut}(A, \overline{A})$, where V is the set of vertices of the graph. Thus the (1.10) is equivalent to

$$\begin{aligned} \underset{A}{\text{minimize}} \quad & f^T L f \\ \text{subject to} \quad & (Df) \perp \mathbf{1}_N \\ & f^T D f = \mathrm{vol}(V). \end{aligned} \tag{1.12}$$

The problem can be relaxed allowing f to take real values. Moreover replacing $g = D^{1/2} f$ the problem becomes

$$\begin{aligned} \underset{g \in \mathbb{R}^n}{\text{minimize}} \quad & g^T D^{-1/2} L D^{-1/2} g \\ \text{subject to} \quad & g \perp D^{1/2} \mathbf{1}_N \\ & \|g\|^2 = \mathrm{vol}(V). \end{aligned} \tag{1.13}$$

The solution of the problem is the second generalized eigenvector of $Lu = \lambda D u$. For the general problem with K clusters, the solution are the first K generalized eigenvectors $u_1, \ldots, u_k$ of the generalized eigenproblem $Lu = \lambda D u$. Finally to find assignments into clusters, k-mean algorithm is used on the rows of matrix $U = [u_1, \ldots, u_k] \in \mathbb{R}^{N \times k}$ whose columns are the first k eigenvectors. For a further reading on spectral graph theory the reader may refer to (Chung, 1997). The main advantage of Spectral clustering method compared to k-means is that it also works to separate non convex sets. Nonetheless, finding the first K eigenvalues of a the Graph Laplacian takes $O(N^3)$ and this can be a problem when dealing with big datasets. (Yan et al., 2009) proposed a framework for fast approximate spectral clustering that

speedups the computation showing a little degradation in clustering accuracy, while (Naumov and Moon, 2016) proposed a parallel version of spectral clustering that runs on GPU.

MST based Graph cuts Minimum Spanning Tree can be used to extract a flat clustering. One simple approach consists in sorting the edges of the MST in decreasing order and removing the $k-1$ heaviest edges. In fact, a new connected component is created each time an edge is removed from the MST. (Asano et al., 1988; Xu et al., 2001) uses this approach showing that the obtained partition is the one that minimizes the L_∞ diameter of the clusters. Recall that the diameter of a graph is the longest $\max_{u,v \in V} d(u,v)$ distance between any two vertices u, v of the graph. Moreover, (Felzenszwalb and Huttenlocher, 2004) proposes an efficient segmentation algorithm based on MST and a region comparison predicate that evaluates if there is evidence of a boundary. Finding the MST is a way faster compared to the spectral clustering methods and this represents the great advantage of this kind of approach compared with spectral clustering. One limitation of this second type of method is that often the obtained solution contains small components that are not relevant.

1.2.2 Hierarchical Clustering

Hierarchical clustering algorithms return a more sophisticated information compared to flat clustering. The goal is to compute a family of nested hierarchical partitions of the input data. Let start introducing a formal definition of a Hierarchical Clustering.

Definition 1.24 (Partial Ordering)**.** A partially ordererd set is a couple $(\mathcal{X}, \preceq)$, where a set $\mathcal{X}$ and $\preceq$ is a binary relation over $\mathcal{X}$, that satisfy

1. (Reflexivity) $a \preceq a$ for all $a \in \mathcal{X}$
2. (Antisimmetry) if $a \preceq b$ and $b \preceq a$ then $a = b$
3. (Transitivity) if $a \preceq b$ and $b \preceq c$ then $a \preceq c$

Note that not for all pairs of elements $(a, b) \in \mathcal{X} \times \mathcal{X}$, it holds $a \preceq b$ or $b \preceq a$. When one of the two holds we say that elements are *comparable*, otherwise we say that they are *incomparable*.

Definition 1.25 (Refinement)**.** Let X be any set and let $S = \{X_1, \ldots, X_k\}$ and $T = \{Y_1, \ldots Y_h\}$ two partitions of X. We say that S is a *refinement* of T if for any $X_i \in S$ it exists $Y_j \in T$ such that $X_i \subseteq Y_j$, and we write $S \preceq T$. Note that $\preceq$ is a partial ordering on the set of all possible partitions of X.

Definition 1.26 (Hierarchical Clustering)**.** A Hierarchical Clustering $\mathcal{H}$ of a set X is a set $\{S_0, \ldots, S_n\}$ of partitions of X such that

1. $S_0 = \{X\}$
2. $S_n = X$
3. $S_i \preceq S_{i-1}$ for all $i = 0, \ldots, n-1$.

A hierarchical clustering is often represented using a tree called *Dendrogram*. The root node of the dendrogram corresponds to the trivial clustering $\{X\}$, while the leaves of the tree are the clusters

$\{x_i\}$ composed of a single element. Each internal node corresponds to a cluster C that is the union of its children $\{C_1, \ldots, C_m\}$. Carlsson and Mémoli (2010) formally define a Dendrogram as pairs (X, λ), where X is the input set of observed data and $\lambda : [0, \infty) \to \mathcal{P}(X)$. The function λ is used to represent the clustering at different levels of the tree. The function has the following properties

1. $\lambda(0) = \{\{x_1\}, \ldots, \{x_n\}\}$.
2. $\exists t_0$, s.t. $\lambda(t) = \lambda(t_0) = \{X\}$ for all $t \geq t_0$.
3. if $r \leq s$, then $\lambda(r)$ refines $\lambda(s)$.
4. $\forall r \exists \epsilon > 0$ s.t. $\lambda(r) = \lambda(t)$ for all $t \in [r, r + \epsilon]$

The first condition means that the lowest value is mapped to the finest clustering possible composed by singletons. The second means that for t large enough the decomposition becomes trivial. The third assures that the partitions obtained are indeed nested, and the fourth condition is a technical condition due to the fact that the set of points in which the partition changes is finite. In Figure 1.5 we illustrate an example of hierarchical clustering of a set and its dendrogram representation. Remark that cutting the dendrogram at any height induces a flat clustering of the input data. As said before, there are two strategies

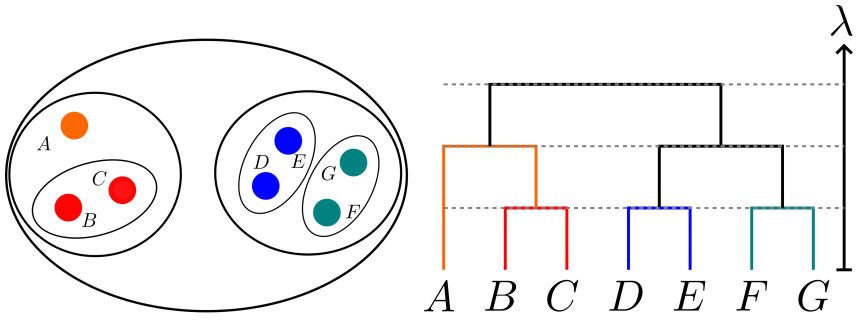


Figure 1.5 (Left) An example of hierarchical clustering and (Right) its dendrogram representation.

to implement hierarchical clustering algorithms, which are agglomerative and divisive. In agglomerative algorithms, each observation forms a singleton, and we merge clusters iteratively starting from the most similar. At each step, in fact, the two closest clusters are merged together. There exist many criteria to define similarities between clusters. Changing the criterion used to merge groups at any level leads to different hierarchies. In the following, we review the most common.

Single Linkage

In Single Linkage (SL) the distance between two clusters C_1, C_2 is defined as the minimum distance of any two points of each cluster. Formally

$$d_{SL}(C_1, C_2) = \min_{x \in C_1, y \in C_2} d(x, y). \tag{1.14}$$

Note that each time that we combine two clusters in Single Linkage, we merge them via the lightest weight connecting the two clusters. It can be proven that this is equivalent to compute the MST of the underlying complete graph and build the hierarchy iteratively merging the closest nodes on the MST.

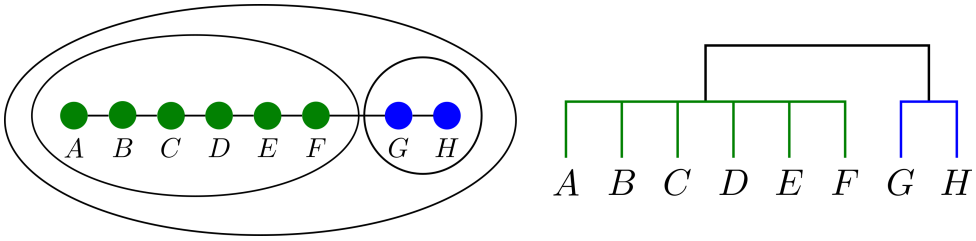


Figure 1.6 An example of chaining effect. Distances between points in the sequence from A to F are small. However, the distance between the first and the i-th element increases as we navigate through the sequence. Single Linkage puts the entire sequence in the same cluster. (Left) Input graph and (right) Single Linkage Hierarchy.

This criterion is *local* in the sense that it puts the attention only on the area where two clusters become close without taking into account global information about the two clusters. Thus, the two clusters may contain really different objects to each other and these would be neglected. This phenomenon is called *chaining effect*. For example, we could have a long sequence of data $(x_1, \ldots, x_n)$ as in Figure 1.6, that are two-by-two very similar, but the difference between the first and the i-th element increases as we navigate through the sequence. However, single linkage looks at the local distance between clusters and aggregates all the elements of the sequence in the same cluster generating a heterogeneous group. One way to circumvent this effect is to use a distance function that integrates some notion of density in the definition rather than use only local geometric information.

Complete Linkage

In Complete Linkage (CL) the distance between two clusters is defined as the distance between the two most distant pairs:

$$d_{CL}(C_1, C_2) = \max_{x \in C_1, y \in C_2} d(x, y) \tag{1.15}$$

Conversely from Single Linkage, Complete Linkage forms more compact clusters. In fact, if we consider the diameter d_C of a group C defined as the maximum distance between any couple of pairs, i.e. $d_C = max_{x,y \in C} d(x, y)$, then it is straightforward that Complete Linkage form clusters with small diameter.

Average Linkage

Average Linkage (AL) measures the average distance between all couples of each cluster, that is

$$d_{CL}(C_1, C_2) = \frac{1}{n_1 n_2} \sum_{x \in C_1} \sum_{y \in C_2} d(x, y), \tag{1.16}$$

where n_1 and n_2 are respectively the number of points in C_1 and C_2. Average Linkage is a sort of compromise between Single Linkage and Complete Linkage. Generally, clusters obtained are relatively compact and far apart.

Ward's minimum variance Method

In Ward's minimum variance Method at each step the goal is to merge the two clusters that leads to the minimum increase in the total within-cluster variance after merging. For the euclidean distance the

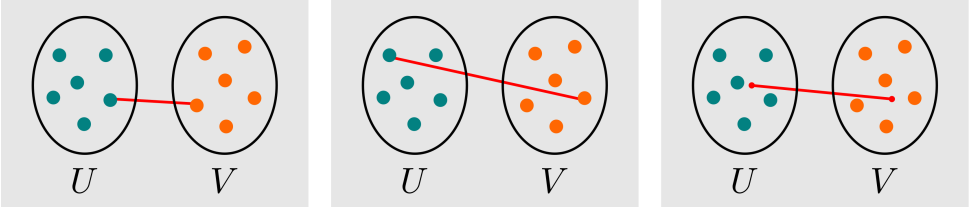


Figure 1.7 (Left) Single Linkage (Center) Complete Linkage (Right) Average Linkage

within-cluster variance W_C of a cluster C is defined as:

$$W_C = \sum_{x \in C} \|x - \mu\|^2,$$

where $\mu = \frac{1}{n_C} \sum_{x \in C} x$ is the cluster centroid. After each merging step the distances between clusters must be updated. This can be achieved using the Lance-Williams algorithm that uses a recursive formula to update distances between clusters at each step. In particular at the first step all the distances are initialized using the euclidean distance between points. Then, given a couple C_i, C_j of clusters merged the formula used to update distances with any other cluster C_k is

$$d(C_i \cup C_j, C_k) = \frac{n_i + n_k}{n_i + n_j + n_k} d(C_i, C_k) + \frac{n_j + n_k}{n_i + n_j + n_k} d(C_j, C_k) - \frac{n_k}{n_i + n_j + n_k} d(C_i, C_j). \quad (1.17)$$

Axiomatic Approaches to Clustering

Kleinberg (2002) proposed to study the clustering problem using an axiomatic approach. In particular, given a set X of $n \geq 2$ points, the author define the process of clustering elements of X as a function f that takes a distance function $d : X \times X \to \mathbb{R}_+$ defined on X and returns a partition S of X. Moreover, the author identified the following three desirable properties for a clustering function:

- **Scale invariance:** the clustering function should not be sensitive to changes in the unit of distance measurements. Formally, for any distance function d and any $\alpha > 0$, we have

  $$f(d) = f(\alpha \cdot d),$$

  where $\alpha \cdot d(x_i, x_j) = \alpha d(x_i, x_j)$, for any $x_i, x_j \in X$.
- **Richness:** the output of a clustering function should be rich. Let $\Gamma(X)$ be the set of all possible partitions of the set X, and let $\mathrm{Range}(f)$ be the image set of the clustering function f. The $\mathrm{Range}(f)$ should coincide with the set of all partitions $\Gamma(X)$ of X. In other words, f should be a surjective function.
- **Consistency:** a clustering function should be consistent with transformations the metric d. For example, if we shrink distances between points in a cluster and expand distances between points in different clusters, we expect that f returns the same partition. Formally, let S be a partition of X, we say that a metric $d' : X \times X \to \mathbb{R}_+$ is an *S-transformation* of d if:
  1. $d'(x_i, x_j) \leq d(x_i, x_j)$, for any $x_i, x_j \in X$ belonging to the same cluster of S,
  2. $d'(x_i, x_j) \geq d(x_i, x_j)$, for any $x_i, x_j \in X$ belonging to different clusters of S.

The consistency property says that given d and d' two distance functions on X, if $f(d) = S$ and d' is an *S-transformation* of d, then $f(d') = S$.

Then the author proved that no clustering scheme satisfying these conditions can exist.

Theorem 1.27. *For each $n \geq 2$, there is no clustering function f that satisfies Scale-Invariance, Richness and Consistency.*

Later on, Zadeh and Ben-David (2012) relaxed the *richness* condition, to a K-richness condition. Basically, this condition states that the clustering function f should cover the set of all possible partitions of X in K clusters. The argument carried by the authors for this condition is that in many algorithms the number of clusters K is required as input. In addition, the authors proposed to include a further condition

- **Order Consistency:** for any couple of distances d and d', if the order of couples of points in d is the same as the order of couples in d', then $f(d) = f(d')$.

The authors proved that Single Linkage is the only clustering method that has all the listed properties.

Theorem 1.28. *Single Linkage is Consistent, K-Rich, Scale-Invariant and Order-Consistent.*

Single Linkage, ultrametrics and stability

We conclude this section on Hierarchical clustering showing two interesting properties about single-linkage. The first is about a link between single linkage clustering and ultrametric distance. This link has been established in the works of Johnson (1967) and Jardine et al. (1967). However, in the following we will refer to the result shown by Carlsson and Mémoli (2010). Let start introducing the definitions of metric space and ultrametric distance.

Definition 1.29 (Ultrametric)**.** Let X be a set, a metric over X is a function $d : X \times X \to \mathbb{R}_+$ that satisfies

1. $d(x, y) = 0$ if and only if $x = y$, (identity of indiscernible)
2. $d(x, y) = d(y, x)$, (symmetry)
3. $d(x, y) \leq d(x, z) + d(y, z)$, (triangular inequality)

for all $x, y, z \in X$. An ultrametric is a metric function $u : X \times X \to \mathbb{R}_+$ that satisfies also the following ultrametric inequality:

$$u(x, y) \leq \max\{u(x, z), u(y, z)\}, \text{ for all } x, y, z \in X.$$

An ultrametric space is a particular metric space in which all triangles are isosceles.

Remark 1.30. Given a metric space (X, d) there is a canonical way to construct an ultrametric u from d:

$$u(x, y) := \min \left\{ \max_{i=0,\dots,k-1} d(x_i, x_{i+1}), \text{ s.t. } x = x_0, \dots, x_k = y \right\}.$$

Such ultrametric u is sometimes known as *sub-dominant ultrametric*, and it has the property that if $u' \leq d$ is any other ultrametric on X, then $u' \leq u$.

The first result we report is (Carlsson and Mémoli, 2010, Theorem 9) and it states that dendrograms and ultrametrics are equivalent.

Theorem 1.31. *Given a finite set X, there is a bijection $\Psi : \mathcal{D}(X) \to \mathcal{U}(X)$, between the collection $\mathcal{D}(X)$ of all dendrograms over X and the collection $\mathcal{U}(X)$ of all ultrametrics over X such that for any dendrogram $\lambda \in \mathcal{D}(X)$, the ultrametric $\Psi(\lambda)$ over X generates the same hierarchical decomposition as λ, that is:*

$$\text{for each } r \geq 0, x, y \in B \in \lambda(r) \iff \Psi(\lambda)(x, y) \leq r.$$

Furthermore, this bijection is given by

$$\Psi(\lambda)(x, y) = \min\{r \geq 0 \mid x, y \text{ belong to the same element of the partition in } \lambda(r)\}.$$

The Theorem above allows us to represent dendrograms as ultrametric spaces and vice versa. In this way, any hierarchical clustering method can be represented as a map from finite metric spaces into finite ultrametric spaces. In this regard, the authors prove that the ultrametric produced by Single Linkage Hierarchical Clustering on a metric space (X, d) coincides with the subdominant ultrametric of distance d.

The second result proved by Carlsson and Mémoli (2010) that we mention is the stability to perturbations of single-linkage. To evaluate the effect of perturbations on dataset and compare two hierarchies, they make use of the Gromov-Hausdorff distance, that measures how far two compact metric spaces are from being isometric. They prove that single-linkage, differently from average-linkage and complete-linkage, is the only hierarchical method that is stable and continuous in the sense of the Gromov-Hausdorff distance. This means that for any small perturbation of the input data, the hierarchies obtained with the single linkage clustering for the original and the perturbed data are at small distance from one another.

1.2.3 Evaluate a clustering

An important aspect is how to evaluate the quality of a partition returned by any algorithm. Since clustering is an unsupervised task, often there does not exist some ground truth label for the observed data. Thus, the goal of the metrics proposed is to compare two different partitions. However, in our experiments we will always evaluate a predicted clustering against a ground truth label. In this section we recall the most common scores used to compare clustering methods.

Rand Index

Introduced by Rand (1971), the Rand Index computes a similarity score between the two partitions by considering all pairs of samples and counting pairs that are assigned to the same or different clusters in both clusterings. In other words, given two partitions $S = \{X_1, \ldots, X_k\}$ and $T = \{Y_1, \ldots, Y_h\}$ of the same dataset of N points, the following quantities are computed

- True Positive (TP): the number of pairs of elements that are in the same subset in S and in the same subset in T
- True Negative (TN): the number of pairs that are in different subsets in S and in different subsets in T

- False Positive (FP): the number of pairs that are in different subsets in S but in the same subset in T
- False Negative (FN): the number of pairs that are in the same subset in S but in different subsets in T

Then Rand Index $RI(S,T)$ between the two clustering is defined as:

$$RI(S,T) = \frac{TP+TN}{TP+FP+FN+TN}, \tag{1.18}$$

that is the fraction of coinciding decisions between the two clusterings, and it holds $0 \leq RI(S,T) \leq 1$. Note that the normalization quantity in the previous equation is equal to $\binom{n}{2}$. However, other normalization factors has been discussed, as the one defined by Hubert and Arabie (1985) that defined the *Adjusted Rand Index* ARI. This last is the corrected-for-chance version of the Rand index. Such a correction for chance establishes a baseline by using the expected similarity of all pair-wise comparisons between clusterings specified by a random model:

$$ARI(S,T) = \frac{\text{index} - \text{expected index}}{\text{max index} - \text{expected index}}. \tag{1.19}$$

In practice given the two partitions S and T, the Adjusted Rand Index is computed using the contingency table:

$S \diagdown T$	Y_1	Y_2	$\cdots$	Y_h	sums
X_1	n_{11}	n_{12}	$\cdots$	n_{1h}	a_1
X_2	n_{21}	n_{22}	$\cdots$	n_{2h}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_k	n_{k1}	n_{k2}	$\cdots$	n_{kh}	a_k
sums	b_1	b_2	$\cdots$	b_h	

,

where $n_{ij} = |X_i \cap Y_j|$ is the number of objects in common between sets X_i and Y_j. The Adjusted Rand Index is

$$ARI(S,T) = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right]/\binom{n}{2}} \tag{1.20}$$

Purity

Purity metric is a so-called *external criterion of cluster quality*, that is it evaluates how well a given partition S matches some gold standard classes that are given along with data. Thus let $S = \{X_1, \ldots, X_k\}$ the partition computed by the algorithm and let $\{C_1, \ldots, C_m\}$ possible classes. Let n_{ij} the number of objects in cluster i that belong to class j and let n_i the number of objects in cluster X_i. The purity of a cluster X_i is defined as $p_i = \max_j n_{ij}/n_i$, that is the empirical probability of the most frequent class in cluster i. The overall purity of a clustering is defined as

$$P(S,C) = \frac{1}{N}\sum_i n_i p_i = \frac{1}{N}\sum_i \max_j n_{ij}. \tag{1.21}$$

When comparing two partitions with the same number of clusters, in which one of the two is the ground truth and the other is the predicted partition, purity measures the accuracy of the prediction up to a permutation of the labels.

Mutual Information

The last way to measure cluster quality is Mutual information. Given random variables X, Y, mutual information is defined as

$$MI(X,Y) = \int\int p_{(X,Y)}(x,y) \log\left(\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)}\right) dx\, dy,$$

where $p_{(X,Y)}$ is the joint probability density function of X and Y and p_X and p_Y are marginal probability density functions of variables X and Y respectively. Now let $S = \{X_1, \dots, X_k\}$ and $T = \{Y_1, \dots, Y_h\}$ two partitions we can define $p_{S,T}(i,j) = \frac{|X_i \cap Y_j|}{N}$, the probability that a random observed data is in cluster i in the first partition and in cluster j in the second. Moreover let $p_S(i) = \frac{|X_i|}{N}$, the probability that an object belongs to cluster X_i, and $p_T(j) = \frac{|Y_i|}{N}$ the probability that an object belongs to class j. Thus mutual information between the two partitions is the quantity defined as

$$MI(S,T) = \sum_i \sum_j p_{(S,T)}(i,j) \log\left(\frac{p_{(S,T)}(i,j)}{p_S(i)p_T(j)}\right) \tag{1.22}$$

This quantity is bounded by $0 \leq MI(S,T) \leq \min\{H(S), H(T)\}$, where $H(S)$ and $H(T)$ are respectively the entropy of S and T defined as

$$H(S) = \sum_i p_S(i) \log p_S(i)$$

and

$$H(T) = \sum_j p_T(j) \log p_T(j).$$

Since mutual information achieves high scores when one of the two partitions is made of lots of small clusters the compensation for this is the *normalized mutual information*

$$NMI(S,T) = \frac{MI(S,T)}{1/2(H(S) + H(T))}. \tag{1.23}$$

Another variation of mutual information is the Adjusted Mutual Information (AMI) (Vinh et al., 2010), that similarly to Adjusted Rand Index, correct the scores using the expected mutual information between two random assignments

$$AMI(S,T) = \frac{MI(S,T) - E\{MI(S,T)\}}{\max(H(S), H(T)) - E\{MI(S,T)\}}. \tag{1.24}$$

The AMI holds one when the two partitions are identical and zero when the MI between the two partitions is equal the expected value due to chance.

1.3 MST applications to Image Segmentation

We conclude this chapter discussing Graph Clustering applications to Image Analysis, and specifically to Image Segmentation. A natural way to build a graph from an image $\mathcal{I}$ is to consider each pixel p of the image as a vertex of the graph and use edges to link adjacent pixels. The graph is known also as *pixel adjacency graph*. Pixels' adjacency mostly used in literature are three (see Figure 1.8):

- 4-connectivity: in which each pixel is connected with the closest top-bottom-left-right pixels. This creates a regular grid over the image.
- 8-connectivity: each pixel is connected with all the 8 surrounding pixels.
- 6-connectivity: each pixel is connected with the pixels in order to form a hexagonal grid. In practice, this connectivity is harder to implement with respect two the two above.

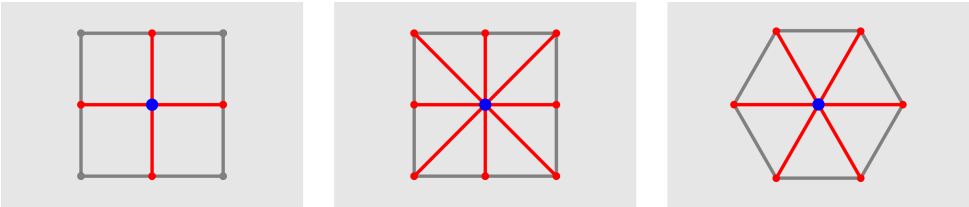


Figure 1.8 (Left) 4-connectivity (Center) 8-connectivity (Right) 6-connectivity

Each connectivity implicitly defines a different topology on the graph. Moreover, the number of edges in the graph is affected by this choice. Since the complexity of many algorithms depends on the number of edges in the graph this imply that also run time is affected by choice of connectivity. In our case we will use the 4-connectivity. Commonly, the weight function $w : E \to \mathbb{R}$ on the edges measures the dissimilarity between two connected pixels. For example, in the case of gray-scale images a simple weight function is the one that measures intensity differences between neighboring pixels, i.e. $w(e) = w(v_i, v_j) = |\mathcal{I}(p_i) - \mathcal{I}(p_j)|$ for each $e = (v_i, v_j) \in E$, where $\mathcal{I}(p_i)$ is the intensity of pixel p_i. The goal in image segmentation is to split the input image in several regions each one possibly containing a different object. An example of image segmentation task is to separate the object portrayed in the foreground of the image from the background. Among the great variety of existing approaches, hereby we review three methods that use graphs and in particular the MST to achieve the task.

1.3.1 λ-Flat Zones

Segmenting an image, we generally aim to group together homogeneous zones of the image. λ-flat zones (also referred to α connected components) uses this principle to achieve the segmentation. Without loss of generality, we can assume that the weights are ranged in the set $\{0, \dots, |E| - 1\} \subset \mathbb{N}$. In fact, it is always possible to find a one-to-one correspondence from the set $\{W(e) | e \in E\}$ to the set $\{0, \dots, |E| - 1\}$. Given an integer $\lambda \in \{0, \dots, |E| - 1\}$ we can extract from the graph $\mathcal{G}$ a subgraph $\mathcal{G}_\lambda = (V, E_\lambda, W)$ such that $E_\lambda = \{e \in E | W(e) < \lambda\}$, that is the graph $\mathcal{G}_\lambda$ is obtained from $\mathcal{G}$ removing all the edges whose weight is equal or greater than λ. The set $P_\lambda = \{\mathcal{C}_0, \dots, \mathcal{C}_{n_\lambda}\}$ made of all connected components of $\mathcal{G}_\lambda$ is a partition of the set of nodes V. The connected components are also called *lambda-quasi-flat zones*, since the variation between two neighbouring nodes in a connected component does not exceed

λ. Remark that if $\lambda_1 \leq \lambda_2$, then the partition obtained using λ_1 is finer than the one obtained using λ_2. That is, for each connected component $\mathcal{C} \in P_{\lambda_1}$ it exists a connected component $\mathcal{C}' \in P_{\lambda_2}$ such that $\mathcal{C} \subseteq \mathcal{C}'$. In that case, we write $P_{\lambda_1} \preceq P_{\lambda_2}$ to indicate that partition P_{λ_1} is finer than partition P_{λ_2}. By making varying the values of lambdas from zero to $|E|-1$ we obtain a sequence of partitions such that $P_0 = V \preceq \ldots \preceq P_{|edges|-1} = \{V\}$, that is the *$\lambda$-quasi-flat zone hierarchy*. It turns out that we obtain the same result thresholding the edges of the minimum spanning tree of $\mathcal{G}$. λ-flat zone hierarchy returns is equal to single linkage hierarchy when applied on the image graph. Indeed, Cousty et al. (2018) shows the equivalence between MST, quasi-flat zones hierarchy and saliency map of an image.

1.3.2 Watershed Cuts

Probably the most popular algorithm know in this domain is Watershed (Beucher and Lantuéjoul, 1979). The idea is to model the image as topographic relief and to flood the surface with water from the minima of the image. Each time the water coming from two different sources meet a barrier is created preventing the two regions to merge. Applying this transformation to image gradient the contours of the image will correspond to watershed lines while homogeneous zones of the image will correspond to catchment basins. This approach leads to an over segmentation of the input image because an image may contain several regional minima. To better control the process, one solution is to flood from a fixed number of markers chosen among the image pixels (Meyer and Beucher, 1990). Moreover, (Meyer, 1994) establish a link between Minimum Spanning Forest and Watershed algorithms with markers. This lead to Watershed Cuts (Cousty et al., 2009) that we will describe better later on. Angulo and Jeulin (2007) propose to randomly select the markers and generate a random segmentation rather than using deterministic markers. The idea is that repeating this process multiple times, we can evaluate the strength of the contours of the image. Stronger contours appear more frequently because there are many possible configurations of markers which select them. Finally, it is worth to cite (Couprie et al., 2011) that proposed Power Watershed a framework that unify watershed with markers, graph cuts, random walker and shortest path algorithm using the same formulation.

The watershed cut (Cousty et al., 2009) is the version on graph of the Watershed algorithm with markers. It needs a set $M = \{p_0, \ldots, p_k\}$ of pixels as markers. At the end of the process each marker will be contained in a different region of the segmented image. Basically, the method takes as input a weighted graph $\mathcal{G} = (V, E, W)$ and a set M of markers and proceeds in the following way:

1. add to the set of nodes V a special node z called *well*, that is $V' = V \cup \{z\}$
2. for each marker p, add an edge (p, z) to the set of edges E, whose weight is $m-1$ (with $m = \min_{e \in E} W(e)$), i.e. $E' = E \cup \{(p, z) | p \in M\}$
3. compute a minimum spanning tree $\mathcal{T}'$ of the graph $\mathcal{G}' = (V', E', W')$
4. return the connected components $\mathcal{C}_0, \ldots, \mathcal{C}_k$, of the subgraph $\mathcal{F} \subseteq \mathcal{T}'$ restricted only to nodes in V.

Remark that the connected components are the regions of our segmentation and that a marker is contained in each of them. In fact, each marker p is connected in $\mathcal{G}'$ to the well z with an edge whose weight is minimum. Necessarily those edges will be in the minimum spanning tree $\mathcal{T}'$ of the extended graph $\mathcal{G}'$, and for this reason, each path in the MST that connects two markers must pass by the well node z.

We conclude that in the subgraph $\mathcal{F} \subseteq \mathcal{T}'$ restricted to nodes V, two markers must belong to different connected components.

1.3.3 $(\alpha - \omega)$ constrained connectivity

Introduced by Soille (2008) this method extends the concept of λ-quasi-flat zones and tackles the problem of chaining-effect Soille (2011). In fact, it can happen that distinct objects in the image are separated by one or more transitions going in steps having an intensity height less than or equal to λ. It follows that those objects fall in the same λ-quasi-flat zone even though they are distinct. Essentially, the idea proposed is to introduce a connectivity index to measure the degree of connection of a connected component. Briefly, let I be a greyscale image, and consider a λ-quasi-flat zone $\mathcal{C}$. We define the *range* of the quasi-flat zone $R(\mathcal{C})$ as the most significant difference of intensity among two pixels in $\mathcal{C}$, *i.e.*, $R(\mathcal{C}) = \max_{p,q \in \mathcal{C}} |I(p) - I(q)|$. In the original paper (Soille, 2008) proposed to use the range of a connected component as a measure of connection, but it could be any predicate with a non-decreasing property on λ-*quasi-flat zones* such as area or volume of λ-*quasi-flat zones*. However, here-under we recall its original definition. Given a pixel p, the (α,ω)-connected component of p is the largest λ-quasi-flat zone containing p such that $\lambda \leq \alpha$ and with a range less than ω,

$$(\alpha,\omega) - CC(p) = \max_{\lambda} \Big\{ \lambda - CC(p) \mid \lambda \leq \alpha \text{ and } R(\lambda - CC(p)) \leq \omega \Big\}$$

where $\lambda - CC(p)$ is the λ-quasi flat zone that contains p. Moreover, two pixels p and q are (α,ω)-*connected* if and only if $q \in (\alpha,\omega) - CC(p)$. It turns out that the relation *"is (α,ω)-connected"* is an equivalence relation and thus it generates a unique partition of the image definition domain.

2 Point Clouds

Resumé

Le chapitre commence par une présentation de la définition des nuages de points 3D ainsi que des technologies existantes pour l'acquisition de données 3D. Par la suite, nous présentons les techniques courantes de traitement et d'analyse des nuages de points ainsi que les bases de données les plus utilisées dans l'état de l'art.

2.1 Introduction

Point clouds are a well-known data format in the domain of 3D vision since the 70s, but it is from the advancements in hard drive storage and computational capabilities during the end of the 90s that point clouds have seen a large diffusion on application domains as civil engineering, cultural heritage maintenance or robotics. In recent years further improvements in laser technology have made it possible to build precise and increasingly less expensive high-resolution sensors. Thanks to these sensors it is possible to obtain precise 3D models of the scanned objects/environments. This chapter is an introductory chapter to point clouds. We start with an abstract definition of point clouds presenting also the main properties. Successively, we move on talking about sensors technology and commonly used data-structures that allow us to handle and visualize point clouds. Finally, we conclude by introducing some popular datasets.

2.2 Point Clouds

A point cloud is a finite set of points $\mathcal{P} = \{\boldsymbol{p}_n \in \mathbb{R}^3 \mid 1 \leq n \leq N\}$, where N is the cardinality of the set $\mathcal{P}$. An intuitive way to imagine a point cloud, is as a set of points that sample the surface of an object. For example, Figure 2.1 shows a point cloud representing the Colosseum.

A vector of features $\boldsymbol{v}_n \in \mathbb{R}^d$ can be associated to each point $\boldsymbol{p}_n \in \mathcal{P}$ in the point cloud. In some cases these features are the colour of the object's surface or the normal vector to the surface. In the basic setting, when no further information is available, point coordinates are used as features vectors. Despite their

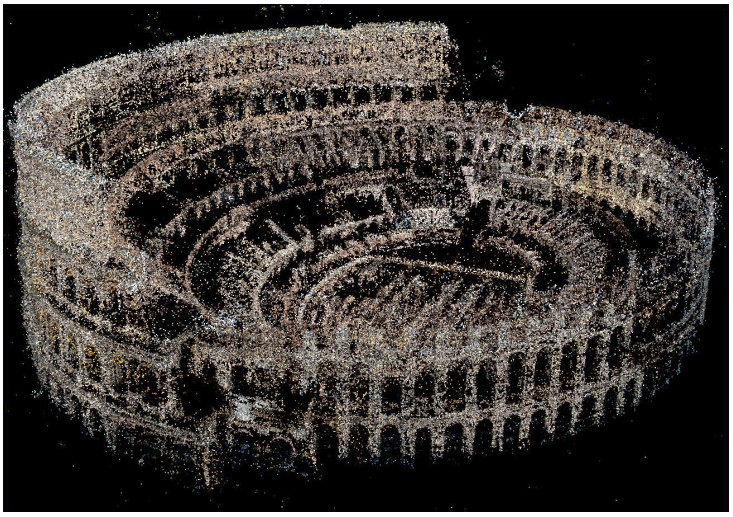


Figure 2.1 Point cloud representation of the Colosseum.

simplicity in definition, point clouds are a particularly challenging data during processing. First, point clouds are not always regularly sampled over the entire object or scene. In real cases one of the issues to face is the variation of density of points across different regions of the scene. Second, point coordinates are not defined on a regular grid, as pixels of an image, but instead are spread over a continuous interval. Thus spacing between points is not always fixed and, contrary to image pixels, distance between neighbouring points can vary. Finally, a point cloud is nothing but a set of points, and it is often stored as a list of coordinates in a file. As a set, the order in which the points are stored doesn't change the represented scene. This means that a point cloud with N points has $N!$ equivalent representations. Designing an algorithm to process a point cloud requires considering all these aspects, and as we will see in this chapter several approaches have been developed during the years to face these properties.

2.3 Point Cloud Scanning

In this section we present three techniques to capture a 3D point cloud. For each technique, we discuss the principles and the main fields of applications.

2.3.1 Photogrammetry

3D point clouds of a given object can be generated from a collection of overlapping images or video sequences. The methods are based on detecting and matching features and recovering extrinsic camera calibration information. The main advantage of this kind of technique is that camera are really cheap, easy to use and provide colour information. Unfortunately, the recovered 3D information is not as precise as the one that we obtain with other active sensors as for example LiDAR.

2.3.2 RGB-D Cameras

RGB-D images are a particular format of images with four channels. The first three are the regular RGB colour channels and the fourth contains depth information. Starting from an RGB-D image it is possible to obtain a 3D point cloud. Using the depth information along with the intrinsic parameters of the camera it is

possible to associate to each pixel a point in the 3D environment. The resolution of the point cloud is thus correlated with the resolution of the camera. Thanks to the cheaper cost compared with more sophisticated sensors, modern RGB-D cameras (such as Kinect, Real Sense and Apple depth cameras) have seen a large diffusion in recent years. Many applications relying on these cameras exist in different domains, such as 3D scene reconstruction, augmented reality or Interactive 3D modelling of indoor environments.

2.3.3 Laser Scanner Technology

LiDAR (Light Detection And Ranging) technology is the method used for measuring distances by illuminating the target with lasers light and measuring the reflection with a sensor. The backscattered laser light is collected with a receiver and the distance to the point is then calculated either by time-of-flight or continuous wave (CW) modulation range measurement techniques. Time-of-flight principle is used to measure the distance of a scanned object from the scanner. Basically, the sensor measures the time that the laser takes to travel from the scanner to the object and go back to the receiver. Knowing that light waves travel with a finite and constant velocity in a given medium, we can estimate the distance ρ of the scanned object thanks to the formula

$$\rho = \frac{c}{\nu}\frac{\tau}{2},$$

where c is the speed of light in a vacuum, ν is the refractive index for the light waves that travel in air and τ is the time-of-flight. Concerning a CW scanner a continuous signal is emitted and its travel time can be inferred considering the phase difference between the emitted and the received signal and the period of that signal. Along with distance, vertical inclination and azimuth angle of the laser are used to reference the 3D position of the acquired point with respect to the scanner position. For further details on the principles of laser scanning, please refer to the great book of Vosselman (2011).

Laser scanners use LiDAR sensors to scan the surrounding environment. Objects hit by the lasers will be in the generated point cloud. The position of points is referenced with respect to the position of the scanner. In case of multiple scans in different positions a global reference needs to be determined in order to include all the scans in a global frame. Two strategies exist to achieve this last task. The first uses other sensors to correct referencing the scanners positions. The second is in the post-processing phase, and is a fundamental problem in 3D vision and photogrammetry called Point Cloud Registration. Along with the position of the points, laser scanners measure also the intensity of the backscattered light. When correctly calibrated, the intensity value depends on the kind of material hit by the laser and this information can be useful in the classification of different objects.

Point clouds acquisition systems can be divided into three main groups. Terrestrial Laser Scanner (TLS), Mobile Laser Scanner (MLS) and Aerial Laser Scanner (ALS). This classification is based on the platform on which the system is installed and on the type of sensors employed.

Terrestrial Laser Scanners: are fixed high-resolution scanners typically installed on a tripod or other kind of support. Modern TLSs can acquire over one million points per second at a range that can vary from around 200 meters up to one kilometer. Normally, a single scan takes a few minutes to be acquired and thus the resulting point clouds contain billions of points. In the last years, these scanners are also equipped with camera sensors in order to add colour information. They are typically employed in civil engineering to generate precise 3D models of buildings or civil infrastructures used for example in

quality inspection, construction progress tracking, construction safety management, building renovation and heritage maintenance.

Aerial Laser Scanners: are generally installed on aircraft or on Unmanned Aerial Vehicles (UAV). The generated scans have in general higher spatial resolution than radars. These scanners are employed to obtain virtual city models or digital terrain models (DTM) of certain areas.

Mobile Laser Scanners: are generally mounted on top car, van or other kind of vehicles along with other sensors as for example cameras, Global Positioning System (GPS) and Inertial Moving Unit (IMU). It scans while the vehicle is moving in the traffic. IMU and GPS are used to reference the different scan to a global reference system. These scanners have lower resolution than TLS, but the acquisition process is faster compared to TLS and operation can be done inside the vehicle. The main applications are in Mobile Mapping System for Intelligent Transportation System and in Intelligent Vehicles Technologies as for example Autonomous Cars.

For a more in-depth review on applications of Laser Scanner Technologies, please refer to (Soilán et al., 2019). In our work, we focus on point cloud applications for Autonomous Driving. MLS employed in these cases use multiple laser beams to scan the environment around the vehicle. Generally, the lasers are installed on a shaft, as the teeth of a comb, and the laser spins around the shaft axis during the capture. Rotating around its axis the scanner acquires points in the 360 degree field-of-view around it. In modern research, one of the most employed scanners is Velodyne HDL-64E (see Figure 2.2a), that has 64 laser beams. It scans up to 2.2 millions 3D points per second at a range of 120 meters.

2.4 Point Cloud Processing

In the previous section we have reviewed modern technologies to acquire point clouds. Recent years, mainly thanks to the reduction of production costs of the scanners, have seen large employment of these sensors. 3D data are useful because they provide a more complete information on the surrounding environment compared to images. On the contrary, due to their properties, point clouds require defining geometrical structures in order to navigate and visualize the point cloud.

First of all we need to define neighbourhoods. In literature, there are two main approaches to define a neighbourhood of a point $\boldsymbol{p}$ in a point cloud $\mathcal{P}$. The first is the so called ϵ-neighbourhood. Let ϵ be a real positive number, we define the ϵ-neighbourhood of the point $\boldsymbol{p}$ as the set

$$\mathcal{N}_\epsilon(\boldsymbol{p}) = \{\boldsymbol{q} \in \mathcal{P} \mid d(\boldsymbol{p}, \boldsymbol{q}) \leq \epsilon\}$$

of points of $\mathcal{P}$ closer than ϵ to $\boldsymbol{p}$ with respect to a given distance metric d. Usually, this metric is the Euclidean norm, and in this case, the neighbourhood $\mathcal{N}_\epsilon(\boldsymbol{p})$ is a sphere centred in $\boldsymbol{p}$. For this reason, this kind of neighbourhood is also called *spherical neighbourhood*.
However, other kinds of metrics are also used. For example, let H be a plane and π_H the projection on H.

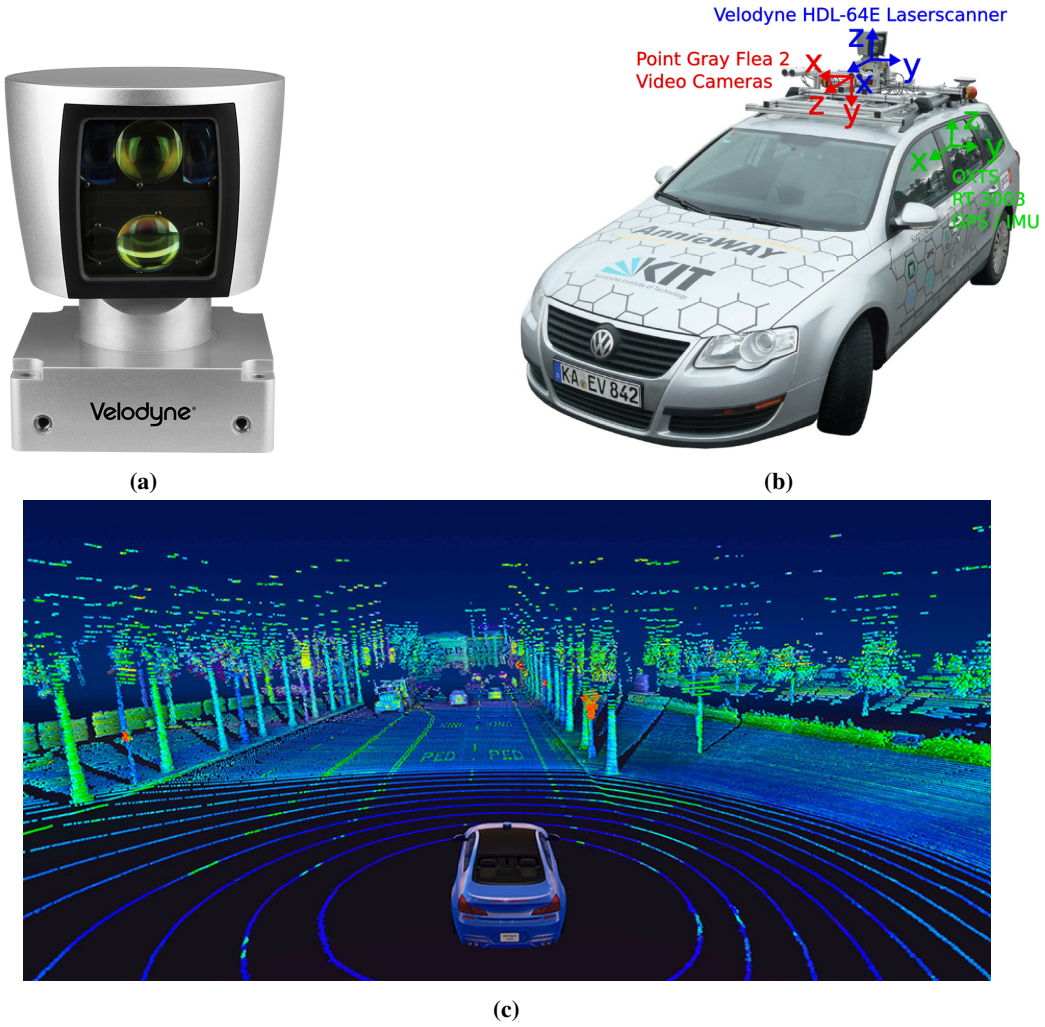


Figure 2.2 (a) Velodyne HDL-64E (b) Car and sensor platform used to record KITTI Dataset Benchmark. A 64 layers laser scanner has been installed on top of the car, along with 4 cameras. (Source Geiger et al. (2012)). (c) Illustration of a scan obtained with a MLS platform.

Consider the distance function

$$d_H(\boldsymbol{p}, \boldsymbol{q}) = \|\pi_H(\boldsymbol{p}) - \pi_H(\boldsymbol{q})\|_2$$

as the distance between the projected points. In this second case neighbourhoods $\mathcal{N}_\epsilon(\boldsymbol{p})$ are cylinders centred in $\boldsymbol{p}$ whose central axis has the same direction of the normal to the plane H, and are called *cylindrical neighbourhood*.

The second strategy is to fix an integer k and consider for each point $\boldsymbol{p}$ its closest k neighbours. This entails that the cardinality of the neighbourhood is fixed by k, and does not depend on the point $\boldsymbol{p}$ or on the local density of the point cloud. For this reason, this kind of neighbourhood are particularly useful when point cloud density is heterogeneous. This neighbourhood is known in literature as *k-Nearest Neighbourhood* (k-NN).

2.4.1 3D Data Structures

In order to compute the neighbourhoods previously defined, we need data-structures that efficiently organize the point cloud and that can guarantee fast access to points. Hereby we present two fundamental data-structures introduced to visualize structures in 3D computer graphics. Both belong to binary space partitioning (BSP) methods, that are methods to recursively subdivide and organize the space in disjoint convex sets using hyperplanes. In this way it is possible to move through different parts of the 3D space navigating among the branches of these trees. The two most commonly used data structures in this field are Octrees and k-D trees.

Octrees

The Octree is a tree data structure useful for space indexing, streaming and data compression, introduced by Meagher (1980). It is an extension of the binary tree to the 3D space. The root of the tree represents and stores information about the bounding box of the entire 3D space aligned to X, Y, Z axis. Using the middle point inside this box and three planes orthogonal to the standard base vectors $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}$, the bounding box is divided into eight parts called *octants*. Each octant is associated to one child of the node, and this subdivision pattern is recursively applied to every internal child until a minimal bounding box size or a minimum number of points is obtained at the leaf level. Finally, the leaves of the octree store the points of the point cloud contained in the bounding box associated with them. This splitting scheme implicitly defines the location of every internal node by its level and its position in the octree. Figure 2.3 from Vo et al. (2015) illustrates an example of a two level octree and shows how the space is decomposed at each level. Burstedde et al. (2011) developed *p4rest* a software library that enables the dynamic management of a forest of octrees.

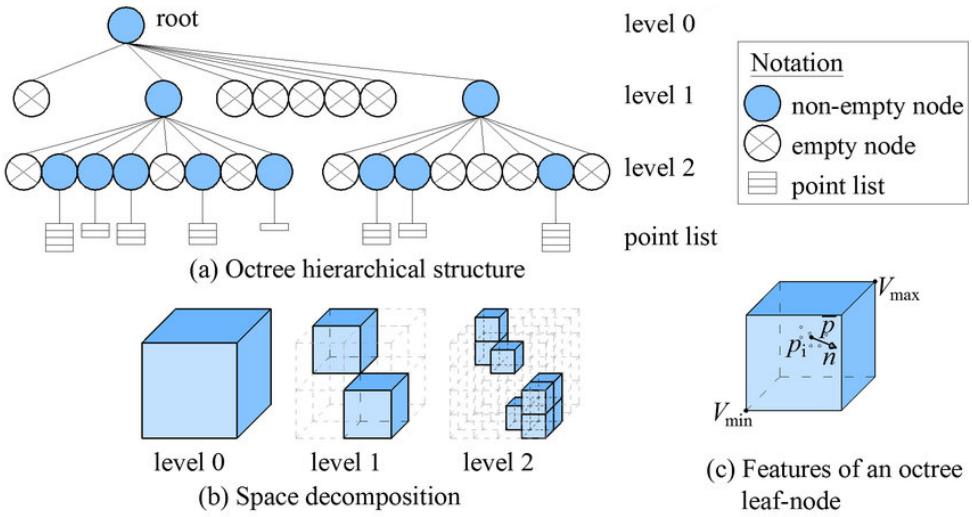


Figure 2.3 Example of Octree decomposition (Image source Vo et al. (2015))

k-D trees

k-D trees, introduced by Friedman et al. (1977), is a space partitioning data structure meant to organize points in a general k dimensional data space. Basically, it is a binary tree in which each internal node defines a cutting hyperplane that splits the space in two parts along one specific coordinate. Points to the left of the hyperplane are represented by the left subtree, while points to the right are stored in the subtree rooted at the right child of the node. The splitting value is set as the median of the distribution of the point coordinates values along the chosen direction. The hyperplane direction is chosen accordingly to the level of depth of the node. The leaf nodes of the trees are k-dimensional points. Nanoflann Blanco and Rai (2014) is a library written in C++ that allows to efficiently build k-D trees.

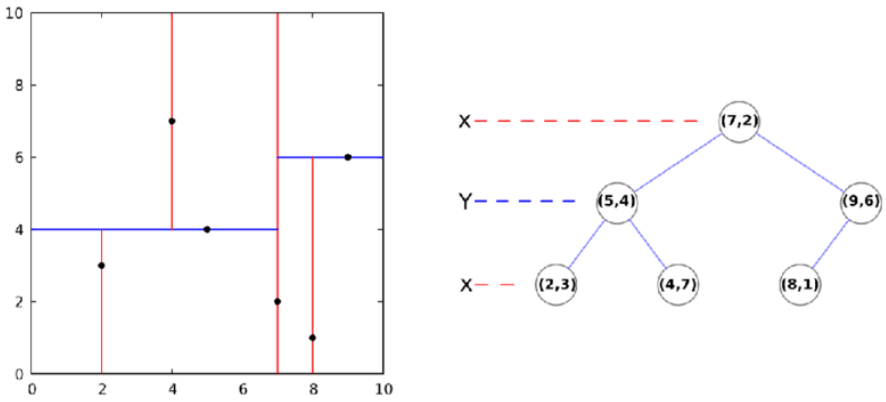


Figure 2.4 Example of space partition using a 2-D tree (Image source Wikipedia)

2.4.2 Feature Extraction

Since, their first applications in the field of civil engineering to generate Building Information Model (BIM) of physical infrastructures, the employment of automatic procedures for the analysis of the point clouds has been necessary to deal with the massive amount of data. This has opened a wide range of problems to solve for example 3D reconstruction, geometric modelling, object recognition, semantic segmentation. In all these problems, feature extraction has played a fundamental role in development of the proposed solutions. In the next lines we review some geometrical features that can be computed starting from a raw point cloud.

A general point cloud can also be seen as a set of sampled points over a given surface $\mathcal{S}$. Hoppe et al. (1992) proposed a method to estimate tangent plane of the local surface at a given point P in the point cloud, that is based on PCA of the local neighbourhood of the point. Given a point $\boldsymbol{p} = (x, y, z)^T \in \mathbb{R}^3$ in a point cloud $\mathcal{P}$, let $\mathcal{N}_\epsilon(\boldsymbol{p})$ be its ϵ-neighbourhood as defined in Section 2.4. Let $\boldsymbol{p}_i = (x_i, y_i, z_i)^T \in \mathcal{N}_\epsilon(\boldsymbol{p})$ points in the neighbourhood, and $\bar{\boldsymbol{p}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{p}_i$, the centre of gravity of $\mathcal{N}_\epsilon(\boldsymbol{p})$ that is made of n points. Given $M = (\boldsymbol{p}_1 - \bar{\boldsymbol{p}}, \ldots, \boldsymbol{p}_n - \bar{\boldsymbol{p}})$, the 3D covariance matrix, also known as 3D structure tensor, is defined as $C = \frac{1}{n} M^T M$. The eigenvectors and eigenvalues of the matrix C describe the directions and the magnitude that maximize the variation of data. In particular, since C is a symmetric and positive semidefinite matrix, and it always exists a matrix U, such that $C = U^T \Lambda U$, where Λ is a diagonal matrix and $U^T U = U U^T = I$. The column vectors of the matrix U are the eigenvectors of the matrix C, while the values $\lambda_1 \geq \lambda_2 \geq \lambda_3 > 0$ in the diagonal of the matrix Λ are the associated eigenvalues. The first

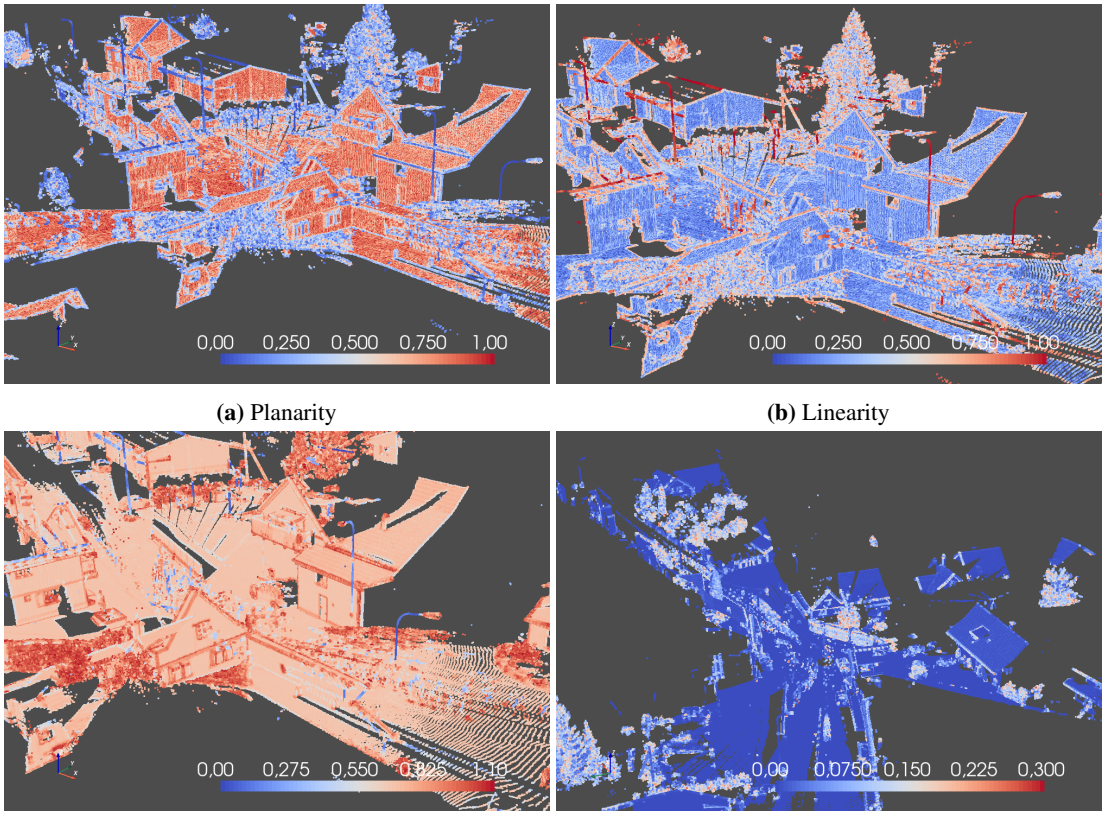
TABLE 2.1
LIST OF GEOMETRICAL FEATURES.

Features	Definitions
Planarity	$\frac{\lambda_2-\lambda_3}{\lambda_1}$
Linearity	$\frac{\lambda_1-\lambda_2}{\lambda_1}$
Sphericity	$\frac{\lambda_3}{\lambda_1}$
Eigentropy	$-\sum \lambda_i \log(\lambda_i)$
Omnivariance	$(\prod \lambda_i)^{\frac{1}{3}}$
Change of curvature	$\frac{\lambda_3}{\lambda_1+\lambda_2+\lambda_3}$

represents the directions that maximize the variation of data, while the seconds represent the magnitude of variation. In particular, the eigenvectors associated to the biggest two eigenvalues also span the least square best fitting plane to $\mathcal{N}_\epsilon(\boldsymbol{p})$. Hence, the third eigenvector is the normal to the estimated local tangent plane. Later on, Demantké et al. (2012), used spectral information on the 3D structure tensor to retrieve local shape information on a point cloud. Since the eigenvalues describe the magnitude of variation along specific directions, they derived linearity, planarity and sphericity features of the local neighbourhood analysing ratio among them. The definition of these features is shown in Table 2.1, while Figure 2.5 illustrates how these features highlight different parts of the point cloud according to the local geometry.

Weinmann et al. (2015), did the same analysis using the k-NN of each point. Moreover, they also proposed other features to describe the geometrical structure of the local neighbourhoods, as for example Eigentropy. This last is used to measure the disorder of points in a 3D covariance ellipsoid, and it is used to find the optimal value of k that defines the neighbourhood size. The optimal value corresponds to the respective k with minimal eigentropy.

The size of the neighbourhood influences the quality of geometrical descriptors. Thus, either we use ϵ neighbourhood or k-NN, an important focus is to put on how to find the optimal size of the neighbourhood (and necessarily of ϵ or k). On the one hand, if the neighbourhood is too small, the descriptors are fast to compute, but they do not really capture the local geometry. On the other hand, if the neighbourhood is too big the entire process could be too costly and important details may be missing. To deal with this problem, Hackel et al. (2016) and Thomas et al. (2018) proposed to use a multi-scale approach. The first used k-NN to define neighbourhoods, while the second used ϵ neighbourhoods. Their solution is based on iterative subsampling of the point cloud, using a voxel-grid filter. The space is firstly divided in voxels of a minimal size, and points in each voxel are replaced with their centroid. Then geometrical features are computed using centroids as points. This strategy induces a feature normalization avoiding a great disparity in the number of points or in the point distance according to point density heterogeneity. This process is iterated increasing the size of the voxels of a factor of 2 at each step and computing the features for each step, until a maximum size is reached. This strategy allows to introduce an intrinsic feature normalization, to reduce the memory footprint and to obtain multi-scale information.



(a) Planarity **(b)** Linearity

(c) Eigentropy **(d)** Change of curvature

Figure 2.5 Planarity, linearity, eigentropy and change of curvature obtained on a point cloud of the Fountain in Balgach in the Semantic3D Dataset Hackel et al. (2017).

2.4.3 Point Cloud Projections

Another way to extract features from a point cloud consists in mapping the points to pixels of a grid defined on a 2-dimensional manifold, such as a sphere, a cylinder or a plane. The main advantage behind this approach is that it allows working on structured data as images, and to reuse all the tools developed for image analysis. The critical step is the choice of the resolution of the grid, that must be chosen carefully to have a good detail of information in the resulting image. In fact, if the grid is too coarse, several points may be projected on the same pixel causing a loss of information, and on the contrary a too fine resolution leads to an image full of empty valued pixels. In the following we recall two kinds of projection that we use later on in Chapter 3.

Bird Eye View

In the Bird's-eye View projection, the input point cloud is projected over a regular grid defined over a plane parallel to the x, y plane. To associate each point $\boldsymbol{p} = (x, y, z) \in \mathbb{R}^3$, to a pixel $p = (u, v) \in \mathbb{Z}^2$, a spatial resolution need to be defined. This resolution defines the size of each pixel. We call Δx and Δy respectively the height and the width of each pixel. Thus, assuming that the x, y axes are oriented in the

same direction of the u, v axis, the pixel p is obtained using the following formula

$$\begin{cases} u = \lfloor (x - x_0)\Delta x \rfloor, \\ v = \lfloor (y - y_0)\Delta y \rfloor, \end{cases}$$

where x_0, y_0 are respectively the minimum values x and y of the points. An example of this projection is shown in Figure 2.7c.

Spherical View

In the Spherical View projection, the points are projected over a spherical grid. Assuming the reference centred at the scanner position, the projection map is defined using spherical coordinates

$$\begin{cases} \rho = \sqrt{x^2 + y^2 + z^2}, \\ \varphi = \text{atan2}(y, z), \\ \theta = \arccos(z/\rho), \end{cases}$$

The grid is built dividing the azimuth and vertical axis. The spatial resolution $\Delta\varphi$ and $\Delta\theta$ define the size of the pixel. The pixel coordinates (u, v) are defined as

$$\begin{cases} u = \lfloor \varphi \Delta\varphi \rfloor, \\ v = \lfloor \theta \Delta\theta \rfloor. \end{cases}$$

Figure 2.6 schematizes the mapping from points to pixels while in Figure 2.7d we illustrate an example of the spherical view projection.

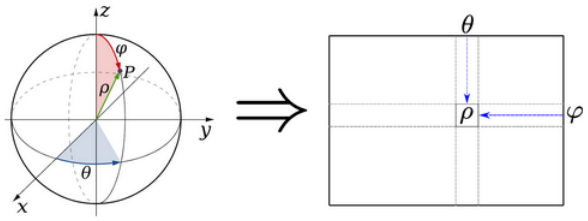
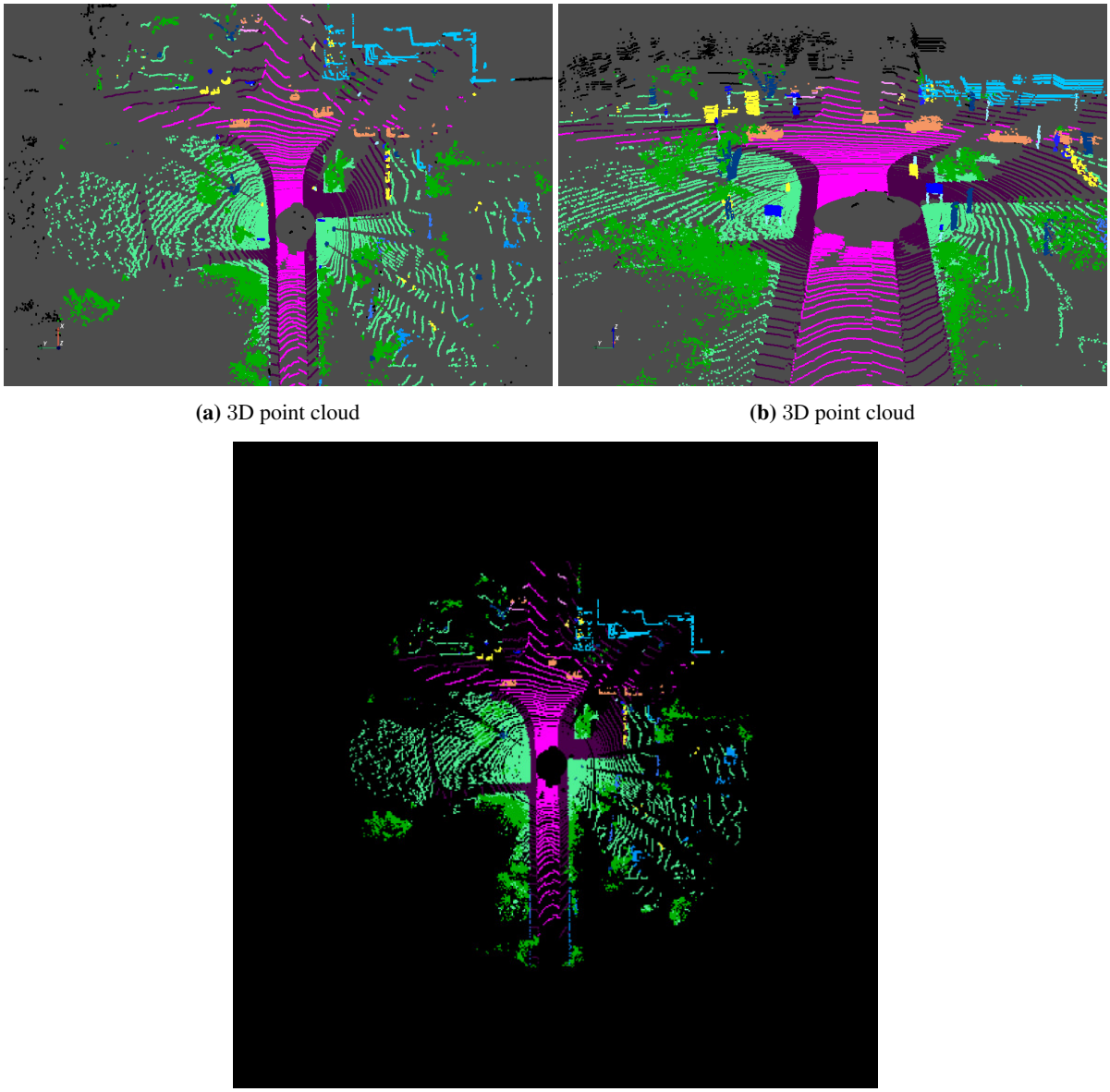


Figure 2.6 Spherical Projection

2.5 Point Cloud for Autonomous Driving

In general, autonomous systems (AS) are characterized by the capability of taking decisions independently by the human interface while facing uncertainty. Autonomous Driving are particular AS that have been developed in the domain of transport. They use sensor systems to capture the surrounding environment and use artificial intelligence to analyse information and take decisions in a continuously changing environment as a substitute for human judgment (Taeihagh and Lim, 2019). The society of automotive engineers (SAE) categories AV based on five levels of automation

- Level 1 (Driver Assistance) The driver and the automated system share control of the vehicle.



(a) 3D point cloud

(b) 3D point cloud

(c) Bird's-eye View projection

(d) Spherical View projection

Figure 2.7 An example of projection of a point cloud onto images. (a, b) Two views of the same 3D point cloud. (c) Image obtained after a Bird's-eye View projection, (d) Image obtained after a Spherical View projection. Some pixels are black because no point falls in.

- Level 2 (Partial Automation) The automated system takes full control of the vehicle: accelerating, braking, and steering. The driver must monitor the driving and be prepared to intervene immediately at any time if the automated system fails to respond properly.
- Level 3 (Conditional Automation) The driver can safely turn their attention away from the driving tasks, e.g. the driver can text or watch a movie. The vehicle will handle situations that call for an

immediate response, like emergency braking. The driver must still be prepared to intervene within some limited time, specified by the manufacturer, when called upon by the vehicle to do so.

- Level 4 (High Automation) As level 3, but no driver attention is ever required for safety, e.g. the driver may safely go to sleep or leave the driver's seat. Self-driving is supported only in limited spatial areas (geo-fenced) or under special circumstances. Outside these areas or circumstances, the vehicle must be able to safely abort the trip, e.g. park the car, if the driver does not take back control. An example would be a robotic taxi or a robotic delivery service that only covers selected locations in a specific area.
- Level 5 (Full Automation) No human intervention is required at all. An example would be a robotic taxi that works on all roads all over the world, all year around, in all weather conditions.

LiDAR sensors acquire crucial 3D information that is more precise compared to images. In order to navigate correctly through traffic and to take accurate decision, perception module of an AS must be able to perform the following tasks on 3D point clouds.

3D Object classification Given a set of different point clouds $\mathcal{X} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and their labels $\mathcal{Y} = \{y_1, \ldots, y_n\}$, belonging to different categories (*e.g.* mug, table, car or aeroplane), we look for a function $f : \mathcal{X} \to \mathcal{Y}$, that correctly assigns each point cloud to its category. For example, we want to associate at the point cloud in Figure 2.8 the category *Aeroplane*.

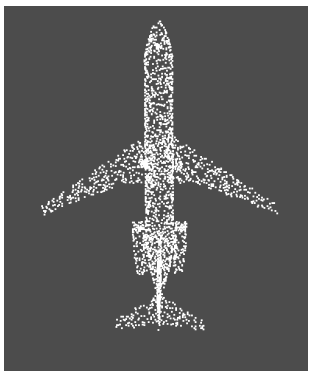


Figure 2.8 Aeroplane

3D Object detection Given an arbitrary point cloud, this task aims to find and locate instances of objects belonging to specific classes (*e.g.* Cars, Pedestrian, Cyclists) and return bounding boxes of the retrieved objects. A bounding box is described as a tuple of parameters $(x, y, z, h, w, l, \theta, c)$, where (x, y, z) are the coordinates of the box centre, (h, w, l) are respectively the height, width and length of the box, θ is the object orientation of the object, and c corresponds to the class of the contained object.

Semantic Segmentation Point cloud segmentation is the task of clustering an input point cloud in disjoint regions where each region is predicted with a semantic label, such as ground, building, car and so on. From a mathematical point of view, given a point cloud $\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ the goal is to fit a function f at each point p that assigns the label y of the corresponding region, that is $y = f(\boldsymbol{p})$. Along with Semantic Segmentation other two kinds of segmentation problems exists in literature, that are Instance

Segmentation (Guo et al., 2020) and Panoptic Segmentation (Sirohi et al., 2021). The first aims to locate and separate the instances of a given category of object (*e.g.* car) in a given point cloud. The main difference with object detection task is that the objective is to return labels for points and not bounding boxes. Instead, Panoptic Segmentation can be seen as a combination of Semantic Segmentation and Instance Segmentation. The goal is to split the point cloud in different semantic regions and for each category separate different instances present in the scene. Figure 2.9 illustrates the difference between these three kinds of problems.

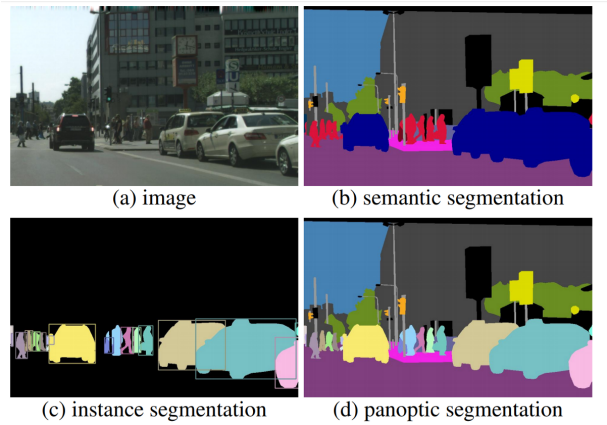


Figure 2.9 An example that illustrates differences between Instance Segmentation, Semantic Segmentation and Panoptic Segmentation

2.6 Databases

In this section, we report a list of some popular point clouds datasets.

2.6.1 SemanticKITTI

SemanticKITTI has been introduced by Behley et al. (2019) as a new benchmark for LiDAR based semantic segmentation. It is built upon the KITTI Vision Odometry Benchmark Geiger et al. (2012) and it provides dense point-wise annotations. It is made of 22 sequences. The authors have proposed a split of sequences in three groups to be used for supervised learning approaches. The first 11 sequences are meant to be used as training set and the last 10 to use as test set. Finally, they propose to use sequence 08 as validation set. Totally the dataset contains around 43k annotated scans. The scans have been collected in different environments such as city centres, country sides or highways and provide a wide range of different situation in which a car drives. Moreover, the scans have always been collected in perfect weather conditions and on flat zones and this makes this dataset a perfect benchmark to test semantic segmentation algorithms. The points are divided in 28 classes. In this great variety of classes there is also a distinction between moving and non-moving objects. The main classes contained in the dataset are:

- Car (Moving / Not Moving)
- Bicycle

- Bus (Moving / Not Moving)
- Motorcycle
- Truck (Moving / Not Moving)
- Person
- Bicyclist (Moving / Not Moving)
- Motorcyclist (Moving / Not Moving)
- Road
- Parking
- Sidewalk
- Building
- Fence
- Lane-marking
- Vegetation
- Trunk
- Terrain
- Traffic-sign

Figure 2.10 illustrates two frames of contained in SemanticKITTI, while Figure 2.11 shows the overall distribution of classes.

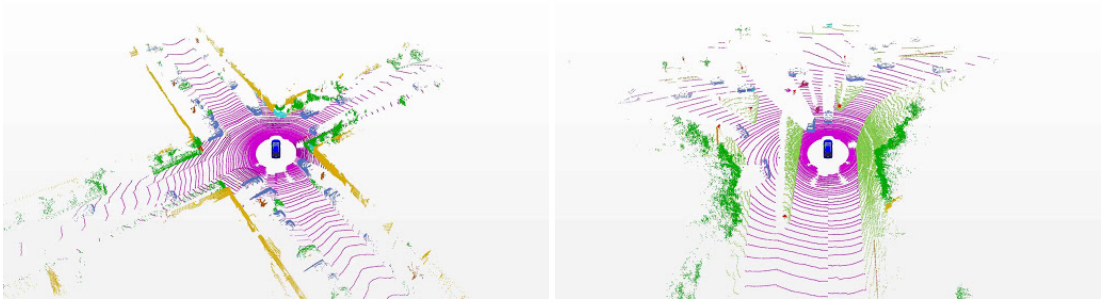


Figure 2.10 Some examples of frames in SemanticKITTI. Source: Behley et al. (2019)

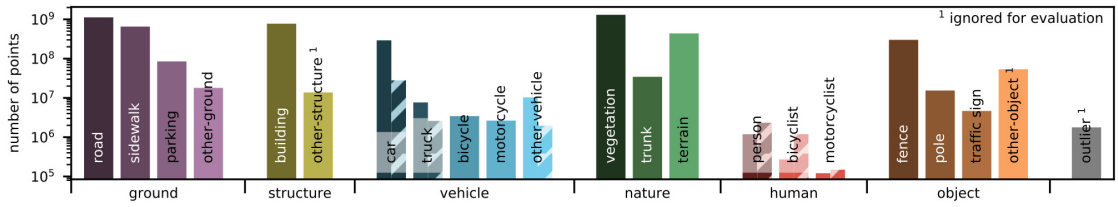


Figure 2.11 SemanticKITTI: Label distribution. Image Source: Behley et al. (2019)

2.6.2 ModelNet

ModelNet is a dataset introduced by Zhirong Wu et al. (2015). The full dataset is a collection of almost 151k CAD models belonging to 660 unique object categories, among the most common object categories in the world. There exist also two variants of this dataset respectively restricted to 40 and 10 categories. The first is called ModelNet40 while the second ModelNet10. In Figure 2.12 some CAD examples of chairs contained in ModelNet40 are depicted.

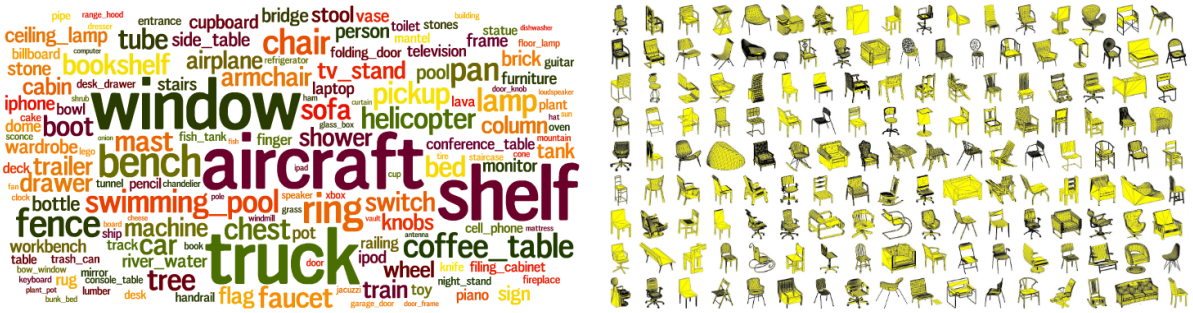


Figure 2.12 Left: word cloud visualization of the ModelNet dataset based on the number of 3D models in each category. Larger font size indicates more instances in the category. Right: Examples of 3D chair models. Image Source Zhirong Wu et al. (2015).

2.6.3 PartNet

The PartNet dataset Mo et al. (2019) is a benchmark that contains fine-grained and hierarchical instance-level part segmentation annotation for about 26k shapes with over 500k part instances from 24 object categories. Some examples of fine-grained segmentation and of hierarchical decompositions are respectively shown in Figures 2.13 and 2.14. The categories have been selected from ShapeNetCore Chang et al. (2015), and respects the following requirements:

1. are mostly seen in indoor scenes,
2. contain interesting intra-class variation
3. provide a huge number of parts

The median number of parts per object is 14 and a maximum number of 230,

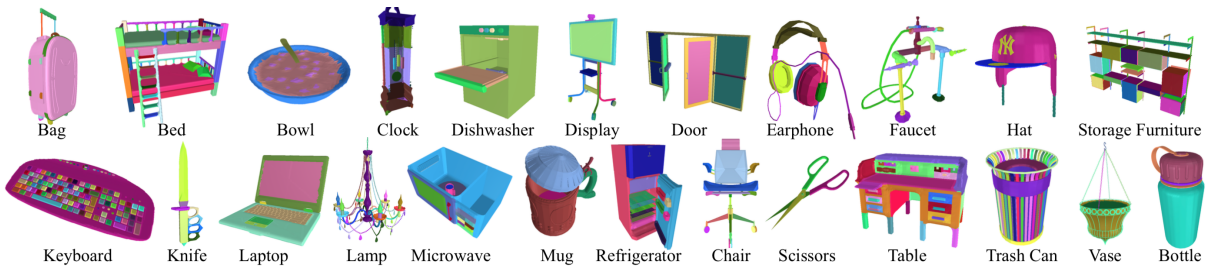


Figure 2.13 Some shapes with fine-grained part annotations for the 24 object categories in PartNet. Image Source Mo et al. (2019).

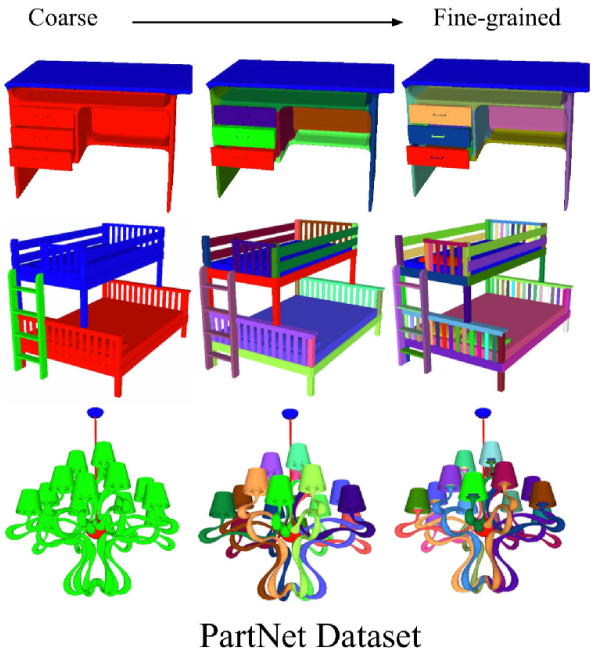


Figure 2.14 Some annotations at three levels of segmentation in hierarchy. Image source Mo et al. (2019).

3 Ground and Road Detection

Resumé

Le chapitre est divisé en deux parties. La première partie se concentre sur le problème de l'identification des sols dans le contexte d'un nuage de points obtenu à l'aide d'un scanner Velodyne monté sur une voiture. La contribution innovante de cette partie est constituée de deux algorithmes de détection des sols par l'utilisation de zones quasi plates. Pour évaluer l'efficacité de ces méthodes, une comparaison avec l'état actuel de la technique est également présentée. La deuxième partie se concentre sur le problème de la détection des routes, toujours dans le même contexte. En particulier, une analyse des performances des méthodes actuelles de détection par réseaux neuronaux sur des scanners à basse résolution est développée. Pour améliorer les performances de ces méthodes, l'utilisation d'informations géométriques telles que la normale à la surface est proposée. Les résultats obtenus montrent que l'utilisation de ces caractéristiques permet d'améliorer la détection, notamment dans le cas de scanners à basse résolution.

3.1 Introduction

Ground detection is a fundamental problem to solve for several applications such as 3D modelling process and mobile robot navigation. Our interest in ground detection is motivated by the fact that once removed the ground from the scene, other objects can be identified as isolated components of the scene. Indeed, this strategy is employed in many object detection or object classification algorithms. Interestingly, ground detection is also of interest in the case of Digital Surface Models obtained by radar interferometry, LIDAR, or photogrammetry. Indeed, these techniques provide an estimation of the height of any object above the ground such as vegetation (trees etc.) and buildings rather than the ground elevation. The latter elevation is needed for many applications such as those related to hydrology or civil engineering. Note that in this case (apart from LIDAR), the input data are provided as a raster (2.5 D) and not 3D point cloud.

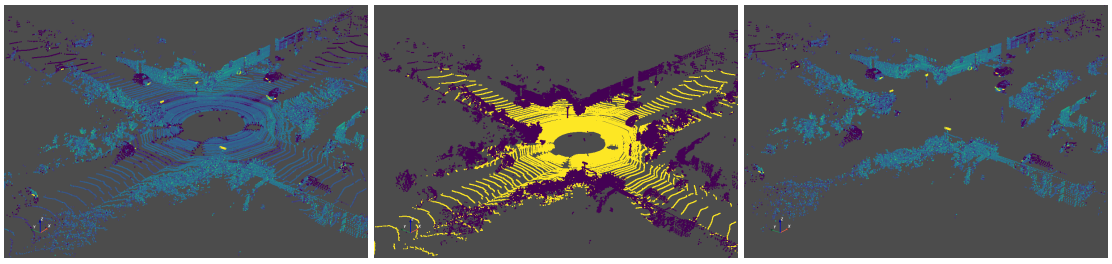
This chapter is divided into two parts. In the first part, we discuss various approaches for the ground detection task. To be specific, we compare several state-of-the-art methods like RANSAC, λ-flat zones, and Convolutional Neural Network (CNN) methods. For each of them, we use different point cloud

representations. In some cases, we will project on 2.5D images, in some others, we can directly work with a 3D representation.

In the second part of the chapter, we focus on road detection. The main motivation is to evaluate the effect of subsampling the number of layers of the scanner. Until these days, a significant majority of the state-of-the-art algorithms in the context of self-driving car applications, have been tested on scenes acquired using high-resolution scanners such as Velodyne HDL-64E. Despite that, the use of this kind of scanners in a real-world scenario is limited due to their considerable costs of production. Even though these costs are decreasing, the employment of high-resolution scanners on a car still remains too expensive. On the other hand, low-resolution scanners are more competitive but provide a smaller amount of information compared to a high-resolution device. For this reason, we analyse the effect of the reduced resolution on a fundamental task such as road detection.

3.2 Introduction to Ground Detection

A common approach in 3D object detection and 3D object classification approaches is to detect the ground as the first step (Roynard et al., 2016; Serna and Marcotegui, 2013, 2014). The idea is motivated by the fact that once removed the ground from the scene, all the other objects in the scene appear as different connected components. The pipeline is illustrated in Figure 3.1, then continues analysing and classifying the remaining components.



(a) Input Point Cloud **(b)** Ground Detection **(c)** Removed Ground

Figure 3.1 A common pipeline for Object classification. Ground detection is the first step to achieve. Once removed from the ground remaining objects can be more easily identified.

Many approaches for ground detection have been proposed in the literature during the previous decade. A simple attempt to solve this problem is to model the ground as a flat surface and carry out a planar approximation using RANSAC paradigm introduced by Fischler and Bolles (1981). Examples of RANSAC based approaches are Gallo et al. (2011); Oniga et al. (2007); Schnabel et al. (2007). Even though those methods are robust to outliers, the assumption of a unique flat ground is not realistic even in the urban context. To solve this problem, Hernandez and Marcotegui (2009); Serna and Marcotegui (2013) proposed to use λ-flat zones to detect the ground in dense point clouds. The method projects the point cloud on a regular grid parallel to the xy plane placed at the lowest value of z coordinate, and storing for each grid cell the value of the minimal elevation among all projected points on the same pixel. This is the

BEV described in section 2.4.3. Once obtained the projected images the segmentation via λ-flat zones is carried out to obtain the ground. Similarly, Roynard et al. (2016) project points on a discrete horizontal grid and the z value with the highest value in the histogram is selected as ground seed. Then a region growing approach is used to detect the ground. Both methods are very similar: lambda flat zone and region growing approaches rely on the same hypothesis of smooth height variation. The unique difference is the initialization step. These methods were proposed for a mobile mapping application with a relative density homogeneity. Unfortunately, this assumption does not hold for standard autonomous driving applications. In that case, the scanner is mounted on top of the car and the axis of the scanner is orthogonal to the ground. The resulting point density decreases with the distance from the scanner. In this kind of scenario, it is not possible to find a good resolution value. In fact, a sufficiently high resolution disconnects object profiles in the projected image. On the other hand, a low resolution accumulates too many points in pixels closer to the scanner where the point cloud density is high and aggregating information we risk to cancel out small objects that we need to detect, such as people or bicycles. In Section 3.3 we present a method that copes with this problem and accurately interpolates missing information. A more recent method has been introduced by Zhang et al. (2016). Their idea is to turn upside down the point cloud and let drop a cloth to the inverted surface from above. The ground is then detected analysing the intersections between the nodes of the cloth and the inverted point cloud. Finally, in recent years several CNN-based methods have been introduced in the more general problems of semantic segmentation of a 3D point cloud (Hu et al., 2020; Landrieu and Simonovsky, 2018; Thomas et al., 2019). Concerning the ground detection task Velas et al. (2018) propose to project the point cloud using a spherical view (see section 2.4.3) and generate 2D images containing range, z and laser intensity values. The resulting images are then used to train a Fully Convolutional Neural Network (FCNN) to obtain a binary segmentation. Finally, the labels are back projected to 3D points. This kind of approach has been also used by Behley et al. (2019) and Milioto et al. (2019) to carry out a semantic segmentation of the scene.

3.3 Ground Detection on Point Clouds with heterogeneous density

We now present two novel methods that use λ-flat zones to detect ground on Point Clouds with heterogeneous density. The two methods differ on data representation used. The first method is based on the work of Hernandez and Marcotegui (2009) that we discussed above, and aims to solve the problems deriving from the high variation of point density in the scene. The second method works directly on 3D point clouds, defining a graph and successively extracting λ-flat zones.

On BEV images

Hereunder, we present a method based on the work of Hernandez and Marcotegui (2009). As we said hitherto, the critical issue that we experience represents the variation of point density. In fact, in these point clouds, the density decreases with the distance to the scanner. This means that projecting the points on a squared grid defined over the xy plane, the pixels far from the scanner have a higher probability to be empty. This causes a problem of connection between peripheral pixels. We have considered two strategies to address this issue. The first and the simplest approach is to reduce the grid resolution. In this way, the surface of each element of the grid is bigger, and this increases the probability that at least one point falls in. The main drawback is that the projection could merge information relative to different objects

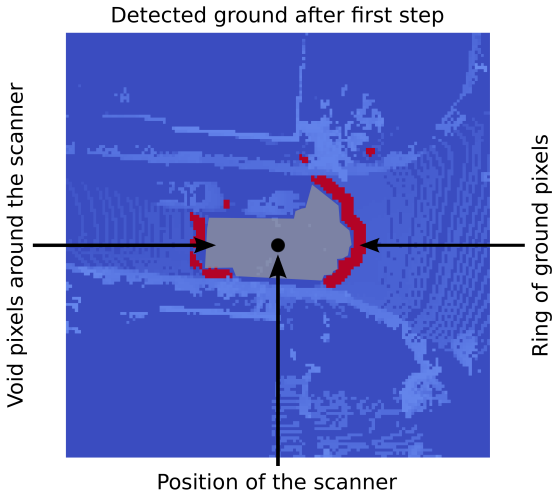


Figure 3.2 A zoom of the I_{max} image. In this case, the car is driving through a narrow street, and road in the front of the car is disconnected from the rear. In red, pixels in the closest ring around the scanner detected as ground.

in places where the point cloud density is high. Thus, we need to find a trade-off between the amount of information merged and the number of pixels reconnected. Unfortunately, we found out that this first approach is not sufficient to solve the connection problem and for this reason, we moved on to the second solution. It consists of splitting the xy grid as a particular polar grid, that we will introduce later on, and interpolate values in each circular sector. The method uses the following BEV images:

- I_{min} that stores the minimal elevation (vertical distance from each 3D point to the projection plane) among all projected points on the same pixel,
- I_{max} that stores the maximal elevation among all the projected points on the same pixel,
- I_{acc} that stores the number of points projected on each pixel.

To obtain these images we use a resolution of 5 *pixels/m* for the xy grid, that is, the size of the pixel side is 20 *cm*. Along with this, the BEV images are 8-bit encoded images and the resolution used for the elevation is 10 *levels-of-gray/m*. Given the importance of the task we aim to solve, for security reasons, we prefer a high precision-score in our detection rather than a high recall. This means that false negatives are preferred to false positives. For this reason, differently from the original work in which the I_{min} image was used, we interpolate and segment the I_{max} image. This gives us higher confidence in the detected ground. The method can be divided into the following steps:

1. identify the ground around the scanner,
2. build a polar grid and interpolate values,
3. compute λ-flat zones and extract ground on BEV image,
4. back project ground label from BEV image to 3D points.

1. Identify the ground around the scanner The first step is to retrieve the part of the ground closest to the car. The goal is to reconnect the road in front of the car with the one behind. In the original method,

the ground is identified as the biggest λ-flat zone found after segmenting the image. In situations where the car is navigating through narrow streets, this assumption may not be verified, just because the ground in front of the car could not be connected with the ground in the rear, as shown in Figure 3.2. In the proposed example, pixels in the sides of the car represent either a wall or other cars, the ground in the front is isolated from the one behind. To solve this problem, we detect the ground among the pixels in the closest ring around the car. These pixels will be used later on as markers to detect which λ-flat zones that belong to the ground and merge them together.

We start identifying the void pixels around the scanner using a morphological reconstruction by dilation. We use as marker image f:

$$f(x,y) = \begin{cases} 255 & \textit{if } (x,y) = (x_0, y_0), \\ 0 & \textit{otherwise.} \end{cases}$$

where (x_0, y_0) is the pixel corresponding to the position of the scanner in the image. Furthermore, we use as mask image g:

$$g(x,y) = \begin{cases} 255 & \textit{if } I_{acc}(x,y) = 0, \\ 0 & \textit{otherwise.} \end{cases}$$

Thus, the image I_c containing the identified circle is obtained as $I_c = R_g^{\delta}(f)$. Then, we detect among the points in the closest ring around the car those belonging to the ground. To achieve this, we first locate the ring $\mathcal{R}$ around the car applying a morphological external gradient defined as:

$$I_r = \delta_B(I_c) - I_c,$$

where B is a structuring element of size 5×5. The ring is the set $\mathcal{R} = \{(x,y) \mid I_r(x,y) = 255\}$. Then we compute

$$z = \min_{(x,y)\in\mathcal{R}} I_{max}(x,y),$$

the smallest z value in I_{max} on the set $\mathcal{R}$. Finally, we assign as ground only the pixels in the ring $\mathcal{R}$ such that $|I_{max} - z| < 0.5m$. In Figure 3.2 we mark in red the resulting detected ground.

2. Build Dart Board and Interpolate image In the second step, we interpolate information contained in the I_{max} image. This is a necessary step in the method because it fills information on void pixels. Namely, we define another grid, and then we map pixels of I_{max} image onto its elements. To better explain our choice, let us analyse how points are spatially located in an ideal environment where the ground is a plane orthogonal to the axis of the scanner. As said before, the scanner has several lasers, called *layers*, that acquire points around the car. In this setting, the scanner has 64 layers and the inclination angle of each layer is fixed. During the acquisition phase, the scanner spins around its vertical axis. Looking at points for a fixed yaw angle as in Figure 3.3, we can see that the distance of the points from the scanner grows with the tangent of the vertical angle of the layers. Thus, in this context, a polar grid on the xy plane, where the length of intervals in the radial axis increases with a tangential trend, would be better suited than the Euclidean grid to prevent disconnections.

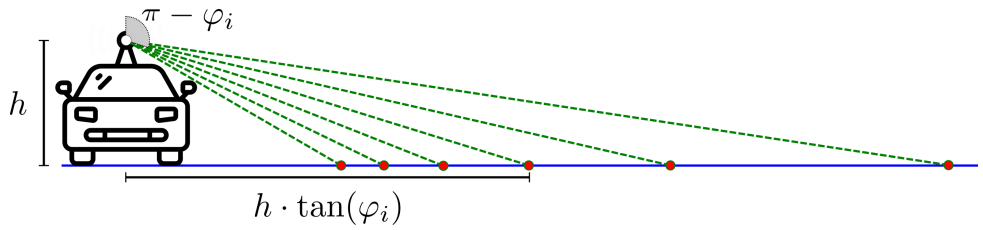


Figure 3.3 The hypothetical case of perfectly flat ground. The distances between points and the scanner depend on the tangent of the inclination angle of the layer and the scanner height h.

To define the intervals in the radial axis, let first consider $l_1, \ldots, l_n$ layers in the scanner directed towards the ground, and let $0 \leq \varphi_1 \leq \ldots \leq \varphi_n \leq \frac{\pi}{2}$ the respective inclination angles of the layers. Furthermore, let h be the distance between the scanner and the ground. In the hypothesis of an ideal environment, we can estimate the radial distances r_i of the scanned points as:

$$r_i = h \cdot \tan(\varphi_i), \quad \forall i = 1, \ldots, n.$$

Hence, we split the radial axis with intervals $[r_i, r_{i+1})$, for $i = 0, \ldots, n+1$, where $r_0 = 0$, and $r_{n+1} = \infty$. In this way, the profile of the ground in the grid remains connected because for each cell we have at least one point that falls in. Differently from the radial axis, we choose $0 \leq \theta_1 \leq \ldots \leq \theta_m \leq 2\pi$ angle to evenly split the polar axis. Thus, each element $\mathcal{S}$ of the dartboard is defined as the set of pixels in the product $[r_i, r_{i+1}) \times [\theta_j, \theta_{j+1})$. Figure 3.4 shows the dartboard obtained for the setup used in KITTI Benchmark (Geiger et al., 2012) where the scanner has been placed at height of $h = 1.73m$. As the reader can see, the length of the circular sectors along the radial axis increases moving away from the center of the grid, where the scanner is placed.

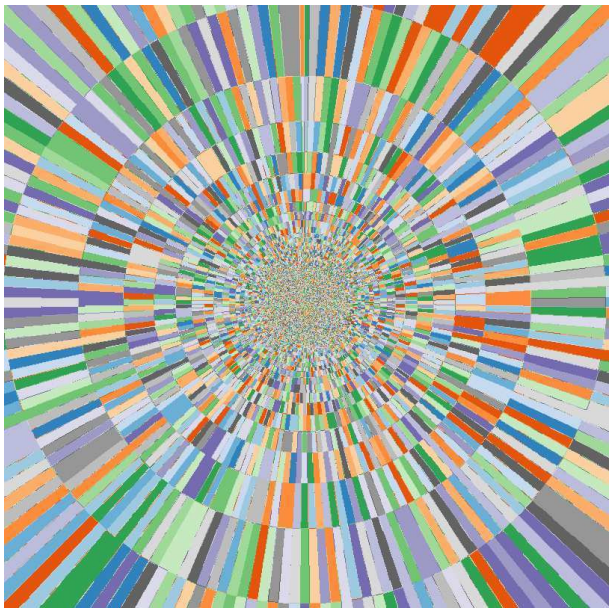
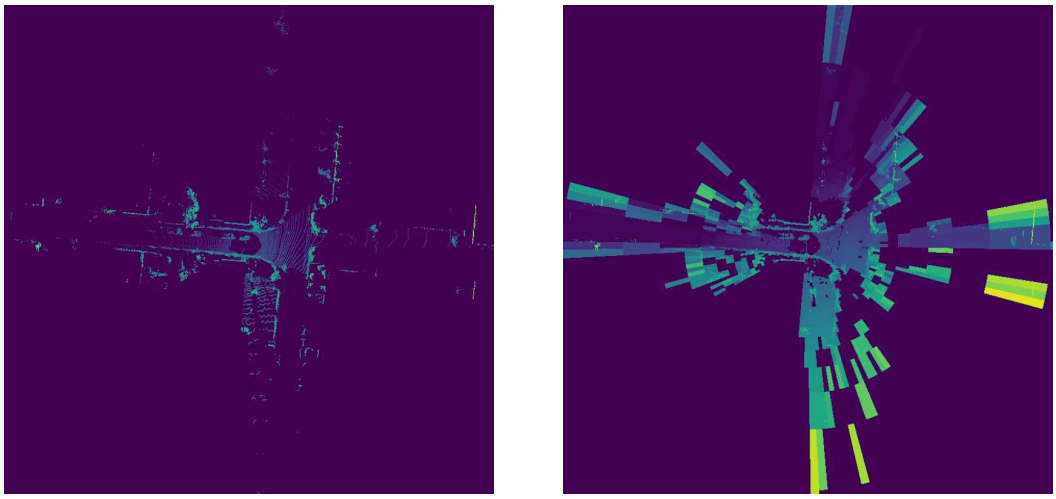


Figure 3.4 Dartboard

Once generated the dartboard, we employ it to interpolate information on void pixels in the I_{max} image and obtain the interpolated image $\hat{I}_{max}$. First of all, we identify the center of the dartboard with the



(a) Example of I_{max} **(b)** Interpolated image $\hat{I}_{max}$ on the dartboard

Figure 3.5 Interpolated image obtained on the frame 3721 in sequence 08 of SemanticKITTI dataset.

position (x_0, y_0) of the scanner in the image. Then for each pixel x, y in the Euclidean grid we compute its polar coordinates r, θ as:

$$\begin{cases} r = \sqrt{(x-x_0)^2 + (y-y_0)^2}, \\ \theta = arctan((y-y_0)/(x-x_0)). \end{cases}$$

The coordinates (r, θ), determine a circular sector $\mathcal{S}$ in the polar grid. In this way, we map each element in the I_{max} domain to an element in the dartboard. Now, let us define the image $\hat{I}$ in which we assign, for each circular sector $\mathcal{S}$ in the dartboard, the minimum value of the I_{max} image among all pixels in $\mathcal{S}$. In formula, the image $\hat{I}$ is defined as:

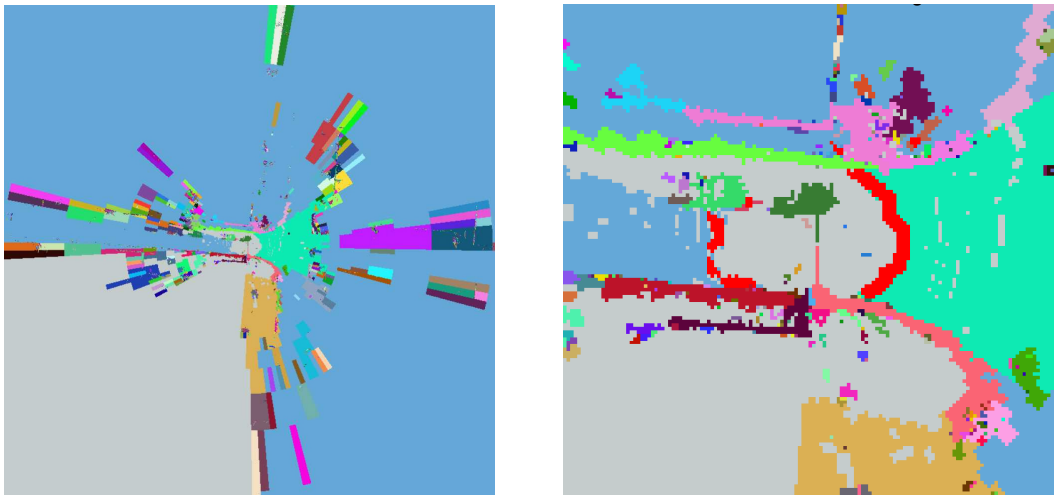
$$\hat{I}(\mathcal{S}) = \min_{(x,y)\in\mathcal{S}} I_{max}(x, y).$$

Since $\hat{I}$ could not contain information relative to vertical objects, we need to recover it. Therefore, we compute the max between values in I_{max} and $\hat{I}$, and obtain the final interpolated image $\hat{I}_{max}$. In other words, for each pixel (x, y) in the image domain, we assign

$$\hat{I}_{max}(x, y) = max(I_{max}(x, y), \hat{I}(x, y)),$$

i.e. only empty pixels are interpolated. In Figure 3.5 we illustrate an example of an image $\hat{I}_{max}$ obtained interpolating values from I_{max} image using dartboard grid in Figure 3.4. Note how peripheral pixels that were disconnected in the I_{max} image are reconnected in the $\hat{I}_{max}$.

3. Compute λ-flat zones and extract ground on BEV image After having interpolated the I_{max} image, we compute λ-flat zones on the $\hat{I}_{max}$ image. Similarly to Hernandez and Marcotegui (2009), we use $\lambda = 0.20m$ as a discrimination value to separate the pavement from other objects. An example of the obtained λ-flat zones is illustrated in Figure 3.6a. Note that in the proposed example, the car is navigating through a narrow street and the road is divided into two main λ-flat zones. To merge the two connected components, we use the marker that we have extracted in the first part of the method. Figure 3.6b shows a



(a) (b)

Figure 3.6 (a) Quasi-flat zones obtained with $\lambda = 0.2m$. (b) Zoom around the car. Red pixels represent the ground detected in the first step of the method.

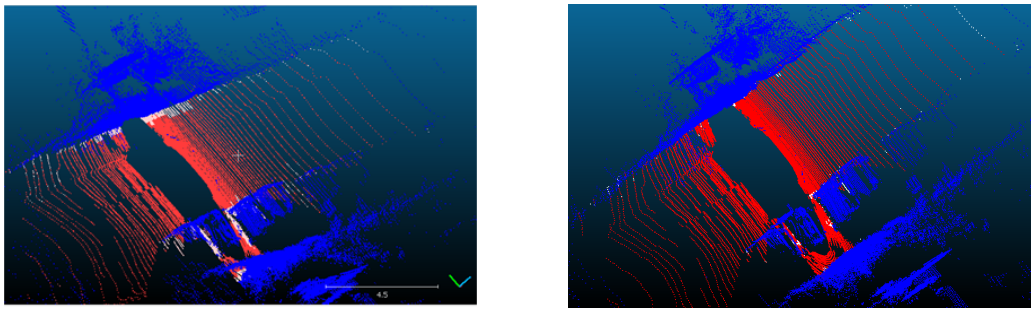
zoom of the obtained segmentation around the car. The red ring in the center of the image is the ground marker detected at the beginning of the method. In fact, we label as ground all the λ-flat zones whose intersection with the detected ground marker is not empty. Thanks to this, we can reconnect the various connected components. At the end of this step, the method returns a binary label BEV image I_g, whose non-zero pixels represent the detected ground so far.

4. Back project labels Finally, the detected ground pixels must be projected back to 3D point clouds. As we said at the beginning of this section, we use the I_{max} image values to extract the ground. This differs from the original method, in which the authors used I_{min} image. Even though the two approaches seem similar, there is a significant difference between the two that we need to consider before the back projection of the labels. In fact, using the I_{max} images does not allow detecting points close to vertical objects. The reader may refer to Figure 3.7a as an example. Projecting back labels detected on I_{max}, we miss ground points close to vertical objects (white points in figure). The cause is that ground points fall in the same pixels of vertical objects, so z values contained in the I_{max} image refer to vertical objects. In our particular case, this effect is accentuated by the gross resolution ($20cm$) that we have set to avoid disconnections. This effect is proportional with the resolution of the xy grid (the coarser the resolution, the higher the effect). To solve the problem, we use I_{min} image to expand the ground detected before projecting it back.

First of all we compute λ-flat zones on I_{min} image, using a value $\lambda = 10\ cm$. Then, let us define $\bar{I}_g$ the extended ground image as:

$$\bar{I}_g(\mathcal{C}) = \max_{(x,y)\in\mathcal{C}} I_g(x,y),$$

for each quasi flat zone $\mathcal{C}$ obtained from I_{min}. Intuitively, we propagate ground labels on λ-flat zones computed on the I_{min} image. In this way, we assign as ground pixels containing both ground points and points belonging to vertical objects. Hence, during the back projection, we need to separate this last group of points. To achieve this, let us consider $\mathcal{F} = \{(x,y)|I_g(x,y) = 1\}$ a subset of the image domain made by pixels initially marked as ground, and let us consider $\bar{\mathcal{F}} = \{(x,y)|I_g(x,y) = 0 \wedge \bar{I}_g(x,y) = 1\}$ a subset



(a) **(b)**

Figure 3.7 (a) Results obtained projecting back ground-detected on I_{max}. We assign red colour to the true positive, blue colour to true negative, and white colour to false negative. Without propagating labels on I_{min}, we miss ground points close to vertical object. This issue is mainly caused by the fact that we have chosen a gross resolution of $20cm$ for the xy-grid. (b) Results obtained after the expansion of the detected ground before the projection.

of the image domain made by pixels where the ground label has been extended. Let p be a point and let (x_p, y_p) the pixel in the image domain where p is projected. If $(x_p, y_p) \in \mathcal{F} \cup \bar{\mathcal{F}}$ then the point p may belong to the ground. To decide if p belongs to the ground or not, we consider the difference between its elevation and the corresponding value in I_{min} image, *i.e.* $|p_z - I_{min}(x_p, y_p)|$. Two different threshold values, respectively $\delta_{\mathcal{F}} = 20\ cm$ and $\delta_{\bar{\mathcal{F}}} = 5\ cm$, are defined according to whether an object has been detected in the pixel initially detected as ground ($\mathcal{F}$) or not ($\bar{\mathcal{F}}$). If so, the tolerance is lower in order to prevent the inclusion of the lower part of the object into the ground. The label $l(p)$ assigned to the point p is:

$$l(p) = \begin{cases} 1 & \textit{if } (x_p, y_p) \in \mathcal{F} \wedge |p_z - I_{min}(x_p, y_p)| \leq \delta_{\mathcal{F}}, \\ 1 & \textit{if } (x_p, y_p) \in \bar{\mathcal{F}} \wedge |p_z - I_{min}(x_p, y_p)| \leq \delta_{\bar{\mathcal{F}}}, \\ 0 & \textit{otherwise.} \end{cases}$$

In Figure 3.7b we illustrate ground detected after this operation of propagating the labels over the I_{min} image.

On 3D point clouds

In this section, we avoid the use of a BEV image, that squeezes a part of the information available during projection. In this case, a graph is defined directly on the 3D point cloud. The nodes of the graph $\mathcal{G}$ are the 3D points. The most popular ways to connect the points are based on how neighbourhoods are often defined in the context of 3D point clouds. These are k-NN and ϵ-ball. In the first case, each point is connected with its first k nearest neighbours, while in the second case, each point is connected with any other point closer than ϵ. Clearly, the choice of the neighbourhood should be based on the specific application. As aforementioned, in the case studied, the scanner is placed on the top of a car, and it spins around its axis. Hence, another definition of neighbourhoods could be inherited from this configuration. In the proposed solution, we consider the spherical projection. Essentially, each point is projected in the spherical grid (Section 2.4.3) and then connected to every other point projected in neighbouring pixels. Intuitively, this is a way to pull back the 4-connection defined on the spherical image to the 3D point cloud. In Figure 3.8 we show the way we establish the connections using spherical projection.

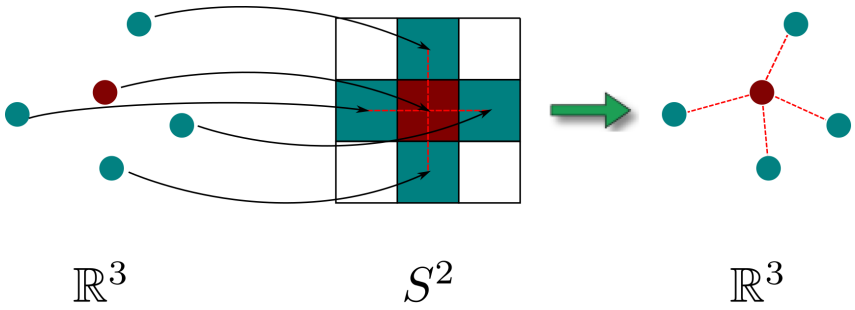
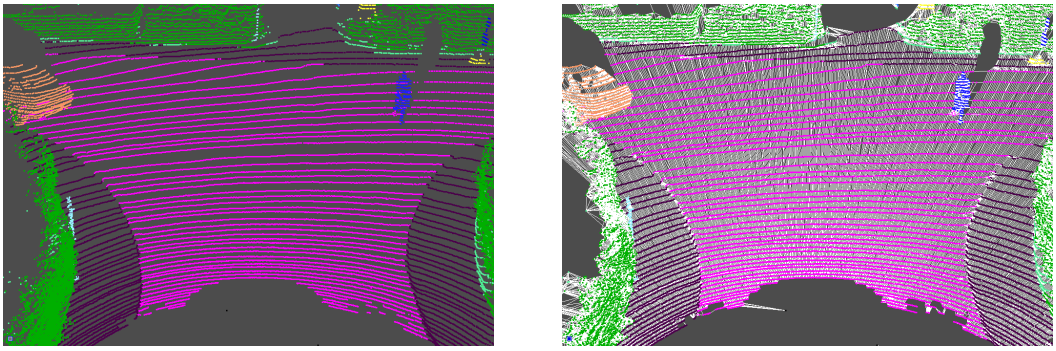


Figure 3.8 The graph is obtained projecting points on the spherical grid S^2.



(a) Input point **(b)** 3D graph

Figure 3.9 An example of the graph built on the 3D point cloud. The colours of the points change according to the class.

Unfortunately, this kind of connection suffers from occlusion effects. In fact, it could happen that one vertical object in the foreground occludes objects in the background, blocking connections between points in the background objects. We try to mitigate this effect adding to this graph also the k-NN graph. Moreover, we prevent connections with outliers removing all edges longer than five meters. In Figure 3.9 we illustrate an example of the 3D graph obtained starting from an input point cloud.

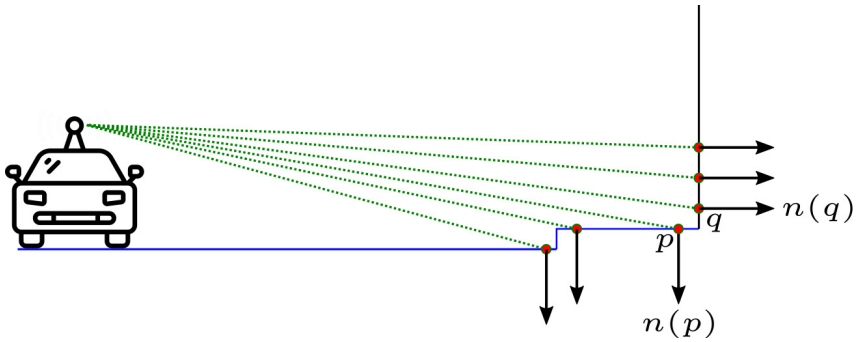


Figure 3.10 An example of scanner

Once determined the graph $\mathcal{G} = (V, E)$, we need to define a weight function $w : E \to \mathbb{R}$, that we can use to split points on the ground from others. Since, we are going to extract λ-flat zones on the weighted graph $(\mathcal{G}, w)$, we look for a w that assigns a high value to edges connecting ground and not-ground nodes. In the previous case, we measured the difference along the z coordinate between two pixels. This time we cannot adopt the same approach. In fact, looking at a vertical profile, as in Figure 3.10, we observe that a

slow variation along z appears when the laser hits vertical objects. Consequently, if we used a weight function that takes into account only the difference in the elevation between two points, then there would be no λ value that splits the ground from the rest. For this reason, we propose to include into the weight function information related to both elevation z of the points and the local horizontality of the surface. Let introduce the function w using the example in Figure 3.10. Assume that $p, q \in \mathcal{P}$ are two neighbouring points and let $n_p, n_q \in S^2$, the corresponding vectors normal to the local scanned surface at point p and q. Furthermore, let us assume that p belongs to the ground while q represents a point on a wall. In order to assign a considerable value to the edge (p, q), we can look at the ratio $r(p) = \frac{p_z}{|\langle n_p, \hat{z} \rangle|}$. This ratio goes to infinity as $\langle n_p, \hat{z} \rangle$ goes to zero. To eliminate the problem of defining the ratio when $\langle n_p, \hat{z} \rangle = 0$, we can apply a logistic function $f(x; k, x_0) = \frac{1}{1-exp(-k(x-x_0))}$, to the value $|\langle n_p, \hat{z} \rangle|$. In Figure 3.11, we illustrate how the logistic function changes when we alter the parameters k or x_0.

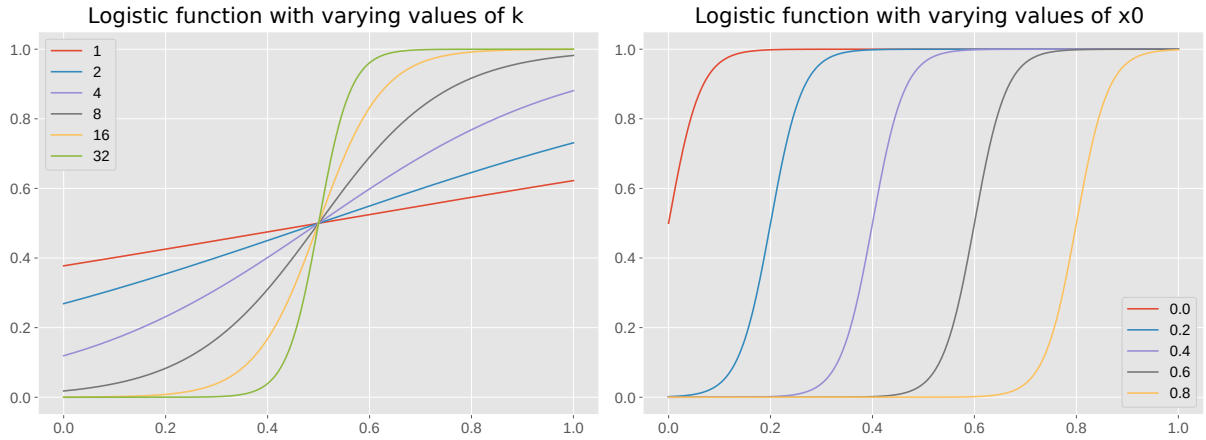


Figure 3.11 Logistic function with varying values of parameters k and x_0.

Thus, we define the weight function $w(p, q) = \left| \frac{p_z}{f(|\langle n_p, \hat{z} \rangle|)} - \frac{q_z}{f(|\langle n_q, \hat{z} \rangle|)} \right|$. The main idea behind, is that if the points p, q both belong to a horizontal surface, then the weight of the edge (p, q) is proportional to the difference between the elevation of the two points. This is a nice property of w that will be useful later on during the choice of the λ. On the contrary, if one of the two belongs to a vertical surface, then the weight becomes high.

The consecutive step is to compute λ flat zones of the weighted graph $(\mathcal{G}, w)$. In our experiments, we choose a value of $\lambda = 0.20m$. As introduced before, our choice is because, given two points p, q lying on a horizontal surface, w assigns to (p, q) a weight proportional to the difference in the elevation of the two points. Accordingly, assuming the ground as a horizontal surface, we have chosen a threshold comparable with the height of a step. In Figure 3.12 we show an example of λ-flat zones obtained. The weighted function defined above groups together points belonging to big flat horizontal surfaces, while vertical objects are shattered in small connected components. At this point, we have to analyse the obtained cluster in order to merge connected components belonging to the ground. So, we first sort the connected components by cardinality in decreasing order. Then, we assume that the biggest connected component belongs to the ground. This assumption is motivated by the fact that normally the ground is the biggest scanned horizontal object in a scene, and the great majority of points belong to it. After that, we iterate

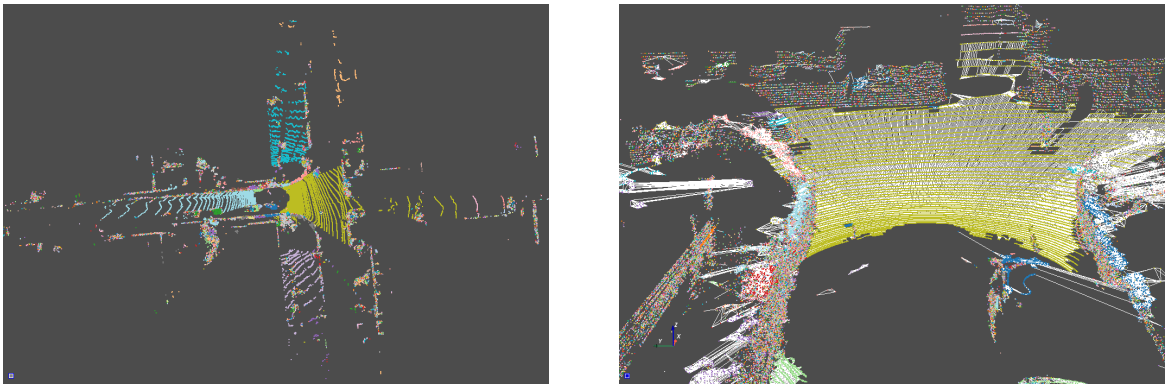


Figure 3.12 An example of the λ-flat-zones with $\lambda = 0.20m$. Different colours mean different connected components. Please remark that function w allows to easily extract horizontal surfaces, for example roads, terrain or roof of cars, while vertical objects are shattered in micro components.

over the other connected components starting from the second to the smallest and at each time we measure if the ground and the connected component are compatible. We use RANSAC algorithm to fit a plane touching both the ground and the component. If more than half of the points in the candidate connected component are inliers for the fitted plane, then we mark the component as compatible with the ground, otherwise we discard it. Only at the end of all the iterations, the components compatible with the ground are added to the ground.

3.4 Experiments on ground detection

We compare our two previously proposed methods against two state-of-the-art algorithms and a naive RANSAC method that we implemented. We test the methods on the Semantic KITTI dataset (Behley et al., 2019) that we introduced in Section 2.6.1. We include RANSAC in the analysis as a baseline benchmark. The first method we pick is CSF (Zhang et al., 2016) because it proved great adaptability to a wide range of different environments, either urban and rural. In addition, we use an FCNN method similar to the one proposed in Velas et al. (2018). The main difference with the original is the network used. Instead of employing the architecture proposed by the authors, we use a U-Net architecture Ronneberger et al. (2015), for its great versatility to different applications. To train and validate the U-Net model, we select one scan over ten in the sequences from 0 to 10 except for the sequence 08. We adopt this last sequence as a test set for all the methods. The split between training and test has been done following directives in Behley et al. (2019).

Since the dataset does not contain an explicit ground class, we derived it by aggregating multiple classes. Furthermore, to have an overview of classification errors made in the predictions, we created a total of eight categories aggregating all classes. The categories that we created are Ground, Building, Vehicles, Cycles, Person, Vegetation, Fixed-Objects, and Moving Objects. In Table 3.1 we list for each category all the classes composing it.

We use the following metrics to benchmark our experiments:

- Precision: $P = \frac{TP}{TP+FP}$
- Recall: $R = \frac{TP}{TP+FN}$

TABLE 3.1
LIST OF THE SEMANTICKITTI CLASSES BELONGING TO EACH CATEGORY THAT WE IDENTIFIED.

Ground	Building	Vehicles	Cycles	Person	Vegetation	Fixed-Objects	Moving-Objects
Road	Building	Car	Bike	Person	Vegetation	Pole	Mov-Car
Parking	Fence	Bus	Motorbike		Trunk	Traffic-sign	Mov-Cyclist
Sidewalk	Other-Structure	On-Rails	Cyclist			Other-Object	Mov-Person
Other-Ground		Truck	Motor-cyclist				Mov-Motor-cyclist
Lane-Marking		Other-Vehicle					Mov-On-rails
Terrain							Mov-Bus
							Mov-Truck
							Mov-Other-vehicle

- Accuracy: $A = \frac{TP+TN}{TP+TN+FP+FN}$
- F_1 Score : $F_1 = 2 * \frac{P*R}{P+R}$
- Intersection over Union (IoU) also called Jaccard Index: $\text{IoU}(A,B) = \frac{|A \cup B|}{|A \cap B|}$

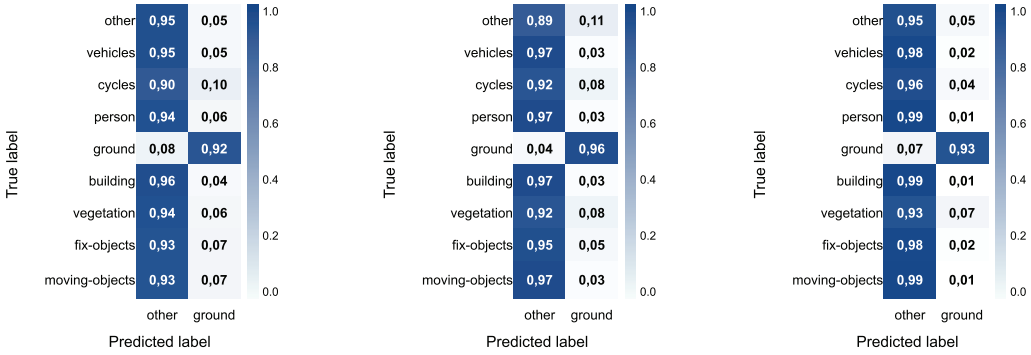
where, TP, TN, FP, FN indicate respectively the number of true positives, true negatives, false positives and false negatives, and A, B are any two sets. The sets used to compute the Jaccard Index are the set of predictions and the ground truth. In all the cases, the scores have been measured using the predictions on the 3D point clouds. In Table 3.2 we list the scores obtained by the methods.

TABLE 3.2
QUANTITATIVE RESULTS OBTAINED ON SEQUENCE 08 OF SEMANTICKITTI DATASET FOR THE GROUND DETECTION TASK

Method	F_1	Recall	Precision	Accuracy	IoU
RANSAC	0.922	0.917	0.927	0.930	0.856
CSF (Zhang et al., 2016)	0.937	**0.976**	0.900	0.940	0.881
BEV λ-FZ (This Thesis)	0.945	0.960	0.930	0.949	0.895
3D λ-FZ + RANSAC (This Thesis)	0.936	0.930	0.943	0.943	0.880
FCNN (Velas et al., 2018)	**0.951**	0.921	**0.982**	**0.957**	**0.907**

We divide the table in two parts. In the first we list the unsupervised methods and in the second we report the only supervised approach. From the results, we can see that all the methods analysed achieve great performances, and the FCNN achieve the highest score in almost all the metrics. Note that our proposed BEV λ-FZ method shows a good trade off between precision and recall, and among the unsupervised methods is the one with the highest Jaccard Index. Moreover, this method needs just few parameters to work and this makes it much easier to explain why it fails compared to FCNN. Along with these metrics, we analyse the confusion matrix to evaluate which categories are confused with the ground. For this reason, in Figure 3.13 we show the confusion matrices of all the approaches. Clearly, the returned prediction is binary, and the points predicted as not-ground are classified as *"other"*. Looking at the

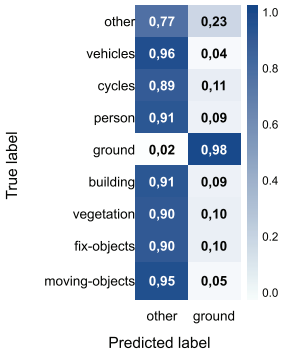
matrices, we can see that the vegetation is, in general, the class with the highest rate of points classified as ground. This can be explained by the fact that in this category there are low plants and separating them from terrain with propagation approaches is cumbersome.



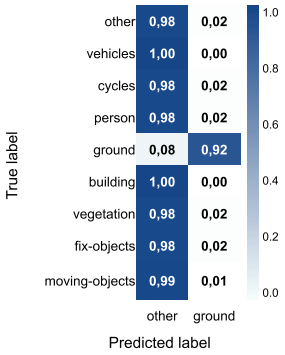
(a) RANSAC

(b) BEV λ-FZ

(c) 3D λ-FZ + RANSAC



(d) CSF



(e) FCNN

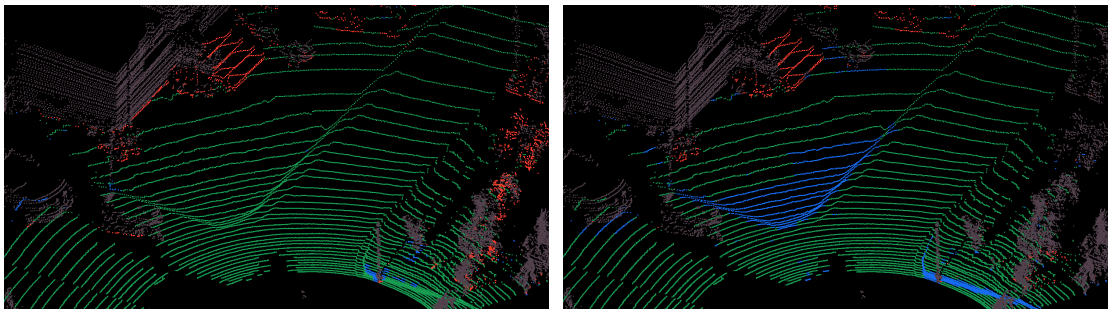
Figure 3.13 The confusion matrices help to analyse the misclassifications between Ground and other aggregated categories. (a) Naive RANSAC (b) BEV λ-quasi flat zones (c) 3D λ-quasi flat zones + RANSAC (d) CSF (e) FCNN based approach.

Let now analyse qualitatively the results and see some examples in which our proposed methods fail. To visualize the predictions in the following figures, we use the colour code hereby:

1. Green means True Positive
2. Red means False Positive
3. Blue means False Negative
4. Gray means True Negative

Let start presenting an example in which the 3D-FZ method fails. In Figure 3.14 we illustrate the predictions achieved by our two methods. Note that the approach on the 3D graph does not detect a part of the sidewalk (center image). To find what causes the problem, let us analyse Figure 3.15, which shows the λ-flat zones of the 3D graph. Note that, although the sidewalk is virtually flat, it is segmented into three parts. The over-segmentation is generated by the weight function that takes high values even when there are minor changes of orientation between neighbouring points. This happens because we want to limit the chaining effect produced when measuring only the difference along the z coordinates of two points. In

this case, the merging step misses identifying these points as ground. Moreover, in the aforementioned example, we can see that the weight function we employed does not completely prevent the chaining effect. Take for example the three steps detected as ground in the top of Figures 3.14a and 3.14b. Looking at Figure 3.15 once again, we can observe that the steps are in the same quasi-flat zone of the road, and this is clearly caused by the chaining effect. The identical thing happens in Figure 3.17, in which we illustrate the prediction carried out by our BEV approach. Here the stairs are classified as ground because the λ used is too large to detect the steps. A second example with stairs is illustrated in Figure 3.17, where we also show the predictions carried out by CSF and U-Net. In any case, it is not straightforward to prevent this kind of error. Using a smaller value of λ could circumvent it, but at the same time, we risk cutting off some other zones on the ground. Finally, the last example that we describe is the one in Figure 3.18. Here, the BEV λ-FZ does not detect a zone of the garden. The error is caused by the fact that the zone is disconnected from the other ground by a bush. Recall that our method labels as ground all the λ-flat zones whose intersection with the ground marker around the car is not empty. Hence, in this case, the garden cannot be detected as ground because it is not contained in any λ-flat zone connected with the ground marker. From the application point of view, the bush would prevent the car from reaching this ground zone. Thus, missing it is not critical for the application. Nonetheless, from our experience, this kind of missing detection happens only in isolated zones of the scene that cannot be easily reached. This is confirmed also by the high recall rate of the method, as shown in Figure 3.19.



(a) BEV λ-FZ **(b)** 3D λ-FZ + RANSAC

Figure 3.14 Example 3D approach also fails to detect as ground a part of the sidewalk. Green points are true positives, red ones are false positives, blue false negatives and grey ones are true negatives.

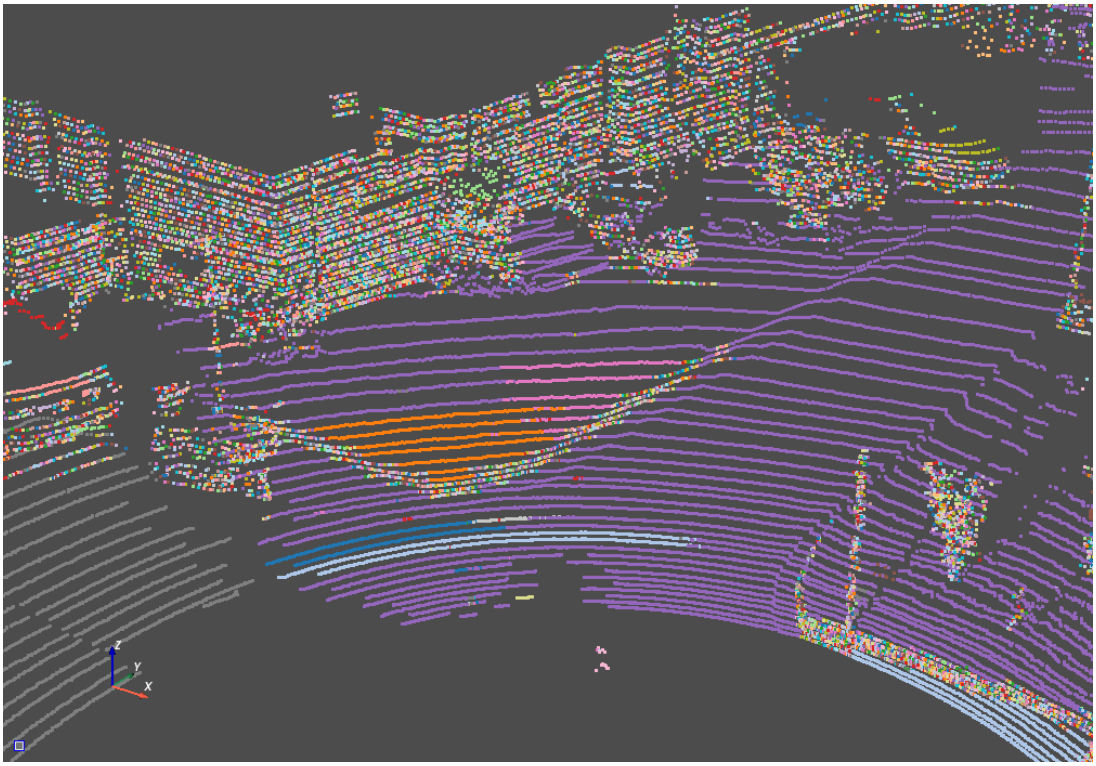
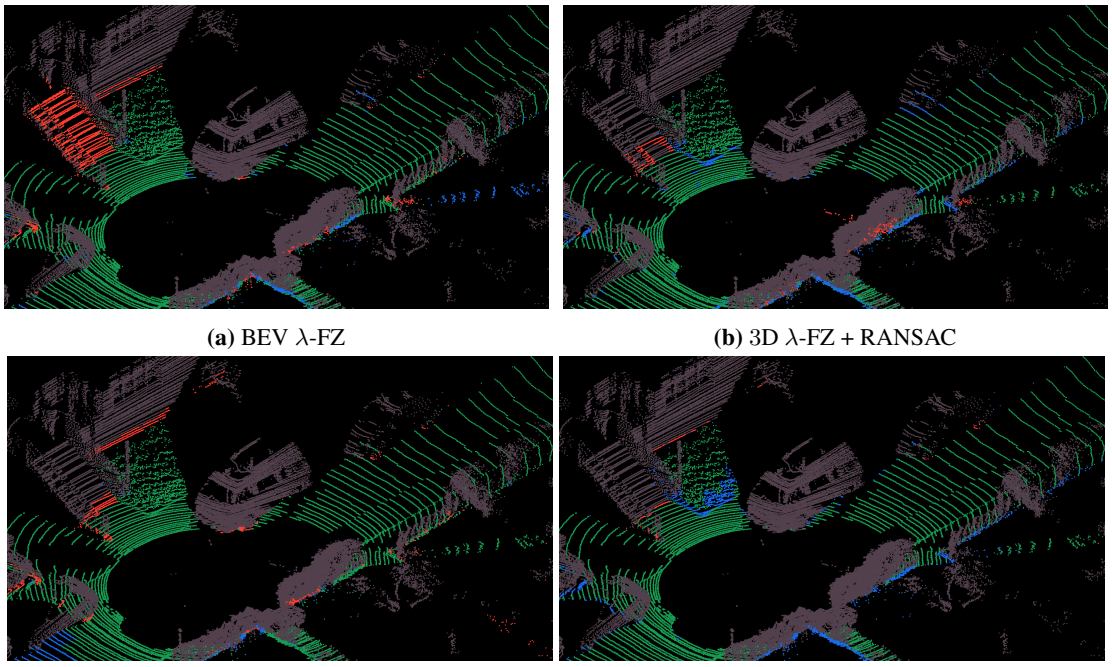


Figure 3.15 Quasi Flat Zones

Figure 3.16 λ-flat-zones obtained by the method 3D λ-FZ + RANSAC. Each colour corresponds to a different connected component.



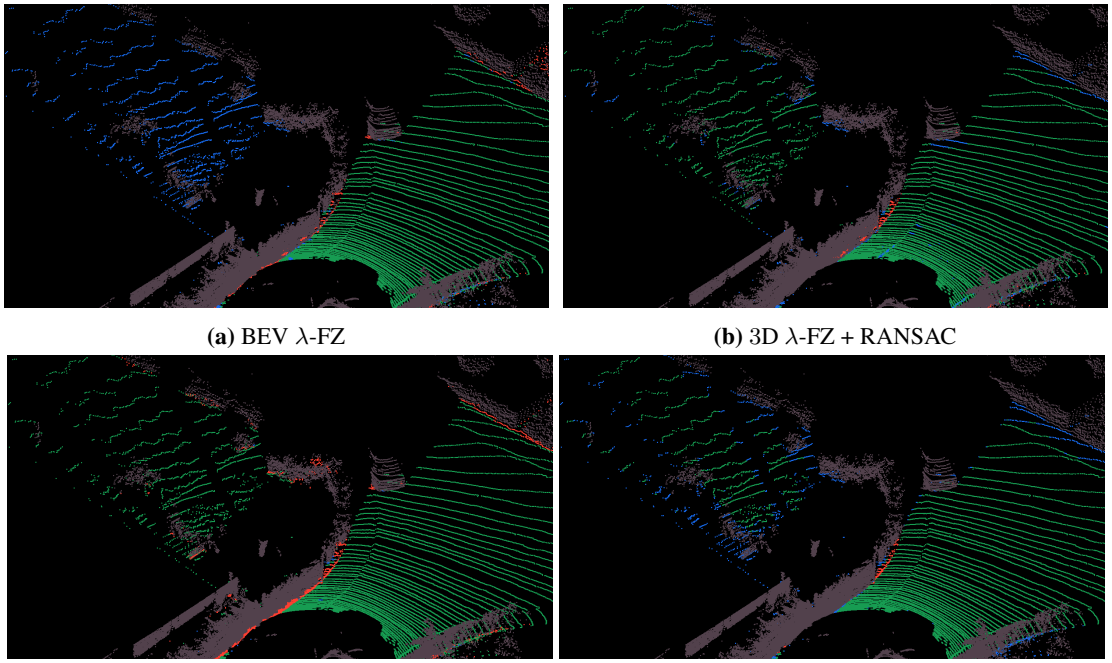
(a) BEV λ-FZ

(b) 3D λ-FZ + RANSAC

(c) CSF

(d) FCNN

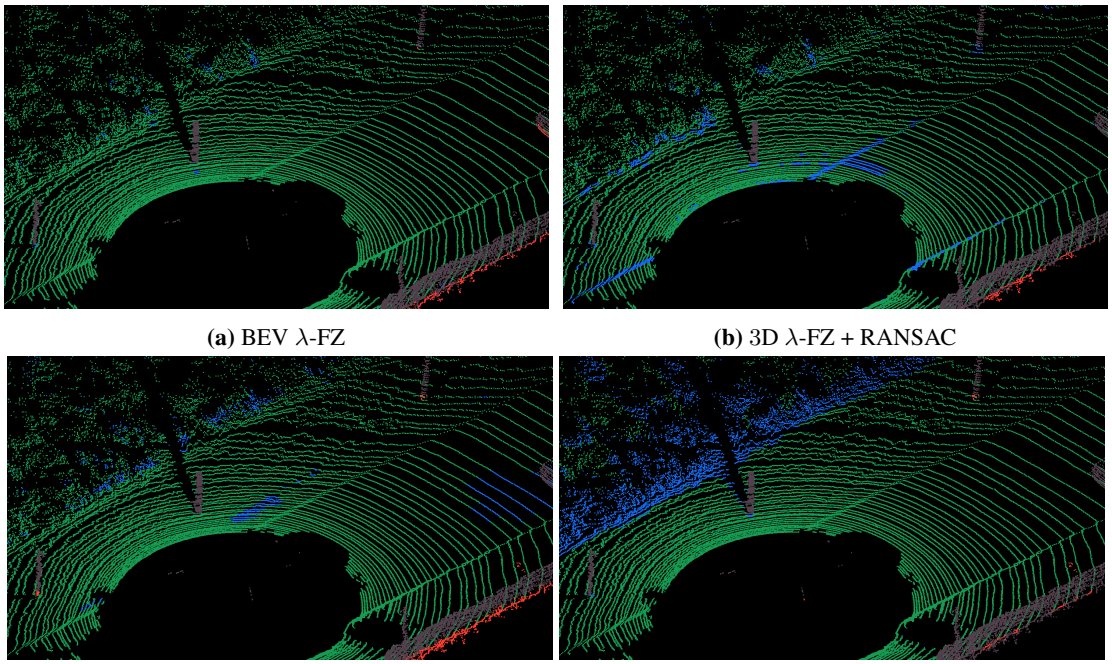
Figure 3.17 In this example BEV λ-FZ detects a stair nearby the road as ground. The λ used in this case is too big to catch the step. Green points are true positives, red ones are false positives, blue false negatives and grey ones are true negatives.



(a) BEV λ-FZ **(b)** 3D λ-FZ + RANSAC

(c) CSF **(d)** FCNN

Figure 3.18 BEV λ-FZ considers as ground the biggest flat zone in the projection image. Sometimes this method does not recover all the spots. In this example, it does not detect a piece of the garden behind a bush because it is not connected with the road. Green points are true positives, red ones are false positives, blue false negatives and grey ones are true negatives.



(a) BEV λ-FZ **(b)** 3D λ-FZ + RANSAC

(c) CSF **(d)** FCNN

Figure 3.19 Predictions obtained by the four analysed methods. Green points are true positives, red ones are false positives, blue false negatives and grey ones are true negatives. In this is an example FCNN fails to detect all the points and in particular points on the terrain.

3.5 Road detection

Modern day LIDARs are multi-layer 3D laser scanners that enable a 3D-surface reconstruction of large-scale environments. They provide precise range information while poorer semantic information as compared to colour cameras. They are thus employed in obstacle avoidance and SLAM (Simultaneous localization and Mapping) applications. The number of layers and angular steps in elevation & azimuth of the LIDAR characterizes the spatial resolution. With the recent development in the automated driving (AD) industry, the LIDAR sensor industry has gained increased attention. LIDAR scan-based point cloud datasets for AD such as KITTI usually were generated by high-resolution LIDAR (64 layers, 1000 azimuth angle positions (Fritsch et al., 2013)), referred to as a dense point cloud scans. In the recent nuScenes dataset for multi-modal object-detection a 32-Layer LIDARs scanner has been used for acquisition (Caesar et al., 2020). Another source of datasets are large-scale point clouds which achieve a high spatial resolution by aggregating multiple closely spaced point clouds, aligned using the mapping vehicle's pose information obtained using GPS-GNSS based localization and orientation obtained using inertial moment units (IMUs) (Roynard et al., 2018). Large-scale point clouds are employed in the creation of high-precision semantic map representation of environments and have been studied for different applications such as detection and segmentation of urban objects (Serna and Marcotegui, 2014). We shall focus on the scan-based point cloud datasets in our study.

Road segmentation is an essential component of autonomous driving tasks. In complement with obstacle avoidance, trajectory planning and driving policy, it is a key real-time task to extract the drivable free space as well as determine the road topology. Recent usage and proliferation of Deep Neural Networks (DNN) for various perception tasks in point clouds has opened up many interesting applications. A few applications relating to road segmentation include, binary road segmentation (Caltagirone et al., 2017) where the goal is to classify the point cloud set into road and non-road 3D points. Ground extraction (Velas et al., 2018) regards the problem of obtaining the border between the obstacle and the ground. Finally, a recent benchmark for semantic segmentation of point clouds was released with the Semantic-KITTI dataset by Behley et al. (2019). In Rangenet++ Milioto et al. (2019) evaluate the performance of U-Net & Darknet architectures for the task of semantic segmentation on point clouds. This includes the road scene classes such as pedestrians, cars, sidewalks, vegetation, road, among others.

3.5.1 Motivation & Contributions

We first observe that different LIDAR sensor configurations produce different distributions of points in the scanned 3D point cloud. The configurations refer to, LIDAR position & orientation, the vertical field-of-view (FOV), angular resolution and thus number of layers, the elevation and azimuth angles that the lasers scan through. These differences directly affect the performance of deep learning models that learn representations for different tasks, such as semantic segmentation and object detection. Low-resolution 16 layer LIDARs have been recently compared with 64 layer LIDARs (del Pino et al., 2018) to evaluate the degradation in detection accuracy, especially w.r.t distance. From Table 3.3 we observe that the HDL-64 contains 4x more points than VLP-16. This increases the computational time & memory requirements (GPU or CPU) to run the road segmentation algorithms. Thus, it is a computational challenge to process a large amount of points in real-time.

TABLE 3.3
CHARACTERISTICS OF DIFFERENT LIDARS. THE PRICES ARE REPRESENTATIVE.

LIDAR	Velodyne HDL-64	Velodyne HDL-32	Velodyne VLP-16
Azimuth	$[0°, 360°)$ step $0.18°$	$[0°, 360°)$ step $0.1° - 0.4°$	$[0°, 360°)$ step $0.2°$
Elevation	$[-24.3°, 2°]$ step 1-32 : $1/3°$ step 33-64 : $1/2°$	$[-30.67°, +10.67°]$ $1.33°$ for 32 layers	$[-15°, 15°]$ $2°$ for 16 layers
Price (as reviewed on 2019)	$\sim$ 85 k$	$\sim$ 20 k$	$\sim$ 4 k$
Effective Vertical FOV	$[+2.0°, -24.9°]$	$[-30.67°, +10.67°]$	$[+15.0°, -15.0°]$
Angular Resolution (Vertical)	$0.4°$	$1.33°$	$2.0°$
Points/Sec in Millions	~ 1.3	~ 0.7	~ 0.3
Range	120m	100m	100m
Noise	$\pm$2.0cm	$\pm$2.0cm	$\pm$3.0cm

The objective of this study is to examine the effect of reducing spatial resolution of LIDARs by subsampling a 64-scanning layers LIDAR on the task of road segmentation. This is done to simulate the evaluation of low-resolution scanners for the task of road segmentation without requiring any pre-existing datasets on low-resolution scanners. The key contribution and goal of our experiment are: First, to evaluate the impact of the point cloud's spatial resolution on the quality of the road segmentation task. Secondly, determine the effect of subsampling on different point cloud representations, namely on the Bird Eye View (BEV) and Spherical View (SV), for the task of road segmentation. For BEV representation we use the existing LoDNN (LIDAR Only Deep Neural Networks) architecture (Caltagirone et al., 2017), while for SV we use a simple U-net architecture. Figure 3.20, provides a global overview of the methodology used. Finally, we propose to use surface point normals as complementary features to the ones already used in current state-of-the-art research. Results are reported on the KITTI road segmentation benchmark (Fritsch et al., 2013), and the newly introduced Semantic KITTI dataset by Behley et al. (2019).

3.5.2 Related Work

LoDNN (Caltagirone et al., 2017) is a Fully Convolutional Network (FCN) based binary segmentation architecture, with encoder containing sub-sampling layers, and decoder with up-sampling layers. The architecture is composed of a core context module that performs multi-scale feature aggregation using dilated convolutions. In the class of non-deep learning methods, Chen et al. (2017) build a depth image in spherical coordinates, with each pixel indexed by a set of fixed azimuth values (ϕ) and horizontal polar angles (θ), with intensity equal to the radial distances (r). Authors assume for a given scanner layer (a given ϕ) all points belonging to the ground surface shall have the same distance from the sensor along the x axis.

Lyu et al. (2019) propose a FCN based encoder-decoder architecture with a branched convolutional block called the ChipNet block. It contains filters with ($1 \times 1 \times 64 \times 64$, $3 \times 3 \times 64 \times 64$, $3 \times 3 \times 64 \times 64$) convolutional kernels in parallel. They evaluate the performance of road segmentation on Ford dataset and KITTI benchmark on a Field-Programmable Gate Array (FPGA) platform. The work closest to our study is by del Pino et al. (2018), where they compare a high-resolution 64-layer LIDAR with a low-resolution

system, 16-layer LIDAR, for the task of vehicle detection. They obtain the low resolution 16-layer scans by sub-sampling the 64-layer scans. The results demonstrate that their DNN architecture on low resolution outperforms their geometric baseline approach. They also report similar tracking performance w.r.t their high-resolution HDL-64 sensor at close range.

Additionally, Jaritz et al. (2018) study joint sparse-to-dense depth map completion and semantic segmentation using NASNet architectures. They work with varying densities of points reprojected into the Front View (FV) image, that is the image domain of the camera sensor. Authors achieve an efficient interpolation of depth to the complete FOV using features extracted using early and late fusion from the RGB-image stream.

3.6 Methodology

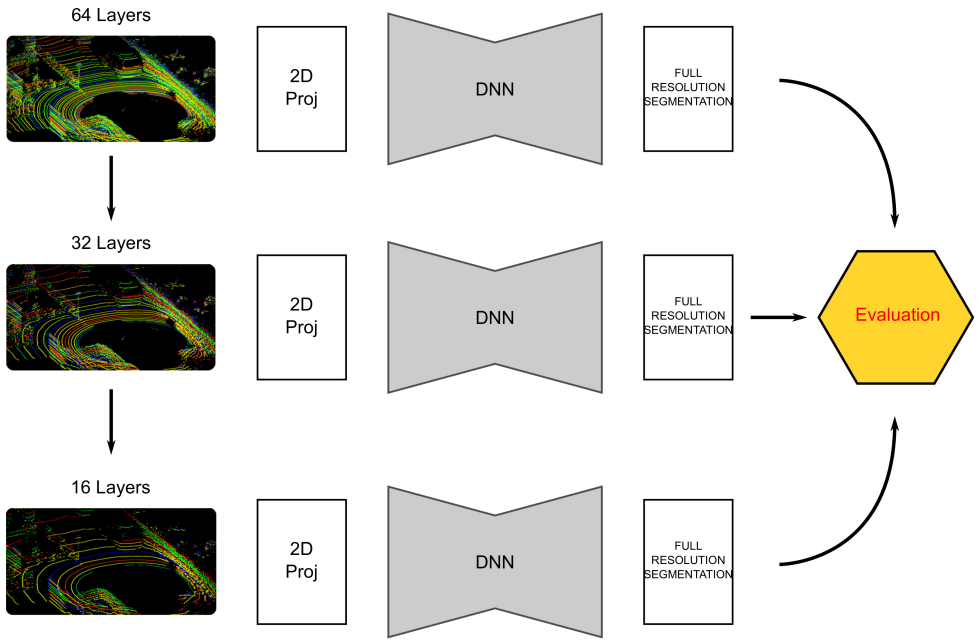


Figure 3.20 Overall methodology to evaluate the performance of road segmentation across different resolutions. See Figures 3.22 for more details on the architectures used.

In our study, each LIDAR scan is projected on an image. The two projections we consider are the Bird's-eye View (BEV) and Spherical View (SV) introduced in section 2.4.3. Concerning BEV images, we define a grid of 20 meters wide, $y \in [-10, 10]$, and 40 meters long, $x \in [6, 46]$, as in Caltagirone et al. (2017). This grid is divided into cells of size 0.10×0.10 meters. Within each cell, we evaluate six features: number of points, mean reflectivity, and mean, standard deviation, minimum, and maximum elevation. Each point cloud is thus projected and encoded in a tensor of $400 \times 200 \times 6$, where $400, 200$ are the BEV image height and width. We refer to the set of these six features as *BEV Classical Features* (see Figure 3.25).

Regarding the SV grid, the size of the cells is generally chosen according to the Field Of View (FOV) of the scanner. For instance, Behley et al. (2019) project point clouds to 64×2048 pixel images by varying the azimuth angle (θ) and vertical angle (φ) into two evenly discretized segments. In our case, we use a slightly different version of the SV. Instead of evenly dividing the vertical angle axis and associating a

point to a cell according to its vertical and azimuth angles, we retrieve for each point the scanner layer that acquired it, and we assign the cell according to the position of the layer in the laser stack and the value of the azimuth angle. Basically, for a scanner with 64 layers, we assign the point $\mathbf{x}$ to a cell in row i if $\mathbf{x}$ has been acquired by the i-th layer starting from the top. We decided to use this approach because in the scanner the layers are not uniformly spaced and using standard SV projection causes different points to collide on the same cell and strips of cells are empty in the final image as illustrated in Figure 3.21. A similar approach to ours has also been used in Triess et al. (2020). In subsection 3.6.2, we describe how we associate each point to the layer that captured it. However, we underline that our method relies on the way the points are ordered in the point cloud array.

Finally, following Velas et al. (2018), in each grid cell we compute the minimum elevation, mean reflectivity and minimum radial distance from the scanner. This information is encoded in a three-channel image. Since these three features are already used in the state of the art for the ground segmentation task, in SV we refer to *SV Classical Features*, as the set of SV images composed by minimum elevation, mean reflectivity and minimum radial distance.

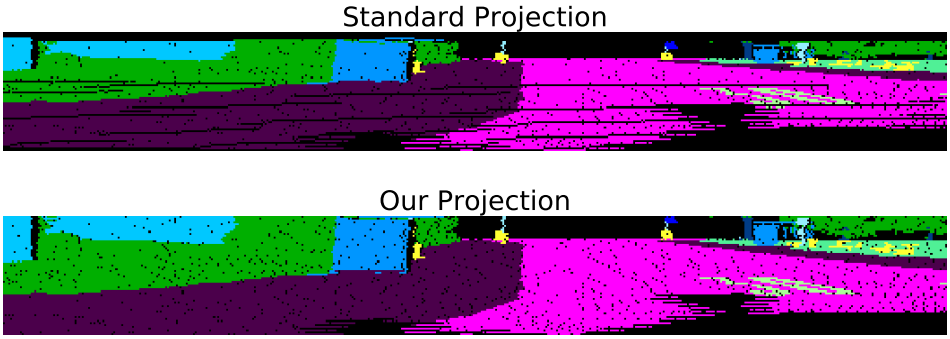


Figure 3.21 SV projection: two cropped images showing the difference between the standard projection, and our projection.

Once extracted these feature maps we use them as input to train DNNs for binary segmentation. We trained LoDNN model for the case of BEV projection, and U-Net model in the case of SV projection.

3.6.1 DNN models

The LoDNN architecture from Caltagirone et al. (2017) is a FCN designed for semantic segmentation, it has an input layer that takes as input the BEV images, an encoder, a context module that performs multi-scale feature aggregation using dilated convolutions and a decoder which returns confidence map for the road. Instead of Max Unpooling layer[1] specified in Caltagirone et al. (2017), we use a deconvolution layer (Zeiler et al., 2010). Other than this modification, we have followed the author's implementation of the LoDNN. The architecture is reported in Figure 3.22a.

The U-Net architecture (Ronneberger et al., 2015), is a FCN designed for semantic segmentation. In our implementation of U-Net, the architecture is made of three steps of downsampling and three

[1] In the context of DNN, a *layer* is a general term that applies to a collection of 'nodes' operating together at a specific depth within a neural network. In the context of LIDAR scanners, the *number of scanning layer* refers to the number of laser beams installed in the sensor.

steps of upsampling. During the downsampling part, a 1×2 Max Pooling layer is used to reduce the features spatial size. In order to compare the different cases ($64/32/16$ scanning layers) among them, the 64 scanning layers ground truth is used for all the cases. For this purpose, an additional upsampling layer of size 2×1 is required at the end of the 32-based architecture and two upsampling layers at the end of the 16. In fact, the size of SV images for the 32 scanning layer is 32×2048. Thus, without an additional upsampling layer, we would obtain an output image whose size is 32×2048. Similarly, for the 16 scanning layer, we add two upsampling layers of size 2×1 to go from 16×2048 to 64×2048 pixels output images. Figures 3.22b, 3.22c & 3.22d illustrate the three architectures used.

In both cases, a confidence map is generated by each model. Each pixel value specifies the probability of whether the corresponding grid cell of the region belongs to the road class. The final segmentation is obtained by thresholding at 0.5 the confidence map.

3.6.2 Sub-sampling point clouds to simulate low resolution

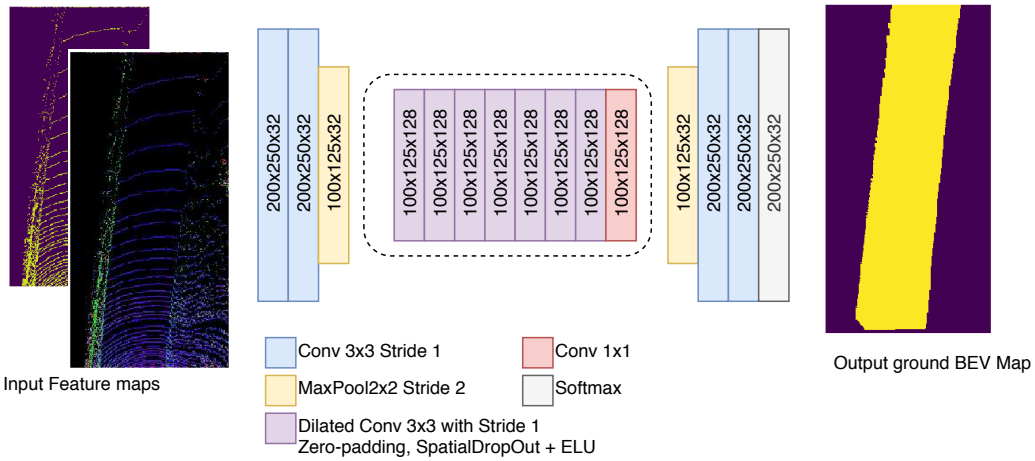
As far as we know, currently there are no dataset available containing scans of the same environment taken simultaneously with scanners at different resolutions. Thus, we simulated the 32 and 16 layer scans removing layers from 64 scans. To achieve this, we first need to associate each point within each 64 scan to the scanning layer it was captured from. We exploit the special ordering in which point clouds have been stored within the KITTI datasets. Figure 3.23 shows the azimuth and polar angles of the sequence of points of a single scan. We observe that the azimuth curve contains 64 cycles over 2π degrees, while the polar curve globally increases. Thus, a layer corresponds to a round of 2π degrees in the vector of azimuths. Scanning layers are stored one after another starting from the uppermost layer to the lowermost one. As we step through sequentially 2π in the azimuth angle (two zero crossings), we label each point to be within the same layer. Once retrieved the layers, we can obtain a 32 scan removing one layer out of two from the 64 scan, and obtain a 16 scan removing three layers out of four. The size of SV input images changes when we remove layers. We move from 64×2048 pixels for a 64 layer scanner to 32×2048 pixels for the 32 layer and to 16×2048 pixels for the 16 layer.

3.6.3 Surface normal extraction

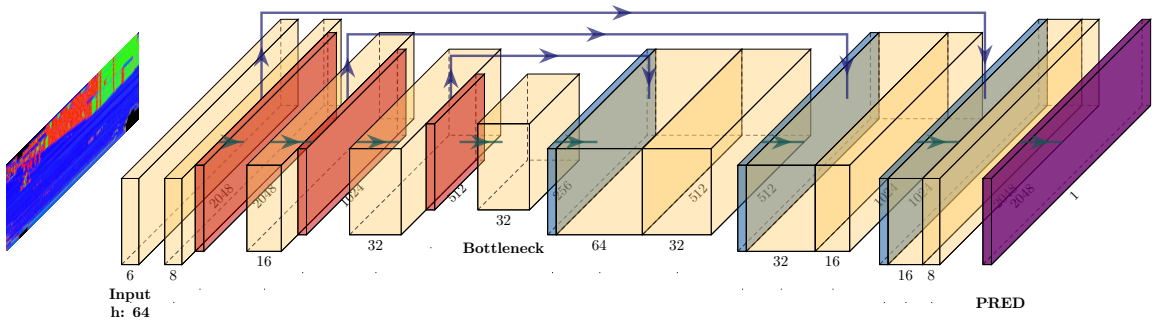
Along with features used in the state of the art, we estimate surface normals from the image containing radial distances in the SV. Our method is inspired by the work of Nakagawa et al. (2015) where the authors estimate surface normals using depth image gradients. Let $p = (\varphi, \theta)$ a couple of angles in the spherical grid and let R be the image containing the radial distances. We can associate to p a point P in the 3D space using the map $\Psi : [0, 2\pi) \times [0, \pi] \to \mathbb{R}^3$ defined as

$$\Psi(\varphi,\theta) = \begin{cases} x = R(\varphi,\theta)\cos(\varphi)\sin(\theta), \\ y = R(\varphi,\theta)\sin(\varphi)\sin(\theta), \\ z = R(\varphi,\theta)\cos(\theta). \end{cases} \tag{3.1}$$

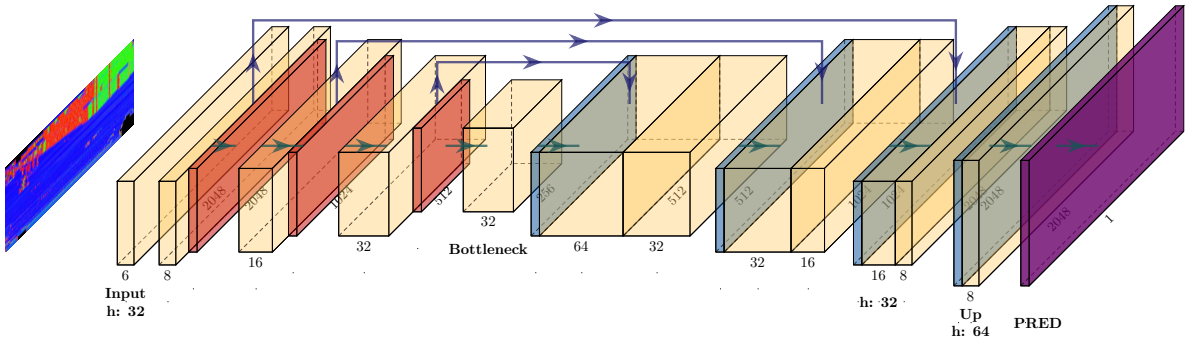
Now, let $p_\varphi = (\varphi + \Delta\varphi, \theta)$ and $p_\theta = (\varphi, \theta + \Delta\theta)$ respectively the vertical and the horizontal neighbouring cells. They have two corresponding points P_φ and P_θ in the 3D space, as well. Since P, P_φ and P_θ compose a local 3D plane, we can estimate the normal vector $\frac{\partial\Psi}{\partial\varphi} \times \frac{\partial\Psi}{\partial\varphi}$ at P using the two vectors v_φ, v_θ



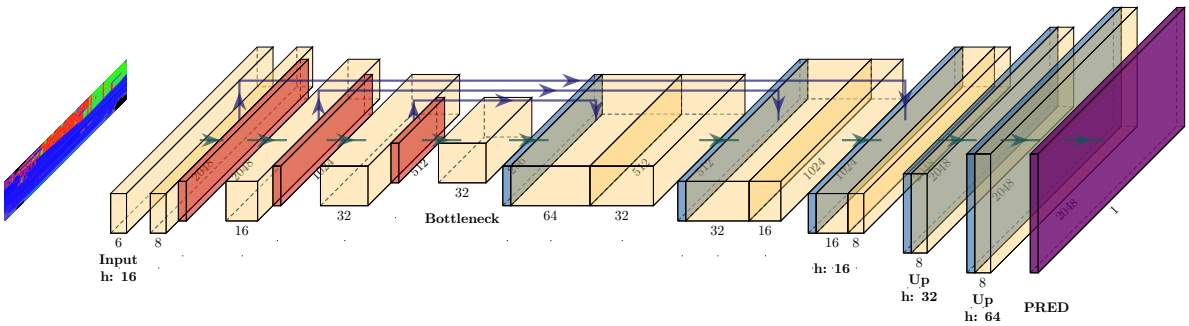
(a) LoDNN Architecture by authors Caltagirone et al. (2017) in our experiments on BEV.



(b) U-Net architecture used for the 64 layer scanner.



(c) U-Net architecture used for the 32 layer scanner.



(d) U-Net architecture used for the 16 layer scanner.

Figure 3.22 (a) LoDNN Architecture used on BEV images. (b-d) U-Net Architectures used on SV images.

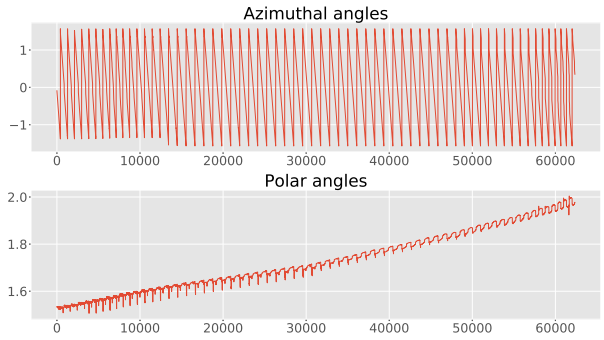


Figure 3.23 Plot containing the azimuths and vertical angles for a single point cloud.

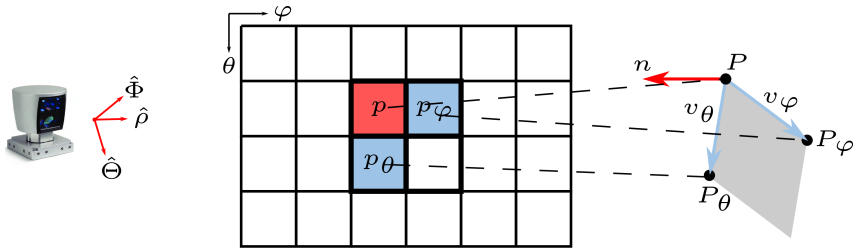


Figure 3.24 Relationship between adjacent pixels in the radial distance image $\mathcal{I}$ and adjacent points in the 3D space. Pixels p, p_φ and p_θ are associated to 3D points P, P_φ and P_θ. Since P, P_φ and P_θ compose a local plane, we compute their 3D gradients as tangent vectors v_φ, v_θ from a radial distance value at p, p_φ and p_θ.

spanning the local surface containing P, P_φ and P_θ, as in Figure 3.24. We compute v_φ using the values of the radial distance image $\mathcal{I}$ at pixels p, p_φ as

$$v_\varphi(\varphi,\theta) = \begin{pmatrix} d_\varphi\mathcal{I}(\varphi,\theta)\cos(\varphi)\sin(\theta) + \mathcal{I}(\varphi,\theta)\cos(\varphi)\cos(\theta) \\ d_\varphi\mathcal{I}(\varphi,\theta)\sin(\varphi)\sin(\theta) + \mathcal{I}(\varphi,\theta)\sin(\varphi)\cos(\theta) \\ d_\varphi\mathcal{I}(\varphi,\theta)\cos(\theta) - \mathcal{I}(\varphi,\theta)\sin(\theta) \end{pmatrix}$$

where

$$\begin{aligned} d_\varphi\mathcal{I}(\varphi,\theta) &= \frac{\mathcal{I}(\varphi+\Delta\varphi,\theta) - \mathcal{I}(\varphi,\theta)}{\Delta\varphi} \\ &= \frac{\mathcal{I}(p_\varphi) - \mathcal{I}(p)}{\Delta\varphi} \approx \frac{\partial\mathcal{I}}{\partial\varphi}(\varphi,\theta). \end{aligned} \tag{3.2}$$

Similarly v_θ is obtained using values at p and p_θ as:

$$v_\theta(\varphi,\theta) = \begin{pmatrix} d_\theta\mathcal{I}(\varphi,\theta)\cos(\varphi)\sin(\theta) - \mathcal{I}(\varphi,\theta)\sin(\varphi)\sin(\theta) \\ d_\theta\mathcal{I}(\varphi,\theta)\sin(\varphi)\sin(\theta) + \mathcal{I}(\varphi,\theta)\cos(\varphi)\sin(\theta) \\ d_\theta\mathcal{I}(\varphi,\theta)\cos(\theta) \end{pmatrix}$$

where

$$\begin{aligned} d_\theta\mathcal{I}(\varphi,\theta) &= \frac{\mathcal{I}(\varphi,\theta+\Delta\theta) - \mathcal{I}(\varphi,\theta)}{\Delta\theta} \\ &= \frac{\mathcal{I}(p_\theta) - \mathcal{I}(p)}{\Delta\theta} \approx \frac{\partial\mathcal{I}}{\partial\theta}(\varphi,\theta). \end{aligned} \tag{3.3}$$

The approximated normal vector is $n = v_\varphi \times v_\theta$. Once the surface point normals are estimated in the SV, we get them back to 3D-cartesian coordinates, and subsequently project them onto the BEV. This adds three supplementary channels to the input images. Figure 3.26 shows the results obtained on SV image with 64 layer. For each pixel we mapped the coordinates (x, y, z) of the estimated normals to the RGB color map, so $x \to R$, $y \to G$ and $z \to B$. Please remark that a FCN can not extract this kind of features through convolutional layers starting from *SV classic features*. In fact, to extract this kind of information using convolution the FCN should be aware of the position of the pixel inside of the image, *i.e.* know the angles (φ, θ), but this would break the translational symmetry of convolutions. This enforces prior geometrical information to be encoded in the features maps that are the input of the DNN. Finally, we also remark that the normal feature maps are computationally efficient since it is a purely local operation in the spherical coordinates.

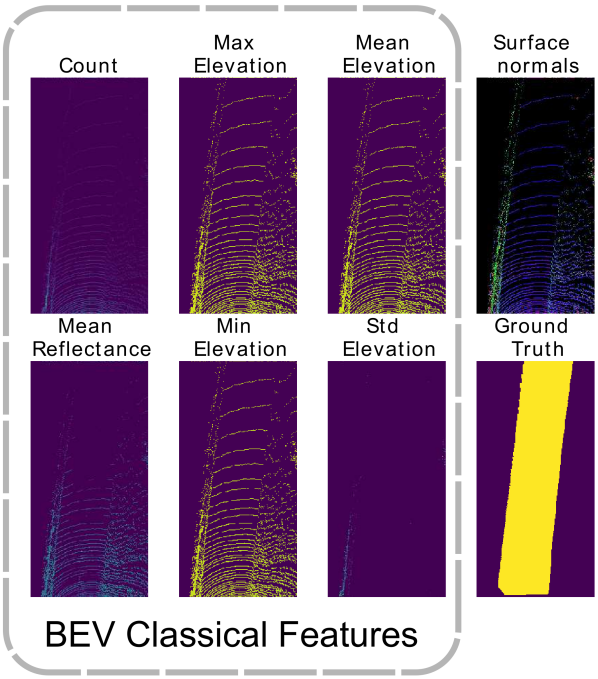


Figure 3.25 An example of features projected on the BEV in case of a 64 layers scanner. Surface normals are computed on SV and projected to BEV.

3.7 Experiments & Analysis

The binary road segmentation task is evaluated on two datasets : 1. the KITTI road segmentation Fritsch et al. (2013), 2. Semantic-KITTI dataset Behley et al. (2019). The input information to the DNNs comes purely from point clouds and no camera image information was used. The BEV and SV representations were used over the point clouds from KITTI road-dataset, while only the SV representation over Semantic-KITTI. The BEV ground truth information for semantic-KITTI did not currently exist at the time of writing this thesis, and thus no evaluation was performed. The projection of the 3D labels to BEV image in semantic-KITTI produced sparsely labelled BEV images and not a dense ground truth as compared to the BEV ground truth in Figure 3.25. The SV ground truth images have been generated by projecting 3D labels to 2D pixels. We consider a pixel as road if at least one road 3D point is projected on the pixel.

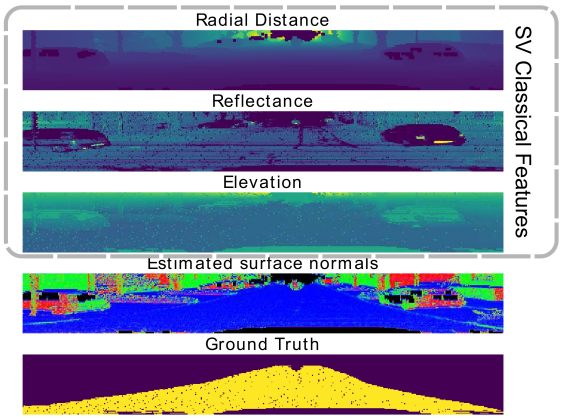


Figure 3.26 A crop example of features projected on the SV in case of a 64 layers scanner. Surface normals are estimated from radial distance image. The last image below is the ground truth for this case.

Training : Adam optimizer with initial learning rate of 0.0001 is used to train the models. The models were trained using an Early-Stopping. We used the Focal loss with gamma factor of $\gamma = 2$ Lin et al. (2017), thus the resulting loss is $L(p_t) = -(1-p_t)^{\gamma}\log(p_t)$, where

$$p_t = \begin{cases} p & \text{if } y = 1, \\ 1-p & \text{otherwise.} \end{cases}$$

The focal loss was useful in the KITTI Road segmentation benchmark. The road class was measured to be around 35% of the train and validation set, in the BEV, while around 5% for the SV. This drastic drop in the road class in SV is due to the restriction of the labels to the camera FOV. While for Semantic-KITTI we observed lesser level of imbalance between road and background classes.

Metrics : Along with Precision, Recall and F_1 scores defined in Section 3.4, we also measured the Average Precision that is defined as

$$AP = \sum_n P_n(R_n - R_{n-1}) \tag{3.4}$$

where, $P = \frac{TP}{TP+FP}$ $R = \frac{TP}{TP+FN}$ and P_n, R_n are precision and recall at n-th threshold. In all the cases, the scores have been measured on the projected images, and we report them in Table 3.4.

Evaluating different resolutions: When subsampling point clouds, the input SV image size changes accordingly. For example, after the first subsampling, the input SV image now has a size of 32×2048. In order to get a fair comparison, the evaluation of all results is made at the original full resolution at 64×2048. In such a case, the number of layers in the U-Net architectures has been increased to up-sample the output segmentation map to the full resolution. This leads to 3 different architectures for 16, 32 and 64 layers, see Figures 3.22b, 3.22c & 3.22d. Three different models were trained on the different SV images. In the Semantic KITTI dataset, the evaluation has been done over the road class. The BEV image on the other hand remains the same size with subsampling. Though, subsampling in BEV introduces more empty cells as certain layers disappear.

3.7.1 KITTI road estimation benchmark

The KITTI road segmentation dataset consists of three categories: urban unmarked (UU), urban marked (UM), and urban multiple marked lanes (UMM). Since the test dataset's ground truth is not publicly available, 289 training samples from the dataset are split into training, validation and test sets for the experiments. Validation and test sets have 30 samples each and the remaining 229 samples are taken as training set.

Ground truth annotations are represented only within the camera perspective for the training set. We use the ground truth annotations provided by authors Caltagirone et al. (2017) in our experiments. The BEV ground truth was generated over the xy-grid within $[-10, 10] \times [6, 46]$ with squares of size 0.10×0.10 meters.

Figures 3.27, 3.28 illustrate the Precision-Recall (PR) curves obtained on BEV images and SV images. The performance metrics for the different resolutions (64/32/16 scanning layers) of the scanners are reported, for both the classical and classical-with-normal features. At full resolution, the classic features obtain state-of-the-art scores as reported by authors Caltagirone et al. (2017). In Table 3.4, we observe that with subsampling and reduction in the number of layers, there is a degradation in the AP along with metrics. With the addition of the normal features, we observe an improvement in AP across all resolutions/number of layers.

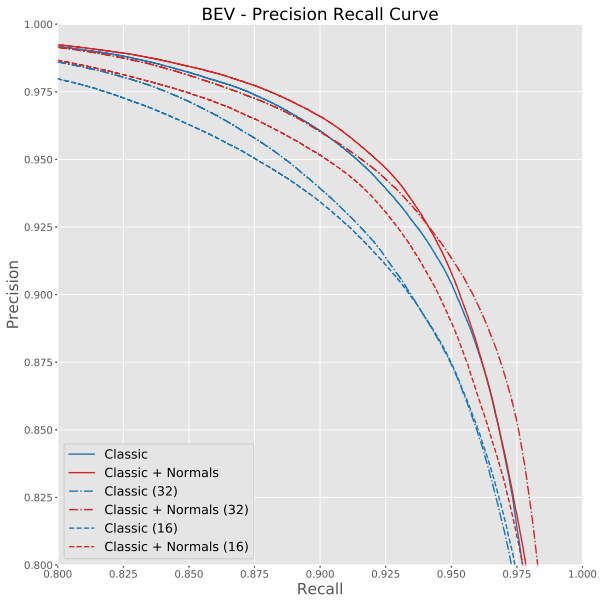


Figure 3.27 KITTI Road Segmentation with BEV images: Precision-Recall Curve for various features with and without sub-sampling.

3.7.2 Semantic-KITTI

The Semantic-KITTI dataset is a recent dataset that provides a pointwise label across the different sequences from KITTI Odometry dataset, for various road scene objects, road, vegetation, sidewalk and other classes. The dataset was split into train and test datasets, considering only the road class. To reduce the size of the dataset, and temporal correlation between frames, we sampled one in every ten frames over

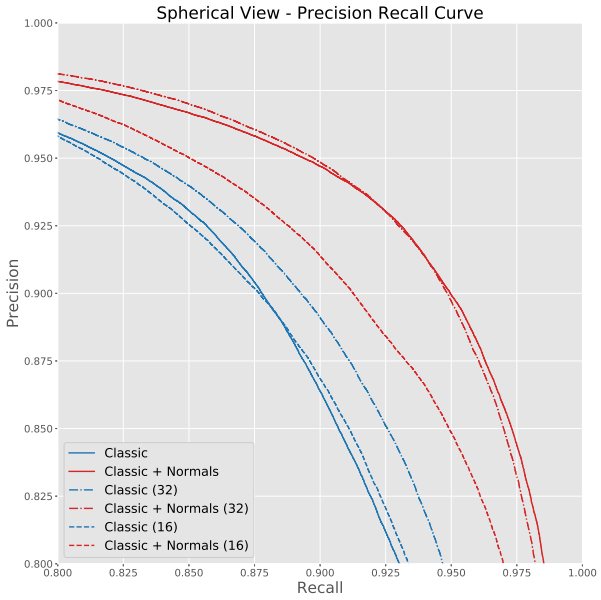


Figure 3.28 KITTI Road Segmentation with SV images: Precision-Recall Curve for various features with and without sub-sampling.

TABLE 3.4
RESULTS OBTAINED ON THE TEST SET OF THE KITTI ROAD SEGMENTATION DATASET IN THE BEV AND SV.

KITTI Road-Seg, BEV	**AP**	**F_1**	**Recall**	**Precision**
Classical (64) (Caltagirone et al., 2017)	0.981	0.932	0.944	0.920
Classical + Normals (64)	0.983	0.935	0.945	0.926
Classical (32)	0.979	0.920	0.926	0.914
Classical + Normals (32)	0.984	0.934	0.937	0.930
Classical (16)	0.978	0.918	0.920	0.915
Classical + Normals (16)	0.981	0.927	0.936	0.919
KITTI Road-Seg, SV	**AP**	**F_1**	**Recall**	**Precision**
Classical (64)	0.960	0.889	0.914	0.889
Classical + Normals (64)	0.981	0.927	0.926	0.929
Classical (32)	0.965	0.896	0.915	0.878
Classical + Normals (32)	0.981	0.927	0.928	0.927
Classical (16)	0.960	0.888	0.900	0.875
Classical + Normals (16)	0.974	0.906	0.914	0.899

the sequences 01-10 excluding the sequence 08, over which we reported our performance scores. The split between training and test has been done following directives in Behley et al. (2019).

With the decrease in vertical angular resolution by subsampling the original 64 layers SV image, we observe a minor but definite drop in the binary road segmentation performance (in various metrics) for sparse point clouds with 32 and 16 scanning layers. This decrease is visible both in Table 3.5 but also in the Precision-Recall curve in Fig. 3.29. With the addition of our normal features to the classical features we do observe a clear improvement in performance across all resolutions (16, 32 and 64 scanning layers). Geometrical normal features channel as demonstrated in Fig. 3.26 show their high correlation w.r.t the

road class region in the ground-truth. Road and ground regions represent surfaces which are low elevation flat surfaces with normal's homogeneously pointing in the same directions.

TABLE 3.5
RESULTS OBTAINED ON THE TEST SET OF THE SEMANTIC-KITTI DATASET IN THE SV.

Semantic KITTI-SV	AP	F_1	Rec	Prec
Classical (64)	0.969	0.907	0.900	0.914
Classical + Normals (64)	0.981	0.927	0.927	0.927
Classical (32)	0.958	0.897	0.902	0.892
Classical + Normals (32)	0.962	0.906	0.906	0.906
Classical (16)	0.944	0.880	0.879	0.882
Classical + Normals (16)	0.948	0.889	0.894	0.883

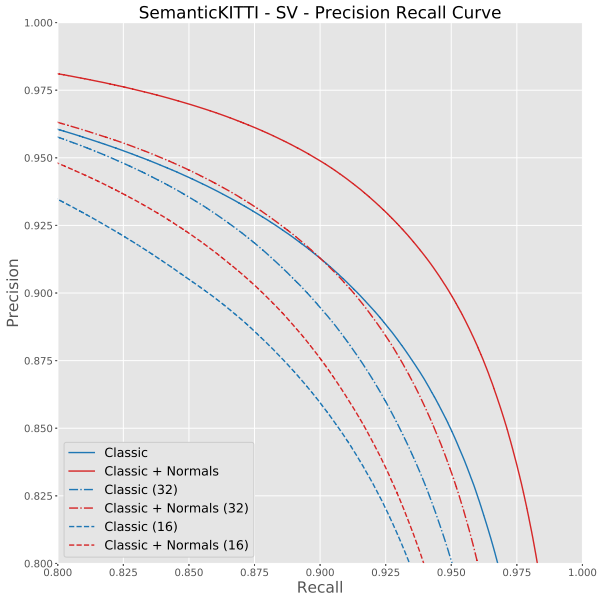


Figure 3.29 SemanticKITTI with SV images: Precision-Recall Curve for various features with and without sub-sampling.

3.8 Conclusions

In the first part of this chapter, we have shown two novel λ-flat-zones based algorithms for Ground Detection. The first works projecting points on a BEV, while the second builds a graph connecting points in the 3D space and using normals to assign weights. Moreover, we have compared our methods with two state-of-the-art methods (CSF and FCNN) and a naive RANSAC on the SemanticKITTI dataset. Results show that our methods are comparable with other two state-of-the-art algorithms, even though FCNN is more precise. In our opinion, the few parameters used and the greater intelligibility in case of error compared to FCNN make our algorithms good candidates for potential applications.

Concerning the second part, in this study we evaluate the effect of subsampled LIDAR point clouds on the performance of prediction for segmentation of road class. This is to simulate the evaluation of

low-resolution scanners for the task of road segmentation. As expected, reducing the point cloud resolution reduces the segmentation performances. Given the intrinsic horizontal nature of the road, we propose to use estimated surface normals. These features cannot be obtained by a FCN using *Classical Features* as input. We demonstrate that the use of normal features increases the performance across all resolutions and mitigates the deterioration in performance of road detection due to subsampling in both BEV and SV. Features based on normal information encode planarity of roads and are robust to subsampling.

4 Minimum Spanning Tree for data streams

Resumé

L'arbre couvrant de poids minimal (ACM) est l'une des structures de données les plus populaires utilisées pour extraire des informations hiérarchiques des images. Ce chapitre traite de la construction de ACM en streaming pour les images. Tout d'abord, nous nous concentrons sur le problème du calcul d'un ACM de l'union de deux graphes avec une intersection non vide, puis nous montrons comment notre solution peut être appliquée au streaming d'images. La solution proposée repose sur la décomposition des données en deux parties. L'une est enregistrable et ne changera pas dans le futur. Elle peut être stockée ou utilisée pour des traitements ultérieurs. L'autre, instable, nécessite des informations supplémentaires avant de devenir stable. L'exactitude de l'algorithme proposé a été prouvée et confirmée dans le cas de la segmentation morphologique d'images de télédétection.

4.1 Introduction

The minimum spanning tree (MST) is one of the most popular data structure used to extract hierarchical information from images. This chapter addresses MST construction in streaming for images. First, we focus on the problem of computing a MST of the union of two graphs with a non-empty intersection. Then we show how our solution can be applied to streaming images. The proposed solution relies on the decomposition of the data in two parts. One *stable* that does not change in the future. This can be stocked or used for further treatments. The other *unstable* needs further information before becoming stable. The correctness of proposed algorithm has been proven and confirmed in the case of morphological segmentation of remote sensing images.

4.2 Streaming context

Let us start introducing the context and the mathematical elements we are going to deal with. Given an image I, we can associate to it a 4-connected weighted graph $\mathcal{G} = (V, E, W)$, with nodes $V = \{p_1, \ldots, p_n\}$

being all image pixels and edges $E = \{e_1, \ldots, e_m\}$ between neighbouring pixels, and weighting edges by color/intensity differences. For example, if we assume I to be a greyscale image we can define $W(e) = |I(p) - I(q)|$, where $e = (p, q)$ and $I(p)$ is the image intensity at pixel p.

Briefly, a MST of a graph $\mathcal{G}$ is a subgraph $\mathcal{T}$ that spans all the nodes of $\mathcal{G}$. It contains no cycle and such that the sum of the weights of its edges is minimal. Prim and Kruskal are the most well-known algorithms (Wu and Chao, 2004) to extract a MST from a graph $\mathcal{G}$. The Kruskal method runs in $O(|E|\log(|V|))$ time in the worst case, where $|\cdot|$ is the function that measures the cardinality of a set. Whilst the implementation of Prim that uses Fibonacci heap runs in $O(|E| + |V|\log(|V|))$. In the case of graph associated to images, (Bao et al., 2014) developed a Prim-based algorithm. The authors propose a method to compute a MST for a 8-bit depth that runs in $O(|V|)$ time. Furthermore, Prim and Kruskal algorithms have been applied also for clustering identification problems. See (Kim, 2009) for further details. However, in this context the biggest challenge is the processing of huge data volumes. To solve this problem, (Olman et al., 2009) proposed a parallel Prim-based algorithm to construct a MST. Similarly to the methods we present later, their method split data in disjoint chunks. Extract a MST for each chunk and a MST for bipartite graphs between two neighbouring chunks. Then merge all the MSTs together and finally extract a MST of the complete data set from this union. Unlike the previous case, our methods split the data in non-disjoint chunks. This give us as main advantage, the footprint of memory is reduced. The graphs are decomposed between a *stable* and an *unstable* part. The first belongs to the final MST. So, it can be stored, and it is not modified any more. From the second we have to extract the remaining of the final MST. In this way, a streaming application can use data on the fly based on stable regions and the footprint of memory is reduced. Therefore, bigger data sizes can be processed.

In this chapter, we focus our attention on streaming applications. Let us introduce the streaming image problem. Consider the simple case of an image I decomposed in two blocks B_1, B_2 and sent one after another. Let B_1 be the first block arriving. Suppose that we compute its MST. The question is, how to compute the MST for the whole image, $I = B_1 \cup B_2$, when B_2 arrives and exploit the information extracted from B_1? Before tackling this problem, basic definitions from graph theory and notation used in this chapter will be given.

Definition 4.1 (Graph Union)**.** Let $\mathcal{G}_1 = (V_1, E_1, W_1)$ and $\mathcal{G}_2 = (V_2, E_2, W_2)$ two weighted undirected graphs, such that

$$W_1|_{E_1 \cap E_2} \equiv W_2|_{E_1 \cap E_2},$$

where $W|_E$ is the restriction of the function W to the set E. We call $\mathcal{G}_1 \cup \mathcal{G}_2$, the weighted undirected graph $\mathcal{G} = (V, E, W)$ with $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, and for all $e \in E$:

$$W(e) = \begin{cases} W_1(e) & \text{if } e \in E_1, \\ W_2(e) & \text{if } e \in E_2. \end{cases}$$

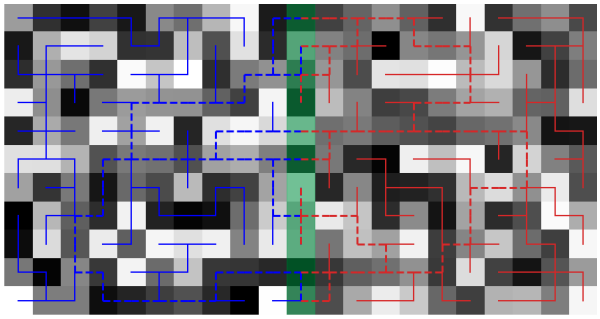


Figure 4.1 $\mathcal{T}_0 = \mathcal{MST}(\mathcal{I}_0)$ in red and $\mathcal{T}_1 = \mathcal{MST}(\mathcal{I}_1)$ in blue. $E_{\mathcal{T}_0}$ and $E_{\mathcal{T}_1}$ in bold and dashed, edges linking common pixels (in emerald) and candidate to form cycles on the union of the two MST.

Definition 4.2. Given a weighted graph $\mathcal{G} = (V, E, W)$ and a subset $E' \subseteq E$ of the edges, we call $\mathcal{G} - E'$ the graph $(V, E \setminus E', W)$ obtained by removing the edges E' from $\mathcal{G}$.

Furthermore, $\mathcal{MST}(\cdot)$ indicates the function that returns a minimum spanning tree of $\mathcal{G}$. $E(\mathcal{G})$ is the set of all edges in $\mathcal{G}$. In order to simplify the notation, from now on we treat both images and graphs as the same objects.

Finally, we come back to our question. Let $\mathcal{G}_t$, a graph streaming over time. This is, at each interval t a new block of the complete graph $B_t = (V_t, E_t, W_t)$ arrives. So it holds:

$$\mathcal{G}_t = \bigcup_{s=0}^{t} B_s.$$

Assume that at time $t-1$ the intersection between $\mathcal{G}_{t-1}$ and the new block B_t is known and it is never empty. In this context, we address the problem of updating the MST of the graph $\mathcal{G}_{t-1}$ each time that a block B_t arrives. In particular, we show two algorithms capable to build a MST of the graph $\mathcal{G}_t$, that exploit information coming from time $t-1$.

We conclude this section showing how this formulation can be applied to a stream of images. Let $\mathcal{I}_t$ be an image streaming over time. Without loss of generality, assume that new pixels come from one side of the image, for example the right side of the image. If B_t is the new block at time t, for $t = 0, \ldots, T$, we have $\mathcal{I}_t = \mathcal{I}_{t-1} \cup B_t$. The last column of $\mathcal{I}_{t-1}$ is also the first column of B_t, as in Figure 4.2a.

For other streaming configurations, as for example a $2D$ tiling, the only difference is that the common pixels should be along the common tiling boundaries, as in Figure 4.2b.

Figure 4.1 illustrates the union of two minimum spanning trees.

4.3 Minimum Spanning tree of a flow of graphs.

We introduce two methods to compute a MST for streaming by examining the case of the union $\mathcal{G}$ of two MSTs $\mathcal{T}_0$ and $\mathcal{T}_1$, as shown in Figure 4.1. In order to extract a MST from the graph $\mathcal{G}$ we need to locate all the cycles contained in it. Then, remove the heaviest edges from them. Therefore, as first step, we need to find edges forming cycles in $\mathcal{G}$. Please remark that all cycles contained in the union pass through the pixels in the common intersection between the two trees at least twice. To simplify our explanation, from now on we will refer to this common intersection as frontier. Indeed, it is straightforward to prove that a cycle in

(a) 1-D streaming

(b) 2-D streaming

Figure 4.2 (a) Example of a one-dimensional streaming of an image $\mathcal{I}_t$. This can be seen as the union of the two non-disjoint images $\mathcal{I}_{t-1}$ and B_t. Without loss of generality, they share a column of pixels. (b) Example of a two-dimensional streaming of an image. In this case the image is the union of tiles who share common boundaries.

$\mathcal{G}$ is generated each time the two trees do not share an edge in the frontier. When this happens, we can find two different paths to join two vertices in the frontier. Without loss of generality, let u, v be two vertices in the frontier. Let's assume that $e = (u, v) \in \mathcal{T}_0$ but $e \notin \mathcal{T}_1$. Since $\mathcal{T}_1$ is connected we can find a different path $\pi = \{v_0 = u, \ldots, v_n = v\}$ in $\mathcal{T}_1$ that joins u and v. Thus $\pi \cup \{e\}$ is a cycle in $\mathcal{G}$. Generalizing this idea, we can retrieve all the cycles looking for the subgraph $\mathcal{G}'$ made by the union of the subgraph $E_{\mathcal{T}_0}$ and $E_{\mathcal{T}_1}$ that contains all the paths in $\mathcal{T}_0$ and $\mathcal{T}_1$ that link any two vertices of the frontier. In Figure 4.1 we draw the edges that belong to $\mathcal{G}' = E_{\mathcal{T}_0} \cup E_{\mathcal{T}_1}$ with bold and dashed lines respectively. Since we are working with trees, it turns out it is easier to find all the paths from any node in the frontier N to a special node r marked as root. To do so, we mark one node in the frontier as root of the MST and traverse the tree twice. The first time, we traverse the tree in a top-down fashion using the depth-first-search-algorithm to build the vector of predecessors. The second time, the tree is traversed from bottom-up. This allows us to store all the edges in paths from the root to any other node in the frontier. We call this procedure *find_unstable_edges*. See the pseudo-code in Procedure 4. The name of this last procedure will be clearer in the next section. Its main purpose is to identify the edges in the tree that may possibly generate a cycle in the following iterations.

Procedure 4 find_unstable_edges

Input: A minimum spanning tree $\mathcal{T}$, root node r and the list of nodes $N = [n_1, \ldots, n_h] \subset \mathbb{N}$ in the frontier
Output: E list of edges linking r to another frontier node N.

```
1:  procedure FIND_UNSTABLE_EDGES
2:      // each node in the graph is identified by a number n ∈ ℕ
3:      E ← ∅ // edges to return
4:      V ← (False, ..., False) // visited nodes
5:      // p[n] : predecessor map of tree nodes. p[r] = −1.
6:      p ← depth_first_order(T, r)
7:      for n ∈ N do
8:          while p[n] >= 0 & V[n] == False do
9:              E ← E ∪ {(n, p[n])}
10:             V[n] = True
11:             n ← p[n]
```

- **Streaming Spanning Tree v1:** At the step t, the graph $\mathcal{G}$ made by the union of $\mathcal{MST}(\mathcal{G}_{t-1})$ and $\mathcal{MST}(B_t)$ may contain cycles. The idea is to identify all the edges that may cause cycles using the

Procedure 4 on both graphs $\mathcal{MST}(\mathcal{G}_{t-1})$ and $\mathcal{MST}(B_t)$. Let $E_{\mathcal{G}_{t-1}}$ and E_{B_t} be all the paths in $\mathcal{MST}(\mathcal{G}_{t-1})$ and $\mathcal{MST}(B_t)$ respectively connecting any two nodes in the frontier N_t.

The method computes $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup E_{B_t})$, and finally it returns:

$$\mathcal{MST}(\mathcal{G}_t) = (\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}) \cup (\mathcal{MST}(B_t) - E_{B_t}) \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup E_{B_t})$$

- **Streaming Spanning Tree v2:** At step t, we consider the graph $\mathcal{G}$ made by the union of $\mathcal{MST}(\mathcal{G}_{t-1})$ and B_t. As in v1, we consider the graph $E_{\mathcal{G}_{t-1}}$ composed of the paths in $\mathcal{MST}(\mathcal{G}_{t-1})$ that link any two vertices of the common frontier because they may possibly generate cycles in $\mathcal{G}$. Instead of computing a MST for the newly arrived B_t. We compute a MST of B_t combined with the candidate edges to form cycles of the previous step. The result is added to $\mathcal{MST}(\mathcal{G}_{t-1})$ without the candidate edges to form cycles.

$$\mathcal{MST}(\mathcal{G}_t) = (\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}) \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$$

Please remark that graph $\mathcal{MST}(\mathcal{G}_{t-1})$ contains edges coming from B_s, $\forall s = 0, \ldots, t-1$. Thus the subgraph $E_{\mathcal{G}_{t-1}}$ could contain edges that belong to any previous block. Procedures 5 and 6 show the pseudo-code for the two methods.

Procedure 5 Streaming Spanning Tree v1

Input: A streaming graph $\mathcal{G}_t$
Output: A MST for the graph $\mathcal{G}_t$

procedure STREAMING_SPANNING_TREE_V1
  $\mathcal{T}_0 \leftarrow \mathcal{MST}(\mathcal{G}_0)$
  // N_1 frontier with block B_1, $r_1 \in N_1$
  $E_{\mathcal{G}_0} =$ *find_unstable_edges*$(\mathcal{T}_0, r_1, N_1)$
  $\mathbf{F}_0 \leftarrow \mathcal{T}_0 - E_{\mathcal{G}_0}$
  $\mathbf{T}_0 \leftarrow \mathbf{F}_0 \cup E_{\mathcal{G}_0}$
  while a new block B_t arrives **do**:
    $\mathcal{T}_t \leftarrow \mathcal{MST}(B_t)$
    // N_t frontier with $\mathcal{T}_t$, $r_t \in N_t$
    $E_{B_t} \leftarrow$ *find_unstable_edges*$(\mathcal{T}_t, r_t, N_t)$
    $\mathcal{F}_t \leftarrow \mathcal{T}_t - E_{B_t}$
    $\mathcal{T} \leftarrow \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup E_{B_t})$
    $\mathbf{T}_t \leftarrow \mathbf{F}_{t-1} \cup \mathcal{F}_t \cup \mathcal{T}$
    // Fetching $E_{\mathcal{G}_t}$ for next iteration
    // N_{t+1} frontier with block B_{t+1}, $r_{t+1} \in N_{t+1}$
    $E_{\mathcal{G}_t} \leftarrow$ *find_unstable_edges*$(\mathbf{T}_t, r_{t+1}, N_{t+1})$
    $\mathbf{F}_t \leftarrow \mathbf{T}_t - E_{\mathcal{G}_t}$

In order to prove the correctness of our methods, we prove the following theorem.

Theorem 4.3. *All the proposed methods return a minimum spanning tree for the graph $\mathcal{G}_t$, for each t.*

Proof. We only prove that the second method returns a MST. The goal is to show that the graph

$$\mathcal{T}_t = (\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}) \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$$

Procedure 6 Streaming Spanning Tree v2

Input: A streaming graph $\mathcal{G}_t$
Output: A minimum spanning tree MST for the graph $\mathcal{G}_t$

1: **procedure** STREAMING_SPANNING_TREE_V2
2: $\mathcal{T}_0 \leftarrow \mathcal{MST}(B_0)$
3: *// N_1 frontier with block B_1, $r_1 \in N_1$*
4: $E_{\mathcal{G}_0} =$ *find_unstable_edges*$(\mathcal{T}_0, r_1, N_1)$
5: $\mathbf{F}_0 \leftarrow \mathcal{T}_0 - E_{\mathcal{G}_0}$
6: $\mathbf{T}_0 \leftarrow \mathbf{F}_0 \cup E_{\mathcal{G}_0}$
7: **while** a new block B_t arrives **do**:
8: $\mathbf{T}_t \leftarrow \mathbf{F}_{t-1} \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$
9: *// Fetching $E_{\mathcal{G}_t}$ for next iteration*
10: *// N_{t+1} frontier with block B_{t+1}, $r_{t+1} \in N_{t+1}$*
11: $E_{\mathcal{G}_t} \leftarrow$ *find_unstable_edges*$(\mathbf{T}_t, r_{t+1}, N_{t+1})$
12: $\mathbf{F}_t \leftarrow \mathbf{T}_t - E_{\mathcal{G}_t}$

returned by Procedure 6 at time t respects the following three properties: 1) $\mathcal{T}_t$ is connected, 2) $\mathcal{T}_t$ is a spanning tree, 3) the sum of all weights $\sum_{e \in E_t} w_e$, is minimal.

First of all, we prove that $\mathcal{T}_t$ is connected.

We show that for any $u, v \in \mathcal{G}_t$ there exists a path $\pi = \{v_0 = u, \ldots, v_n = v\}$ contained in $\mathcal{T}_t$. Let now consider

$$\mathcal{T}'_t = \mathcal{MST}(\mathcal{G}_{t-1}) \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t) \supseteq \mathcal{T}_t.$$

By construction $\mathcal{T}'_t$ is connected since is the union of two connected graphs with a non-empty intersection. For this reason, it is possible to find a path $\pi = \{v_0 = u, \ldots, v_n = v\}$ contained in $\mathcal{T}'_t$.

Using the fact that $\mathcal{T}'_t \supseteq \mathcal{T}_t$, we can conclude that either $\pi \subseteq \mathcal{T}_t$ or it must exist an edge $e = (v_i, v_{i+1})$ such that $e \notin E(\mathcal{T}_t)$. In this last case, by construction $e \in E_{\mathcal{G}_{t-1}}$, but $e \notin \mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}$. However, since $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$ is a connected tree, we can find another path $\pi_1 = \{w_0 = v_i, \ldots, w_m = v_j\} \subseteq \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$ and thus in $\mathcal{T}_t$. Repeating this procedure for all the edges $e = (v_i, v_{i+1})$ of the path π not in $E(\mathcal{T}_t)$, we can build a path π' from u to v entirely contained in $\mathcal{T}_t$. Therefore, $\mathcal{T}_t$ is connected.

Second, we prove that $\mathcal{T}_t$ is a tree by contradiction. Let's assume that $\mathcal{T}_t$ contains cycles. Thus, suppose that $\exists v \in V(\mathcal{G}_t)$ and a path $\pi = \{v_0 = v, \ldots, v_n = v\}$ that is a cycle in $\mathcal{T}_t$.

By definition of $\mathcal{T}_t$ it is straightforward that such a cycle cannot be contained entirely in $\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}$ nor in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$, since they are respectively a forest and a tree. So, the cycle π must pass through the nodes and edges of both graphs. In particular, it must cross at least twice the frontier between $\mathcal{G}_{t-1}$ and B_t. By definition, all paths in $\mathcal{G}_{t-1}$ linking two nodes of the frontier are included in $E_{\mathcal{G}_{t-1}}$. Therefore, $\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}$ cannot contain such edges. As a consequence, $\mathcal{T}_t$ has no cycles.

Finally, we prove the third property by contradiction. We define the cost of a graph $\mathcal{G}$ as the sum of its weights $cost(\mathcal{G}) = \sum_{e \in E(\mathcal{G})} w_e$. So, let suppose that $\mathcal{T}_t$ is not minimal. Then there exists $e \in \mathcal{G}_t$ s.t. $\mathcal{T}_t \cup \{e\}$ contains a spanning tree $\mathcal{T}$ such that $cost(\mathcal{T}) < cost(\mathcal{T}_t)$. Remark that, by definition of $\mathcal{T}_t$, the edge e cannot belong to $E(E_{\mathcal{G}_{t-1}} \cup B_t)$. Otherwise, the edge e would be contained also in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$, which is a contradiction. Let now consider e' the edge in $\mathcal{T}_t$ replaced by e in $\mathcal{T}$. Since the two trees differ only by the two edges, it holds $w_e < w_{e'}$. Two cases are possible:

1. e' is an edge originally in $\mathcal{MST}(\mathcal{G}_{t-1})$,

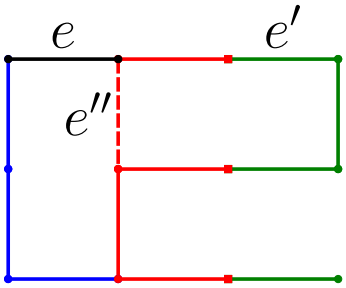


Figure 4.3 In black the edge e, while in blue the edges in $\mathcal{MST}(G_{t-1} - E_{\mathcal{G}_{t-1}})$, in red edges coming from $E_{\mathcal{G}_{t-1}}$. In particular, the dashed red edges are edges initially in $E_{\mathcal{G}_{t-1}}$ but not in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$.

2. e' is an edge originally in the new block B_t, this is, $e' \in E(\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)) \cap E(B_t)$, as in Figure

Both cases lead to a contradiction. In fact, suppose that e' was originally in $\mathcal{MST}(\mathcal{G}_{t-1})$, then $\mathcal{MST}(\mathcal{G}_{t-1}) \cup \{e\}$ contains a cycle that pass through e and e'. If $w_e < w_{e'}$ $\mathcal{MST}(\mathcal{G}_{t-1})$ had chosen e instead of e' but it is not the case. On the other hand, if e' is contained in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t) \cap B_t$, then we can find e'' that belongs to $E_{\mathcal{G}_{t-1}}$ but not in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$, such that $\mathcal{T}_t \cup \{e''\}$ has a cycle containing e' and e'' (see red dashed line in figure 4.3). Since e'' is not contained in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$ we can deduce that $w_{e''} \geq w_{e'}$, and thus $w_{e''} > w_e$. As in the previous case, this lead to a contradiction because it implies that $\mathcal{MST}(\mathcal{G}_{t-1}) \cup \{e\}$, with $e \in \mathcal{G}_{t-1}$ contains a MST $\mathcal{T}'$ such that $cost(\mathcal{T}') < cost(\mathcal{MST}(\mathcal{G}_{t-1}))$.

□

We conclude this section showing an interesting insight of the proposed methods that will be useful for possible applications. As the reader may have already noticed, at the end of each iteration t, we decompose the $\mathcal{MST}(\mathcal{G}_t)$ in two disjoint parts: a) $E_{\mathcal{G}_t}$, a graph made of all edges in $\mathcal{MST}(\mathcal{G}_t)$ that may form cycles when the new block arrives. b) $\mathbf{F}_t = \mathbf{T}_t - E_{\mathcal{G}_t}$, a forest made by all the rest of the graph. The first graph, is indeed made by edges that we call *unstable*. Mostly because we could eliminate some of them in the next step $t+1$. The second graph is made by edges that we call *stable*, since they will belong to all MSTs from now on. This is important for two reasons. 1) At each step, the memory footprint is reduced by discarding edges that are no longer necessary to compute further MSTs. 2) The *stable* edges can be used for further tasks, as we will see below. In Figure 4.4, we report an example that shows the evolution of the *stable* + *unstable* decomposition of minimum spanning trees through the time. In green, we represent the forest $\mathbf{F}_t$ over time, while in red the graph $E_{\mathcal{G}_t}$.

4.4 A divide and conquer implementation

After having proved the correctness of proposed procedures, let move on showing a divide and conquer algorithm to compute in parallel MST based on Procedure 5. As before, we introduce our method using an example. Let I an image as in the left of Figure 4.5. Our a strategy is similar to Merge Sort Algorithm (Cormen, 2009, Chapter 2). The method is divided in three phases: divide, conquer and combine. In the divide phase, it recursively splits the input image into four tiles until an atomic size (defined by the user) is reached. Also in this case, the overlap region between the four parts is not empty. In particular, two blocks on the same line shares a column while the intersection between blocks on the same file is a row as illustrated in Figure 4.2b. Then in the conquer phase, a MST is computed for each atomic tile. Moreover, the edges of each MST are divided in *stable* and *unstable*. Finally, in the combine phase, a strategy as in Procedure 5 is used to recursively merge the MSTs of four tiles from the bottom to the top. Namely, at

(a) $t = 0$ **(b)** $t = 1$

(c) $t = 2$

Figure 4.4 An example of *stable* + *unstable* decomposition of minimum spanning tree. The green graph is the forest $\mathbf{F}_t$ that contains only *stable edges*, while the red graph is $E_{\mathcal{G}_t}$ that contains only *unstable edges*. (b-c) Pixels without edges are stable, so it is possible to store that part of the graph and do not need to consider in following intervals.

each step, the method extracts a MST from the union of the unstable edges in the four tiles. Successively, this last tree is decomposed in *stable+unstable* edge and stable edges are added to stable edges already found in the four tiles. Finally, the *stable/unstable* decomposition is returned for the consecutive step. In Procedure 7, we show the pseudo-code.

4.5 Benchmarks

We conducted our experiments on a high-resolution image taken from the site http://www.gigapan.com/. The image is (12000×47196) pixels high-resolution photo of Planet Mars' surface[1]. The CPU of the machine used for the tests is Intel 3.00 GHz Xeon with 32 GB RAM. We considered the case as shown in Section 4.2 where blocks of an image stream horizontally.

As first benchmark, we measured how long our methods took to build the MST for the complete image, and we compared this against the *brute-force* method, that is the method that load in memory all the image and then compute the MST using Kruskal approach. We make the image stream in blocks B_t of size 12000×4000 pixels, and at each interval t we measure the time the algorithms take to compute $\mathcal{MST}(\mathcal{I}_t)$. In Figure 4.6, we report the results. During the experiment, we remarked that the *brute-force* method is faster compared to our algorithms until the image size reaches the limit to fit entirely in RAM. In our machine, we tested this limit at around 240 megapixels (Mpx). After that, the *brute-force* method

[1] http://www.cmm.mines-paristech.fr/~gigli/mars.jpg

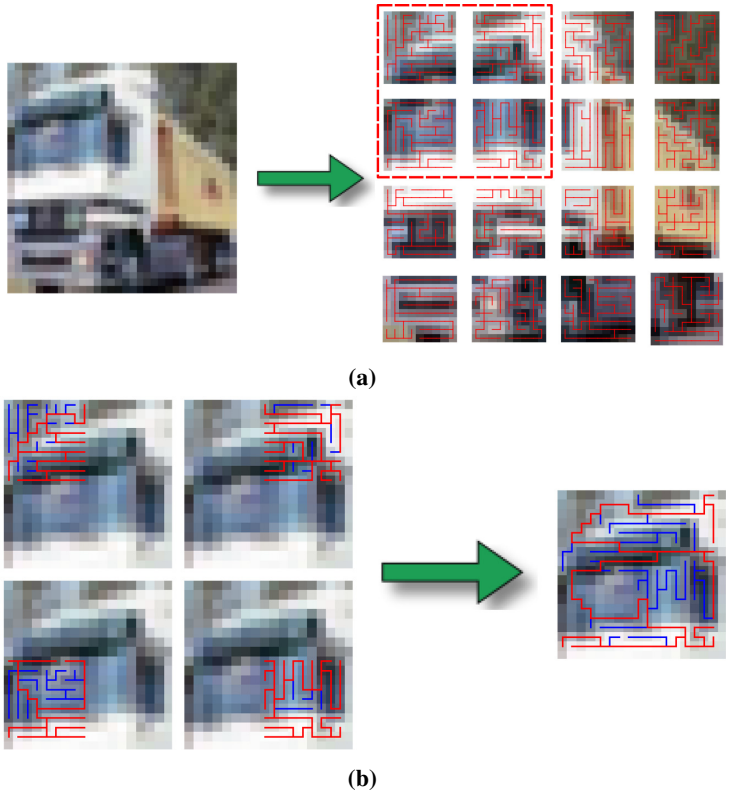


Figure 4.5 Steps of Procedure 7. a) It first splits top-down the input image until an atomic block size is reached. Then it computes an MST for each tile. b) Finally, it merges bottom-up the MSTs of tiles, decomposing at each time edges of an MST between *stable* and *unstable*. In figure, we draw in blue stable edges and in red unstable ones.

needs to use *swap memory*, and its runtime grows until the image size reaches around 380 Mpx, when the image size makes the program crash. On the contrary, thanks to the *stable* + *unstable* decomposition that reduces the memory footprint of Procedures 5 and 6, the execution times of our methods grow quasi-linearly with the image size, and they can treat images of bigger sizes.

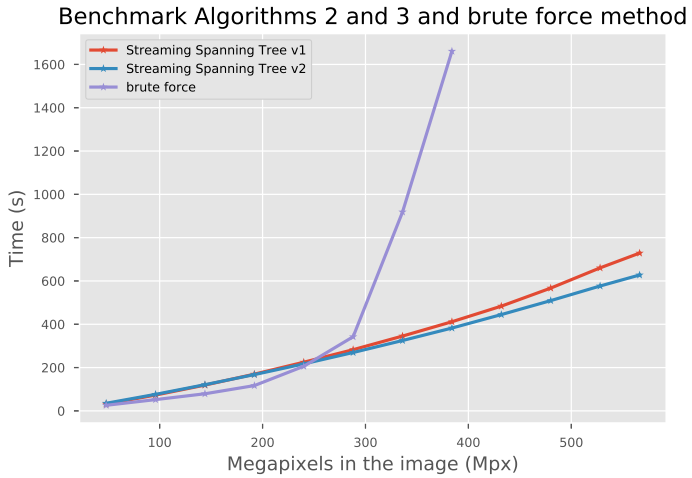


Figure 4.6 Runtime of Procedure 5, Procedure 6 and *brute-force* algorithm on a (12000×47196) pixels image of Planet Mars' surface.

Procedure 7 Parallel MST

Input: An input Image I, size (n_0, m_0) of the atomic element
Output: A MST for the image I

procedure PARALLEL_MST(I, n_0, m_0)
  // Get size of input image
  $n, m \leftarrow I$.shape
  if $n < n_0$ or $m < m_0$ **then**
    $\mathcal{T} \leftarrow \mathcal{MST}(I)$
    // the frontier is made of pixels on the border ∂I of the image
    $E_I \leftarrow$ *find_unstable_edges*$(\mathcal{T}, r_I, \partial I)$
    $F_I \leftarrow \mathcal{T} - E_I$ *// Decomposing $\mathcal{T}$ between stable and unstable edges*
  else
    // split the image in four tiles
    $I_{tl}, I_{tr} \leftarrow I[0 : n/2+1, 0 : m/2+1], I[0 : n/2+1, m/2 : m]$
    $I_{bl}, I_{br} \leftarrow I[n/2 : n, 0 : m/2+1], I[n/2 : n, m/2 : m]$
    // recursively call the function on each tile
    $E_{tl}, F_{tl} \leftarrow$ *parallel_mst*(I_{tl}, n_0, m_0)
    $E_{tr}, F_{tr} \leftarrow$ *parallel_mst*(I_{tr}, n_0, m_0)
    $E_{bl}, F_{bl} \leftarrow$ *parallel_mst*(I_{bl}, n_0, m_0)
    $E_{br}, F_{br} \leftarrow$ *parallel_mst*(I_{br}, n_0, m_0)
    $\mathcal{T}_I \leftarrow \mathcal{MST}(E_{tl} \cup E_{tr} \cup E_{bl} \cup E_{br})$
    $E_I \leftarrow$ *find_unstable_edges*$(\mathcal{T}, r_I, \partial I)$
    $F_I \leftarrow (\mathcal{T} - E_I) \cup (F_{tl} \cup F_{tr} \cup F_{bl} \cup F_{br})$
  return E_I, F_I

Furthermore, in order to better understand how the size of the blocks B_t affects the runtime, we measured the execution times of our methods with different sizes for streaming blocks that are 12000×4000, 12000×8000 and 12000×12000 pixels. We report the results obtained in Figure 4.7. As the reader can see, Procedure 6 performs better than Procedure 5 in the general case. Moreover, we noted an improvement in the execution times to treat the entire image when we use block sizes of 12000×8000 pixels compared with block size of 12000×4000 pixels. This is because, using bigger blocks, we reach the entire image dimensions in fewer iterations, and we spend less time updating the MSTs. Nevertheless, using blocks of bigger sizes also increases the quantity of memory needed at each iteration and thus increases the risk to use *swap memory* that slows down the overall execution time. For this reason, in the case of Procedure 6 we do not remarked the same improvement when we use blocks size of 12000×12000 pixels compared with the ones of 12000×8000 pixels. Thus, the ideal block size should be a trade-off between the footprint of the block B_t and the number of iterations needed to treat the entire image.

Time Complexity

Let now evaluate the time complexity of the Procedure 6. Let $\mathcal{G} = (V, E)$ be a connected graph, and assume that we split it in blocks $\mathcal{G} = \cup_{t=0}^{T} B_t$, where $B_t = (V_t, E_t)$ are blocks as shown in Section 4.3. Moreover, let $|V| = N$, $|E| = M$ and $|V_t| = n_t$, $|E_t| = m_t$. Finally, let $E_{B_t} = (\tilde{V}_t, \tilde{E}_t)$ the graphs containing all the vertices and edges appearing in the unstable paths in $\mathcal{MST}(\cup_{s=0}^{t} B_s)$ for $t \geq 0$. We define $|\tilde{V}_t| = \nu_t$ and $|\tilde{E}_t| = \mu_t$ for all $t \geq 0$ and $\nu_{-1} = \mu_{-1} = 0$.
With $f(N, M)$ we indicate the time complexity of our method to compute the final $\mathcal{MST}(G)$. For the

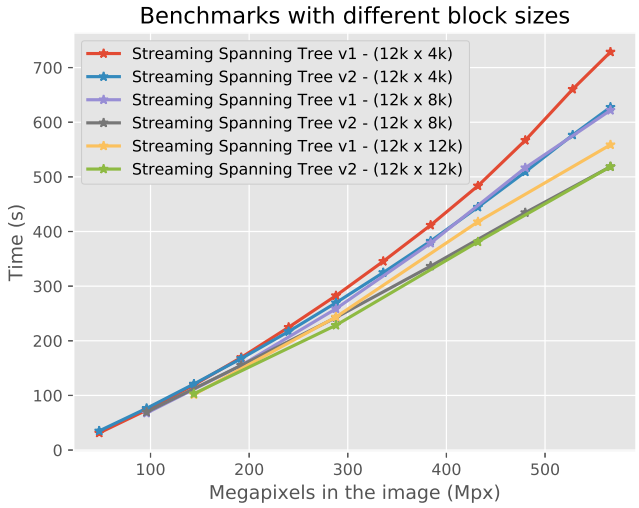


Figure 4.7 Runtime of Procedures 5 and 6 with different block sizes. We used blocks of 12000×4000, 12000×8000 and 12000×12000 pixels.

moment let call $\mathcal{A}$ the algorithm used to compute any MST of a connected graph, and let $c(n,m)$ its complexity, where n,m are respectively the number of vertices and the number of edges in the graph. From now on, when we write $f(\mathcal{G})$, we mean $f(N,M)$.
At time $t=0$ the complexity is $f(n_0,m_0)=c(n_0,m_0)$. At time $t=1$, we have

$$\begin{aligned} f(B_0 \cup B_1) &= f(n_0+n_1, m_0+m_1) = c(n_0,m_0) + O(n_0) + c(B_1 \cup E_{B_0}) \\ &= \underbrace{c(n_0,m_0)}_{\text{cost } t=0} + \underbrace{O(n_0)}_{\text{cost to find } |E_{B_0}|} + \underbrace{c(n_1+\nu_0, m_1+\mu_0)}_{\mathcal{MST} \text{ new part}} \end{aligned}$$

where $O(n_0)$ is the time spent to find unstable paths in E_{B_0}, and $c(n_1+\nu_0, m_1+\mu_0)$ it is obviously the time spent to update the MST for the new part. In fact, to find paths that make E_{B_0} we navigate the tree twice, so the cost is $O(n_0)$.
The cost at time $t=1$, can be bounded from below and from above by

$$c(n_0,m_0) + O(n_0) + c(n_1,m_1) \leq f(B_0 \cup B_1) \leq c(n_0,m_0) + O(n_0) + c(n_1+n_0, m_1+n_0),$$

where the lower bound represents the best case in which $E_{B_0} \subset B_1$, and the upper bound is the worst case where $E_{B_0} = \mathcal{MST}(B_0)$. The worst case is when $\mathcal{MST}(B_0)$ has a comb shape facing the frontier. Please remark that in the worst case E_{B_0} has $n_0 - 1$ edges. At time $t=2$ the time is:

$$\begin{aligned} f(\cup_{t=0}^{2} B_t) = & c(n_0,m_0) + \underbrace{O(n_0) + c(n_1+\nu_0, m_1+\mu_0)}_{\text{cost update } t=1} \\ & + \underbrace{O(n_0+n_1) + c(n_2+\nu_1, m_2+\mu_1)}_{\text{cost update } t=2}, \end{aligned} \tag{4.1}$$

where we are assuming no optimization in the process of finding the unstable edges composing E_{B_1}. The above Equation 4.1 can be rewritten as

$$f(\cup_{t=0}^{2} B_t) = c(n_0, m_0) + c(n_1 + \nu_0, m_1 + \mu_0) + c(n_2 + \nu_1, m_2 + \mu_1) + O(2n_0 + n_1)$$

Now, it is easy to generalize our cost for time $t = T$:

$$f(\mathcal{G}) = f(\cup_{t=0}^{T} B_t) = \sum_{t=0}^{T} c(n_t + \nu_{t-1}, m_t + \mu_{t-1}) + O\Big(\sum_{t=0}^{T-1} (T-t) n_t\Big).$$

So $f(N, M)$ is bounded by the following two quantities:

$$\underbrace{\sum_{t=0}^{T} c(n_t, m_t) + O\Big(\sum_{t=0}^{T-1} (T-t) n_t\Big)}_{\text{best case: } E_{B_t} \subset B_{t+1}\ \forall t} \leq f(N, M) \leq \underbrace{\sum_{t=0}^{T-1} c(\sum_{s=0}^{t} n_s, m_t + \sum_{s=0}^{t-1} n_s) + O\Big(\sum_{t=0}^{T-1} (T-t) n_t\Big)}_{\text{worst case: } E_{B_t} = \mathcal{MST}(\cup_{s=0}^{t} B_s)\ \forall t}. \tag{4.2}$$

Assuming that we are splitting uniformly the graph using blocks all of the same size, *i.e.* $n_t = n$, and $m_t = m\ \forall t$, we can further develop the equation 4.2.

$$\sum_{t=0}^{T} c(n, m) + O\Big(\sum_{t=0}^{T-1} (T-t) n\Big) \leq f(N, M) \leq \sum_{t=0}^{T} c(\sum_{s=0}^{t} n, m + \sum_{s=0}^{t-1} n) + O\Big(\sum_{t=0}^{T-1} (T-t) n\Big),$$

that becomes,

$$(T+1) c(n, m) + O\Big(n \frac{T(T+1)}{2}\Big) \leq f(N, M) \leq \sum_{t=0}^{T} c((t+1)n, m + tn) + O\Big(n \frac{T(T+1)}{2}\Big).$$

Since in our case we used Kruskal as algorithm $\mathcal{A}$ to compute the MST, then $c(n, m) = O(m \log(m))$ and we can develop both upper bound and lower bound expressions. Let start with the lower bound:

$$\begin{aligned} (T+1) c(n, m) + O\Big(n \frac{T(T+1)}{2}\Big) &= O((T+1) m \log(m)) + O\Big(n \frac{T(T+1)}{2}\Big) \\ &= O\Big(\frac{(T+1)}{2} (2m \log(m) + nT)\Big). \end{aligned} \tag{4.3}$$

On the other hand, we can rewrite the upper bound as:

$$\begin{aligned} \sum_{t=0}^{T} c((t+1)n, m + tn) + O\Big(n \frac{T(T+1)}{2}\Big) = O\Big(\sum_{t=0}^{T} (m + tn) \log(m + tn)\Big) + \\ O\Big(n \frac{T(T+1)}{2}\Big). \end{aligned} \tag{4.4}$$

Observe that the first sum in the r.h.s. in Equation 4.4 can be rewritten as by:

$$\begin{aligned}\sum_{t=0}^{T}(m+tn)\log(m+tn) &= m\log(m)+\sum_{t=1}^{T}\log((m+tn)^{(m+tn)})\\ = m\log(m)+\log\Big(\prod_{t=1}^{T}(m+tn)^{(m+tn)}\Big) &= m\log(m)+\log(H(m+nT)),\end{aligned} \tag{4.5}$$

Where $H(\cdot)$ is the Hyperfactorial function defined as $H(n)=\prod_{k=0}^{n}k^k$. Thus, the cost $f(N,M)$ is bounded by

$$O\Big(\frac{(T+1)}{2}(2m\log(m)+nT)\Big)\leq f(N,M)\leq O\Big(m\log(m)+\log(H(m+nT))+n\frac{T(T+1)}{2}\Big)$$

Finally, we can approximate $N\approx nT$ and obtain

$$O\Big(\frac{(T+1)}{2}(2m\log(m)+N)\Big)\leq f(N,M)\leq O\Big(m\log(m)+\log(H(m+N))+\frac{N(T+1)}{2}\Big).$$

However, the term $\sum_{t=0}^{T}(m+tn)\log(m+nt)$ in Equation 4.5 can be further expanded. In fact, let $a_t=\log(m+nt)$ and $b_t=m+nt$, then using summation by parts we can write

$$\begin{aligned}\sum_{t=0}^{T}(m+tn)\log(m+nt)=\sum_{t=0}^{T}a_tb_t &= a_T\sum_{t=0}^{T}b_t-\sum_{t=0}^{T-1}(a_{t+1}-a_t)\Big(\sum_{s=0}^{t}b_s\Big)\\ &= \log(m+nT)\sum_{t=0}^{T}(m+nt)-\sum_{t=0}^{T-1}\log\Big(1+\frac{1}{\frac{m}{n}+t}\Big)\Big(\sum_{s=0}^{t}(m+ns)\Big)\\ &= \frac{(T+1)}{2}\log(m+nT)(2m+nT)-\sum_{t=0}^{T-1}\frac{t+1}{2}\log\Big(1+\frac{1}{\frac{m}{n}+t}\Big)(2m+nt)\\ &\leq \frac{(T+1)}{2}\log(m+nT)(2m+nT).\end{aligned} \tag{4.6}$$

Please remark that this last upper bound is indeed large, but we use it in order to keep the formula simple and have a final explanation. Thus, using the approximation $N\approx nT$ we can rewrite our bounds as

$$O\Big(\frac{(T+1)}{2}(2m\log(m)+N)\Big)\leq f(N,M)\leq O\Big(\frac{(T+1)}{2}\Big(\log(m+N)(2m+N)+N\Big)\Big).$$

Even though the equation above does not look intuitive we can try to get out some understanding from the two bounds. In the best case we can see that the complexity is the same as applying Kruskal $T+1$ times, one on each block, plus the total time spent looking for the unstable graphs, that is $O((T+1)N)$. The complexity of worst-case scenario is upper bounded by the cost of applying Kruskal $T+1$ times on a graph with $m+N$ edges plus the time spent looking for the unstable graphs.

4.6 Application to Image analysis

In this section, we introduce three applications of our algorithm to image segmentation. In particular, we exploited the *stable* + *unstable* decomposition of our methods to implement a streaming version of

λ-quasi-flat zones (Najman et al., 2013)(Zanoguera and Meyer, 2002), watershed-cuts (Cousty et al., 2009) and constrained connectivity (Soille, 2008) for Remote Sensing images. We have introduced these methods in Section 1.3. To achieve our goal, we use slightly different versions of Procedures 5 and 6, that each time t return the stable forest $\mathbf{F}_t$ and the unstable graph $E_{\mathcal{G}_t}$. Remember that the MST for the graph $\mathcal{G}_t$ can be obtained as $\mathcal{MST}(\mathcal{G}_t) = \cup_{s=0}^{t} \mathbf{F}_s \cup E_{\mathcal{G}_t}$. In order to validate the streaming versions of the previously mentioned methods, we applied them to the case of horizontal streaming as described in Section 4.2. We report in Figure 4.8 the image used for our experiments, that we split in three blocks. The blue dashed lines represent the frontiers between two consecutive blocks, while red pixels are the markers used for the watershed-cuts.

Figure 4.8 Image used to validate streaming version of the segmentation methods. The image has been split in three blocks (see blue dashed lines) and the blocks stream from left to right. As explained in Section 4.2 two consecutive blocks share a column of pixels. In red, the pixels used as markers for watershed-cut.

4.6.1 λ-quasi-flat zones

Let explain how to compute a level of the λ-flat zone hierarchy in a streaming fashion using the example in Figure 4.8. At time $t = 0$ we get $\mathcal{MST}(\mathcal{G}_0) = \mathbf{F}_0 \cup E_{\mathcal{G}_0}$. Now, removing the edges in $\mathcal{MST}(\mathcal{G}_0)$ larger than λ, we obtain $\mathcal{C}_0^{(0)}, \ldots, \mathcal{C}_{n_0}^{(0)}$ connected components. We call *unstable connected component* a region $\mathcal{C}$ containing nodes on the *frontier*. On the contrary, we classify as *stable connected component* a region $\mathcal{C}$ that does not contain any node in the *frontier*.

In fact, it is straightforward that a connected component containing nodes on the frontier can change in the following iteration, for example it can expand and include nodes of future blocks. We conclude that, at each time t, we can assign a label only to *stable connected components*. Conversely, we need to keep in memory the *stable edges* contained in *unstable connected components*. We call residual graph $\mathbf{R}_0$, the graph made by nodes contained in *unstable connected components* and *stable edges* contained in them at time $t = 0$. Thank to this decomposition, to get the connected components in future intervals of time, we do not need the entire $\mathcal{MST}(\mathcal{G}_t)$, but is sufficient to consider the graph $\mathbf{R}_{t-1} \cup \mathbf{F}_t \cup E_{B_t}$ and threshold edges larger than λ from it. To summarize, at each time t we do:

1. Consider the graph $\mathbf{G}_t = \mathbf{R}_{t-1} \cup \mathbf{F}_t \cup E_{B_t}$ where $\mathbf{R}_{t-1}$ is the residual graph obtained at previous step.
2. Threshold all edges in $\mathbf{G}_t$ larger than a given λ, obtaining $\mathcal{C}_0^{(t)}, \ldots, \mathcal{C}_{n_t}^{(t)}$ connected components.

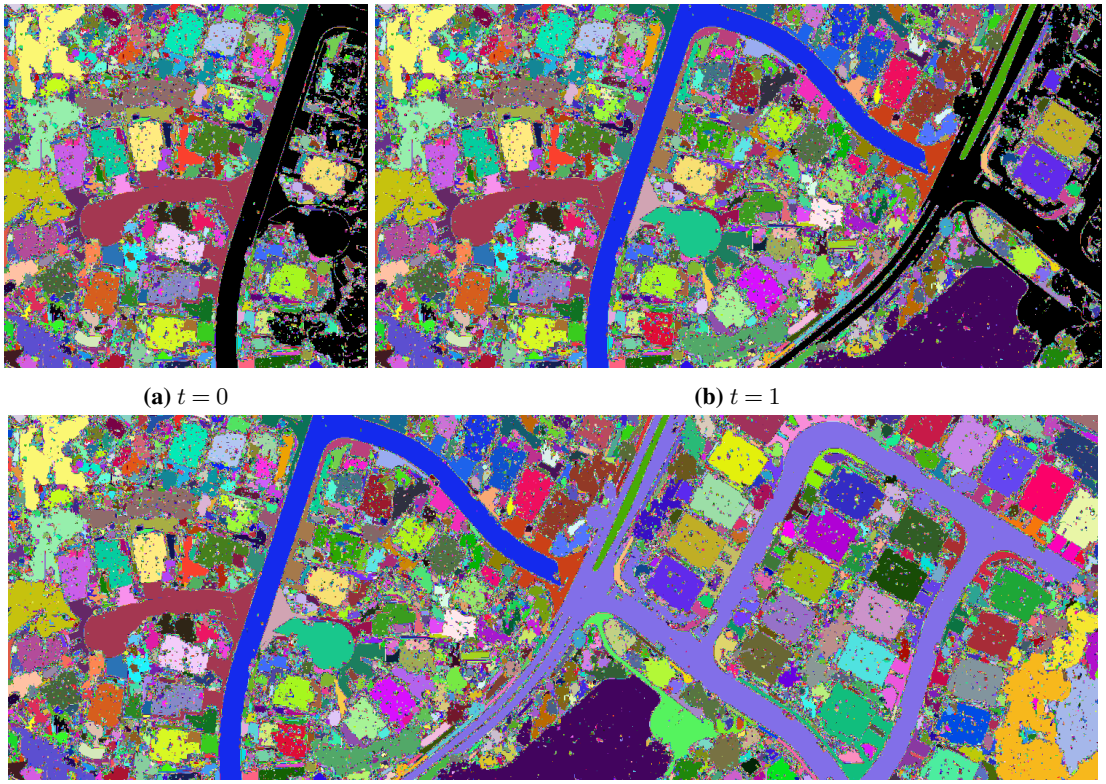
3. Assign a label only to components $\mathcal{C}_j^{(t)}$ whose pixels are not in the frontier. In Figure 4.9 these regions are represented by non-black colors. Mark as *unstable connected components* the regions $\mathcal{C}_j^{(t)}$ containing nodes in the frontier. In Figure 4.9 (a,b) these regions are the black pixels.
4. Assign to $\mathbf{R}_t$ the *stable edges* contained in *unstable connected components* at iteration t.

Figure 4.9 reports the result of the procedure above applied on Figure 4.8.

4.6.2 Watershed cuts

It is possible to derive a streaming version of the watershed cuts like we did to get λ-*quasi-flat zones*. Once again we use the example in Figure 4.8 to illustrate it. At time $t = 0$, we obtain the segmentation as connected components of subgraph $\mathcal{F}' \subseteq \mathcal{MST}(\mathcal{G}'_0)$ as just seen. Also in this case, we split these regions between *unstable* and *stable* based on the criteria of whether they contain frontier nodes or not. Successively, we assign a label only to *stable connected components*, and we retrieve the *residual graph* R_0, that is the graph composed by the nodes in the *unstable connected components* with *stable edges*. Similarly to the previous subsection, to get the connected components in future intervals of time, we do not need the entire $\mathcal{MST}(\mathcal{G}'_t)$, but is sufficient to consider the graph $\mathbf{R}_{t-1} \cup \mathbf{F'}_t \cup E'_{B'_t}$. To summarize, let $\mathcal{G}_t$ a streaming of graph. At each time t we do:

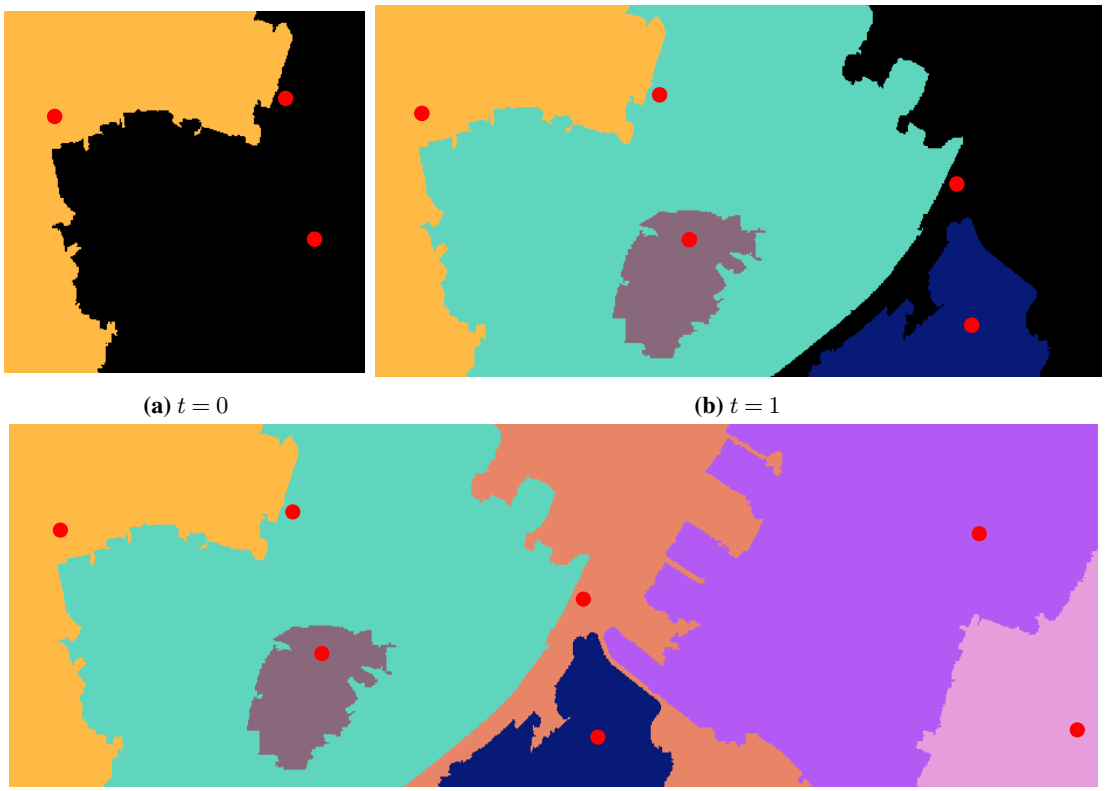
1. Consider the graph $\mathbf{G}'_t = \mathbf{R}_{t-1} \cup \mathbf{F}'_t \cup E_{B'_t}$, where $\mathbf{R}_{t-1}$ is the residual graph obtained at previous step.



(a) $t = 0$ **(b)** $t = 1$

(c) $t = 2$

Figure 4.9 An example of one level of λ-quasi-flat zones in streaming, with $\lambda = 10$ for image in Fig. 4.8. Black pixels in Figures (a) and (b) are those that do not have a stable label in that iteration.



(a) $t = 0$ **(b)** $t = 1$

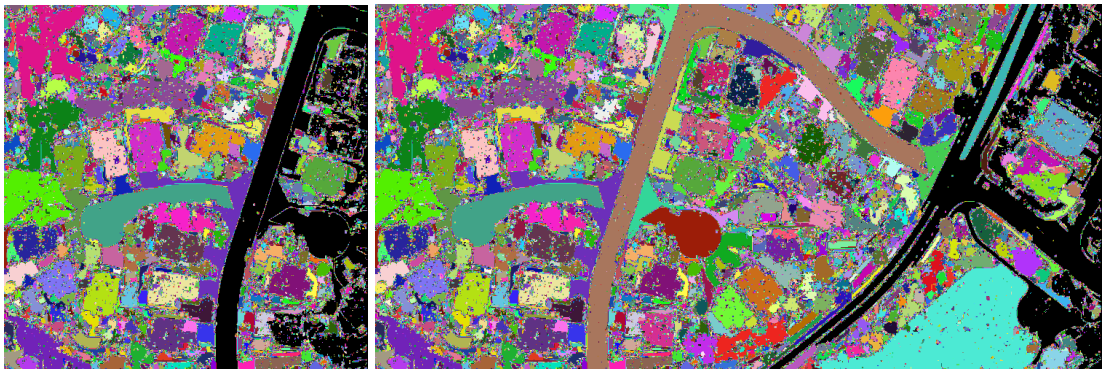
(c) $t = 2$

Figure 4.10 Watershed cuts in streaming for image in Fig. 4.8. Red pixels in the images are the markers of the segmentation. Black pixels in Figures (a) and (b) are the connected components that do not have a stable label in that iteration.

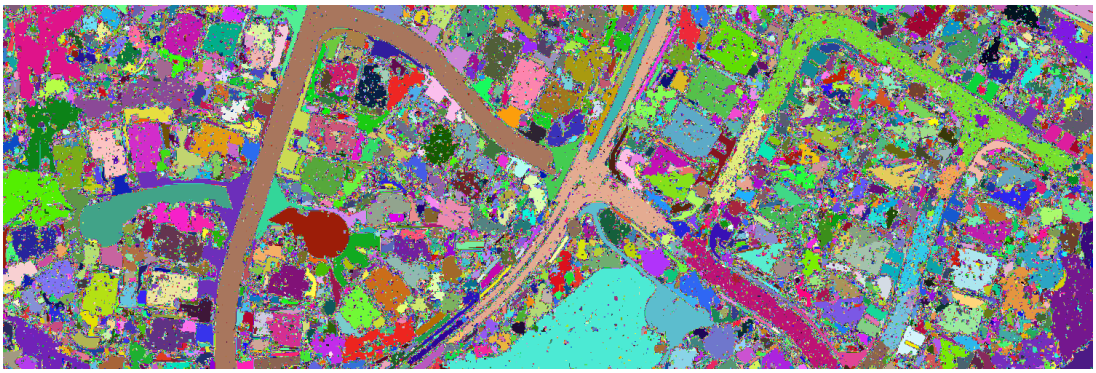
2. Compute $\mathcal{C}_0^{(t)}, \ldots, \mathcal{C}_{n_t}^{(t)}$ connected components of the subgraph obtained removing well z and its connections from $\mathbf{G}'_t$.
3. Assign a label exclusively to components $\mathcal{C}_j^{(t)}$ whose pixels are not in the frontier. In Figure 4.10 these components are non-black regions of the image. Mark as *unstable connected components* the regions $\mathcal{C}_j^{(t)}$ that contain at least one node in the frontier. In Figure 4.10 (a,b): these regions are black pixels.
4. Assign $\mathbf{R}_t$ as the graph made by nodes in *unstable connected components* at iteration t with *stable edges*.

4.6.3 (α, ω)-constrained connectivity

Lastly, let discuss how a streaming version of MST can be used to obtain the (α, ω)-constrained connectivity for a stream of images. We implemented it in a straightforward manner. Basically, at each interval t we first compute the stable α-quasi-flat zones of the image $\mathcal{I}_t$ as in the previous subsection, and then for each stable α connected components we extract the (α, ω) connected components contained in it. In Figure 4.11, we report the result obtained on the test image in 4.8.



(a) $t = 0$ **(b)** $t = 1$



(c) $t = 2$

Figure 4.11 An example of (α,ω)-constrained connectivity in streaming, with $\alpha = 10$ and $\omega = 150$ for image in Fig. 4.8. Black pixels in Figures (a) and (b) are those that do not have a stable label in that iteration.

4.7 Conclusions

This chapter introduced two methods for the computation of a minimum spanning tree for graph streaming. We have shown empirically that their execution time grows quasi-linearly with image size. Finally, we have shown how to apply these methods to segmentation tasks. In particular, how they can be used to extract a level of λ-quasi-flat-zones hierarchy for image streaming. The main advantage of our proposed algorithm is that at each time the MST is decomposed in two parts. The *stable* part can be stored or used in further tasks. As shown in the case of segmentation. The second one, the *unstable*, is kept in memory. Since it may contain edges that could be removed from MST as future information arrives. Thanks to this decomposition, we can reduce the memory necessary to compute the MST and treat images of bigger sizes.

5 Metric Learning

Resumé

L'une des principales limites des techniques d'apprentissage supervisé est le grand nombre d'étiquettes nécessaires pour former un modèle. Dans de nombreuses applications, il est plus facile de définir des relations entre des paires/triplets d'objets que d'attribuer une étiquette à chaque échantillon. Dans ces cas, il n'est pas possible de détecter des objets en utilisant une perte de classification. L'apprentissage de métriques aborde le problème du regroupement en mappant les données dans un espace d'intégration. Dans cet espace, les objets similaires sont cartographiés près les uns des autres, tandis que les objets dissemblables sont cartographiés loin les uns des autres. Ce chapitre est divisé en deux parties. Nous commençons par une application pratique de l'apprentissage de métriques. Nous nous attaquons au problème proposé comme piste sur la recherche et la classification de surfaces dans le cadre du concours SHREC'20 par des chercheurs de l'IMATI CNR et de l'Université de Vérone. Nous illustrons la solution que nous avons soumise à la compétition, et qui s'est classée en deuxième position. Dans la deuxième partie du chapitre, nous nous concentrons sur l'apprentissage de métriques pour le clustering hiérarchique. Dans la formulation classique du problème, les similarités entre les points sont fixes alors qu'elles représentent un élément clé pour la qualité des résultats. Nous proposons une version étendue du problème d'optimisation continue introduit par Chami et al. (2020) qui vise à trouver à la fois une fonction de similarité optimale entre les points et un clustering hiérarchique optimal.

5.1 Introduction to Metric Learning

A great limitation of supervised learning techniques is given by the great number of labels needed to train a model. In many applications, it is easier to define relationships between pairs/triplets of objects than assign a label to each sample. In these cases, it is not possible to detect objects using a classification loss. Metric Learning addresses the clustering problem by mapping data into an embedding space. In this space, similar objects are mapped close to each other while dissimilar objects are mapped far apart. This chapter is divided in two parts. We start with a practical application of metric learning. Namely, we tackle the problem proposed as a track on retrieval and classification of surfaces in the SHREC'20 competition by researchers of IMATI CNR and University of Verona. We illustrate the solution that we submitted to the competition, and that ranked second place. In the second part of the chapter, we focus on metric learning

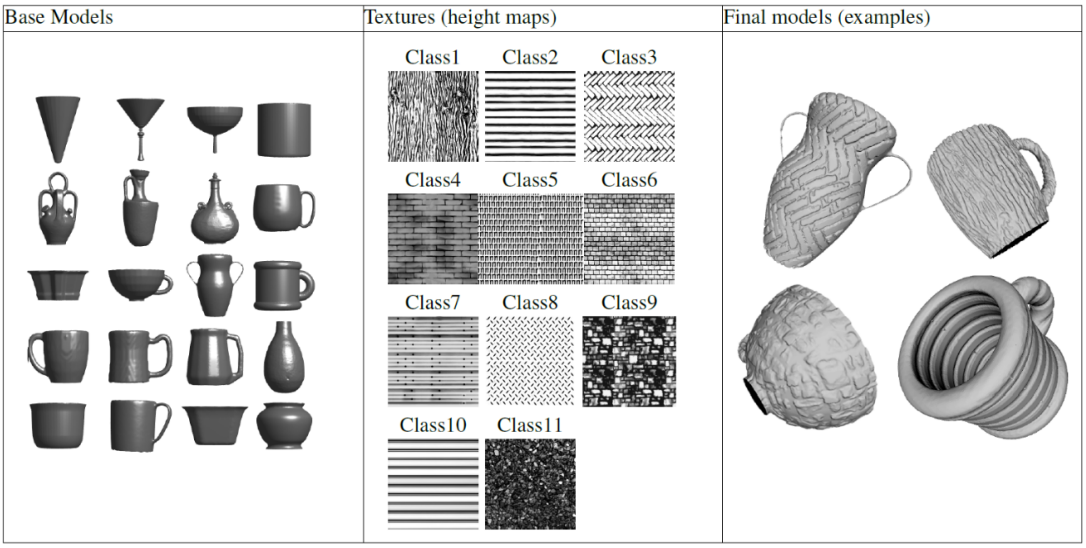


Figure 5.1 Left: Base model on which reliefs are applied. **Center**: the 11 textures used as height-fields on the base model (brighter colours for higher values). **Right:** Some examples of the final models of the dataset. (Image source Moscoso Thompson et al. (2020))

for hierarchical clustering. In the classical formulation of the problem, similarities between points are fixed even though they represent a key element for the quality of the results. We propose an extended version of the continuous optimization problem introduced by Chami et al. (2020) that aims to find at the same time an optimal similarity function between the points and an optimal hierarchical clustering.

5.2 SHREC'20 track: Retrieval and classification of surface patches with similar geometric relief

The aim of this SHREC'20 track was to evaluate the performance of automatic algorithms for the retrieval and classification of relief patterns on the surface of 3D models. The goal was to estimate the similarity between two objects based only on the patterns on their surfaces, without considering their global shape. In the following subsection, we describe the dataset used for the contest and our proposed solution. Please refer to Moscoso Thompson et al. (2020) for the full track report of the challenge.

5.2.1 Dataset

The dataset used is made of 220 mesh surfaces, each one characterized by different relief patterns. The entire dataset has been generated starting from 20 base models that represent several objects like pots, goblets or mugs and from a set of 11 different 3D textures selected from the free dataset Texture Haven[1] as illustrated in Figure 5.1. The resulting surfaces are oriented and the topology of models can contain holes or handles. The challenge is to distinguish different objects from relief of textures rather than the shape of models.

[1] https://texturehaven.com/

5.2.2 Proposed Method

Since the goal is to estimate the similarity between two objects without taking into account the global shape, we considered extracting from each object a set of images containing only the local texture. In this context, each object is represented as a triangle mesh $\mathcal{S} \subset \mathbb{R}^3$. Along with the mesh, we define the graph $\mathcal{G}_\mathcal{S} = (V, E)$ associated to the mesh $\mathcal{S}$, that is the graph whose nodes are the points $\{x_1, \dots x_n\}$ of the mesh, and an edge between the nodes x_i and x_j exists if and only if the two points are the vertices of the same triangle in the mesh. Thus, given a surface $\mathcal{S} \subset \mathbb{R}^3$, our method starts sampling a subset of points $\{x_1, \dots, x_m\} \in \mathcal{S}$ on the surface using Poisson Disk Sampling Yuksel (2015). Then for each point x_i, we use the geodesic distance defined over the graph $\mathcal{G}_\mathcal{S}$ to identify a local neighbourhood. To be more precise, the geodesic distance $d(x_i, x_j)$ between two nodes x_i and x_j of a graph is the length of the shortest path connecting them. Thus, given $r > 0$, the local neighbourhood is defined as $\mathcal{N}_r(x_i) = \{x_j \in V \mid d(x_i, x_j) \leq r\}$. Our goal is to project the local neighbourhood over a plane and obtain an elevation image. For this reason, we aim to select only those neighbours that are as flat as possible, and discard the others. We use covariance-based features to estimate the flatness of the local neighbourhood. Those features are derived from the eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \in \mathbb{R}$ of the neighbourhood covariance matrix defined as:

$$cov(\mathcal{N}_r(x_i)) = \frac{1}{|\mathcal{N}_r(x_i)|} \sum_{x \in \mathcal{N}_r(x_i)} (x - \bar{x})(x - \bar{x})^T,$$

and $\bar{x}$ is the centroid of the neighbourhood $\mathcal{N}_r(x_i)$. The following criteria are used to estimate if the neighbourhood is flat enough:

1. criterion on the planarity: $\frac{\lambda_2 - \lambda_3}{\lambda_1} \geq 0.5$,
2. criterion on the change of curvature: $\frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \leq 0.03$.

The two values have been chosen empirically after some tests over different objects. Once the neighbourhood has been validated by our criteria we project it over the tangent space of the surface at x_i. A regular grid is defined over the tangent space, and each element of the grid corresponds to a pixel of the image. The intensity values of the image correspond to the distance between the point projected over the element and the tangent plane.

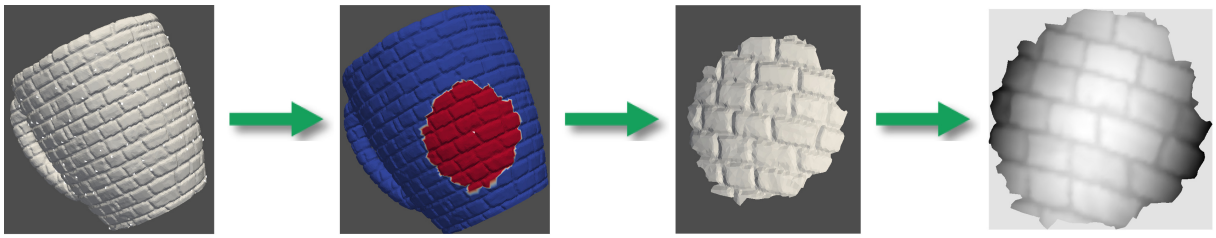


Figure 5.2 The pipeline for extraction of images first select a neighbourhood on a mesh, then check if the neighbourhood satisfies the flatness criteria and finally project it on an image.

The patches generated have different sizes, thus in our experiments we tested two different strategies to uniform all the patch size. The first consists in cropping all the patches to the size of the smallest patch, while the second we pad all to the size of the biggest patch. To conclude, at the end of the first part for each surface $\mathcal{S}_i$ we have obtained a set of images $\{I_1, \dots, I_{m_i}\}$, where $i = 1, \dots, 220$.

Learning the embedding

In the second part, we aim to learn a similarity function between images using a Siamese neural network with the Triplet Loss. The architecture is composed of three CNNs sharing the same weights. In our case as CNN we choose VGG16 Simonyan and Zisserman (2015), without the fully connected layers. The CNNs work in parallel taking as input a triplet of images and generating comparable feature vectors, as shown in Figure 5.3. The Triplet Loss minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity, i.e. an image from a different object Chechik et al. (2010).

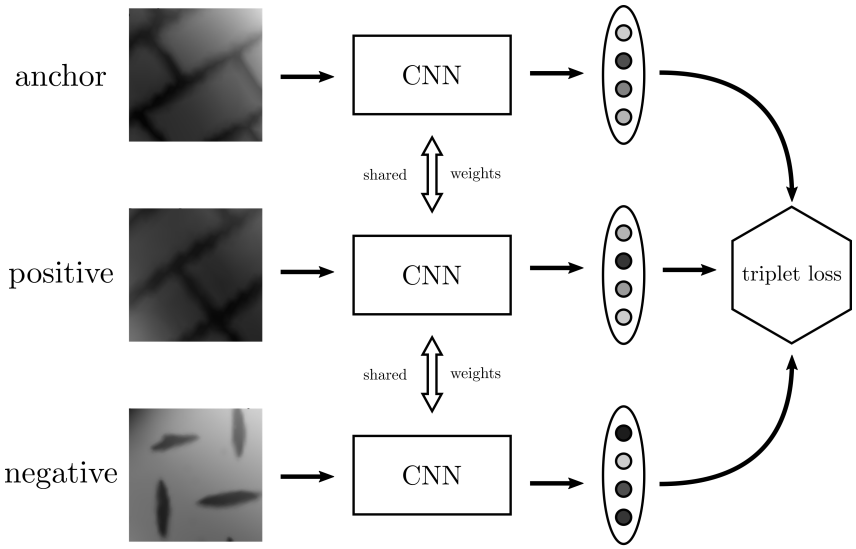


Figure 5.3 Our network consists of a batch input layer and a deep CNN which results in the image embedding by using a triplet loss during training.

Finally we define the distance Δ between two objects $\mathcal{S}_i$ and $\mathcal{S}_j$ as the minimum distance between any couple of images belonging to the two surfaces:

$$\Delta(\mathcal{S}_i, \mathcal{S}_j) = \min_{k,h \in \{1,\dots,m_i\} \times \{1,\dots,m_j\}} \delta(I_h, I_k),$$

where $\delta(I_h, I_k)$ is the similarity function learned by the Siamese neural network.

5.2.3 Metrics

To evaluate the results the following metrics have been used:

- *Nearest Neighbor (NN)*: It measures the percentage of the closest matches that belongs to the same class as the query element. Basically it measures how well a naive nearest neighbor classifier would perform.
- *First Tier (FT) and Second Tier (ST):* These scores measure the percentage of the models in the query's class that appears within the top-K matches. The value of K depends on the size of the query's class. Namely, for the First Tier $K = |C| - 1$, where $|C|$ is the number of elements that belong to the class C, while for the Second Tier $K = 2(|C| - 1)$. The First Tier expresses the recall of the retrieved elements of the same class.

- *Normalized Discounted Cumulative Gain (DCG):* This metric put higher scores to correct results in the top of the list than correct results in the bottom of the ranked list. The idea behind this score is that a user is more likely to consider elements near the front of the list than elements in the end. To the ranked list R is associated a list G, where the element $G_i = 1$ if the element R_i belong to the correct class and 0 otherwise. Discounted Cumulative Gain for query of element q is then defined as follows:

$$DGC_q = \sum_{i=1}^{|R|} \frac{G_i}{\log_2 i + 1}$$

  The value is then normalized with respect to the ideal outcome of that query to obtain the normalized Discounted Cumulated Gain (nDCG) of that query.

- *Precision and Recall* Precision is the fraction of retrieved items that are relevant to the query, or in other words the percentage of objects that belong to the same class of the query element. Recall is the fraction of the relevant documents that are successfully retrieved. Plotting the precision value with respect to the recall we obtain the so-called *precision-recall curve*. This curve gives us a visual understanding of the results. The higher is the integral of this curve the better is our prediction. This integral is called *Area Under Curve* (AUC) and is among the metrics used to evaluate the quality of the results.

- *mAP@k* This metric is mean Average Precision with the number of nearest neighbours for each sample set to k. For a single query is defined as follows:

$$mAP@k = \frac{1}{k} \sum_{i=1}^{k} P(i),$$

  where

$$P(i) = \begin{cases} \textit{precision at } i & \textit{if the ith retrivial is correct}, \\ 0 & \textit{otherwise}. \end{cases}$$

  The number k of neighbours considered is 32.

- e-Measure (e): Similarly as mAP@k e-Measure considers the first k retrieved elements for each query, and it is defined as

$$e = \frac{1}{\frac{1}{P} + \frac{1}{R}},$$

  where P and R are respectively precision and recall values over those results. Also in this case the number of neighbours is fixed to 32.

For the challenge, we submitted two different runs. The main differences between the two runs is the strategy used to uniform the patch images and the use of data augmentation.

In the first run, in fact, we crop all the patches to the size of the smallest patch that is 231×231 pixels. For each patch, crops are computed so that there is the minimum number of void pixels in each image. No data augmentation is used in this run. While in the second run we pad with zeros-values all the patches to the size of the biggest patch that is 836×836. Furthermore, during training we used data augmentation rotating input images with random angles and also flipping vertically and horizon-

tally. We think that both these changes contributed to achieve a better performance as reported in Table 5.1.

Along with our methods, in Table 5.1 we also report the score obtained by the method that won the competition, named Deep Feature Ensamble (DFE). This last method is composed of two steps. In the first step, it extracts images from 3D object transforming the object into a new 3D coordinate system so that the object stands vertically across the y-axis for the ease of rendering and successively extracting the largest inscribed square of the object on the 2D image. After this step, each 3D object has one representing a 2D square image. In the second step, the authors propose an approach similar to ours, using pretrained deep learning models, such as ResNet (He et al., 2016), to extract feature vectors from images. After extracting features, each object it is represented as a feature vector and a metric learning approach it used to optimize the parameters of the network and to compute similarities between objects.

TABLE 5.1
NEAREST NEIGHBOURHOOD, FIRST TIER, SECOND TIER, MAP, NDGC, E-MEASURE AND AUC VALUE OF ALL THE SUBMITTED RUNS. VALUES GO FROM 0, TO 1. THE HIGHER THE VALUE IS, THE BETTER THE METHOD PERFORMS. (SOURCE MOSCOSO THOMPSON ET AL. (2020))

Method	NN	FT	ST	mAP@32	nDCG	e	AUC
Our (run 1)	0.900	0.836	0.990	0.868	0.941	0.686	0.974
Our (run 2)	0.982	0.887	0.992	0.912	0.968	0.690	0.978
DFE (winner)	0.982	0.920	1.000	0.930	0.974	0.715	0.987

5.3 Learning Similarities and Hierarchies

Let now move on to *Similarity-based Hierarchical Clustering*. This is a classical unsupervised learning problem and alternative solutions have been proposed during the years. Hierarchical clustering methods such as Complete Linkage, Ward's Method or λ-flat zones hierarchies that we have introduced in Section 1.2.2 are conceivable solutions to the problem. Dasgupta (2016) firstly formulated this problem as a discrete optimization problem. Given a similarity graph $\mathcal{G}$, that is a graph whose weights represent similarities between vertices of the graph, the author proposes a cost function to evaluate a hierarchical tree T defined over the set of nodes V of the graph as

$$C_G(T) = \sum_{(i,j)\in E} w_{ij}|\text{leaves}(T[i \vee j])|, \tag{5.1}$$

where $i \vee j$ is the Least Common Ancestor (LCA) between leaves nodes i and j in the hierarchical tree T, and $|\text{leaves}(T[i \vee j])|$ is the number of leaves contained in the subtree rooted at $i \vee j$. The intuition behind this formulation is that a good hierarchy merges the most similar points before in the hierarchy and the dissimilar one later on. Successively Wang and Wang (2020) have shown that the cost function can be rewritten as

$$C_G(T) = \sum_{(ijk)\in V^3} [w_{ij} + w_{jk} + w_{ik} - w_{ijk}(T)] + 2 \sum_{(i,j)\in E} w_{ij}, \tag{5.2}$$

where

$$w_{ijk}(T) = w_{ij}\mathbb{1}[\{i,j|k\}] + w_{jk}\mathbb{1}[\{j,k|i\}] + w_{ik}\mathbb{1}[\{i,k|j\}], \tag{5.3}$$

and the relation $\{i,j|k\}$ holds in T if the LCA $i \vee j$ is a proper descendant of the LCA $i \vee j \vee k$. Note that for a binary tree T, only one term in the Equation 5.3 is non-zero. The main issue with this approach is that finding an optimal hierarchy T^* that minimizes cost in the Equation 5.1 is NP-Complex (Dasgupta, 2016, Th. 10). For this reason, several continuous approximations of the Dasgupta's cost function have been proposed in recent years. However, in the formulation of the problem, the input set and the similarities between points are fixed elements. Thus, any change in the input set entails a reinitialization of the problem and a new solution must be found. In addition to this, we underline that the similarity function employed is a key component for the quality of the solution. For these reasons, we are interested in an extended formulation of the problem in which we assume as input a family of point sets, all sampled from a fixed distribution. Our goal is to find at the same time a "good" similarity function on the input space and optimal hierarchical clustering for the point sets. The solution proposed will be validated on a Toy Dataset containing five different kinds of distributions.

5.3.1 Related Works

Our work is inspired by Chierchia and Perret (2020) and Chami et al. (2020) where for the first time continuous frameworks for hierarchical clustering have been proposed. Both papers assume that a given weighted-graph $\mathcal{G} = (V, E, W)$ is given as input. The authors of the first paper aim to find the best ultrametric that optimizes a given cost function. Basically, they exploit the fact that the set $\mathcal{W} = \{w : E \to \mathbb{R}_+\}$ of all possible functions over the edges of $\mathcal{G}$ is isomorphic to the euclidean subset $\mathbb{R}_+^{|E|}$ and thus it makes sense to "*take a derivative according to a given weight function*". Along with this, they show that the min-max operator function $\Phi_{\mathcal{G}} : \mathcal{W} \to \mathcal{W}$, defined as

$$(\forall \tilde{w} \in \mathcal{W}, \forall e_{xy} \in E) \quad \Phi_{\mathcal{G}}(\tilde{w})(e_{xy}) = \min_{\pi \in \Pi xy} \max_{e' \in \pi} \tilde{w}(e'), \tag{5.4}$$

where Π_{xy} is the set of all paths from vertex x to vertex y, is sub-differentiable. As insight, the min-max operator basically maps any function $w \in \mathcal{W}$ to its associated subdominant ultrametric on $\mathcal{G}$. These two key elements are combined to define the following minimization problem over $\mathcal{W}$

$$w^* = \arg\min_{\tilde{w} \in \mathcal{W}} J(\Phi_{\mathcal{G}}(\tilde{w}), w), \tag{5.5}$$

where the function J is a differentiable loss function to optimize. In particular, since the metrics $\tilde{w}$ are indeed vectors of $\mathbb{R}^{|E|}$, the authors propose to use the L^2 distance as a natural loss function. Furthermore, they come up with other regularization functions, such as a cluster-size regularization, a triplet loss, or a new differentiable relaxation of the famous Dasgupta's cost function Dasgupta (2016). The contribution of Chami et al. (2020) is twofold. On the one hand, inspired by the work of Monath et al. (2019), they propose to find an optimal embedding of the graph nodes into the Poincaré Disk, observing that the internal structure of the hierarchical tree can be inferred from leaves' hyperbolic embeddings. On the other hand, they propose a direct differentiable relaxation of the Dasgupta's cost and prove theoretical guarantees in terms of clustering quality of the optimal solution for their proposed function compared with the optimal hierarchy for the Dasgupta's function.

As said before, both approaches assume a dataset $\mathcal{D}$ containing n data points and pairwise similarities $(w_{ij})_{i,j\in[n]}$ between points are known in advance. Even though this is a very general hypothesis, unfortunately, it does not include cases where part of the data is unknown yet or the number of points cannot be estimated in advance, as for example point-cloud scans. In the following section, we investigate a way to extend the works above to our case. To be specific, we cannot assume any more that a given graph is known in advance, and thus we cannot work on the earlier defined set of functions $\mathcal{W}$, but we would rather look for optimal embeddings of the node features. Before describing the problem, let us review hyperbolic geometry and hyperbolic hierarchical clustering.

5.3.2 Hyperbolic Hierarchical Clustering

The Poincaré Ball Model $(\mathbb{B}^n, g^{\mathbb{B}})$ is a particular hyperbolic space, defined by the manifold $\mathbb{B}^n = \{x \in \mathbb{R}^n \mid \|x\| < 1\}$ equipped with the following Riemannian metric,

$$g_x^{\mathbb{B}} = \lambda_x^2 g^E, \quad \text{where } \lambda_x := \frac{2}{1-\|x\|^2},$$

where $g^E = \mathbf{I}_n$ is the Euclidean metric tensor. The distance between two points in the $x, y \in \mathbb{B}^n$ is given by,

$$d_{\mathbb{B}}(x,y) = \cosh^{-1}\left(1 + 2\frac{\|x-y\|_2^2}{(1-\|x\|_2^2)(1-\|y\|_2^2)}\right). \tag{5.6}$$

It is thus straightforward to prove that the distance of a point to the origin is $d_o(x) := d(o,x) = 2\tanh^{-1}(\|x\|_2)$. Finally, we remark that $g_x^{\mathbb{B}}$ defines the same angles as the euclidean metric. The angle between two vectors $u, v \in T_x\mathbb{B}^n \setminus \{\mathbf{0}\}$ is defined as

$$\cos(\angle(u,v)) = \frac{g_x^{\mathbb{B}}(u,v)}{\sqrt{g_x^{\mathbb{B}}(u,u)}\sqrt{g_x^{\mathbb{B}}(v,v)}} = \frac{\langle u, v\rangle}{\|u\|\|v\|},$$

and $g^{\mathbb{B}}$ is said to be *conformal* to the Euclidean metric. In our case, we are going to work on the Poincaré Disk, that is $n = 2$. The interested reader may refer to (Brannan et al., 2012, Chapter 6) for a wider discussion on Hyperbolic Geometry. Please remark that the geodesic between two points in this metric is either segments of circles orthogonal to the boundary of the ball or straight lines that go through the origin, as shown in Figure 5.4.

The intuition behind the choice of this particular space is motivated by the fact that the curvature of the space is negative and geodesic coming out from a point has a *"tree-like"* shape (see Figure 5.5). Moreover, Chami et al. (2020) proposed an analogy of the Least Common Ancestor (LCA) in the hyperbolic space. Given two leaf nodes i, j of a hierarchical T, the LCA $i \vee j$ is the closest node to the root r of T on the shortest path π_{ij} connecting i and j. In other words,

$$i \vee j = \arg\min_{k\in\pi_{ij}} d_T(r,k),$$

where $d_T(r,k)$ measures the length of the path from the root node r to the node k. Similarly, the *hyperbolic lowest common ancestor* between two points z_i and z_j in the hyperbolic space is defined as the closest

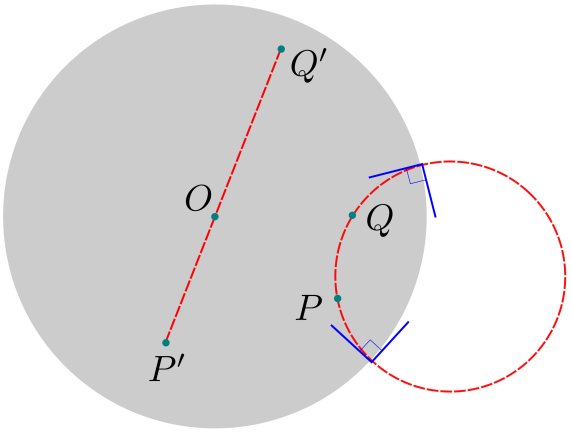


Figure 5.4 Geodesics on the Poincaré Disk are either segments of circles orthogonal to the boundary of the ball as for the case of points P and Q or straight lines passing through the origin as for the case of points P' and Q'.

point to the origin in the geodesic path, denoted $z_i \rightsquigarrow z_j$, connecting the two points:

$$z_i \vee z_j \coloneqq \arg\min_{z \in z_i \rightsquigarrow z_j} d(o, z).$$

Thanks to this definition, it is possible to decode a hierarchical tree starting from leaf nodes embedding into the hyperbolic space. The decoding algorithm uses a union-find paradigm, iteratively merging the most similar pairs of nodes based on their hyperbolic LCA distance to the origin. Finally, Chami et al. (2020) also proposed a continuous version of Dasgupta's cost function. Let $Z = \{z_1, \dots, z_n\} \subset \mathbb{B}^2$ be an embedding of a tree T with n leaves, they define their cost function as:

$$C_{\text{HYPHC}}(Z; w, t) = \sum_{ijk} (w_{ij} + w_{ik} + w_{jk} - w_{\text{HYPHC},ijk}(Z; w, t)) + \sum_{ij} w_{ij}, \tag{5.7}$$

where

$$w_{\text{HYPHC},ijk}(Z; w, t) = (w_{ij}, w_{ik}, w_{jk}) \cdot \sigma_\tau(d_o(z_i \vee z_j), d_o(z_i \vee z_k), d_o(z_j \vee z_k))^\top,$$

and $\sigma_\tau(\cdot)$ is the scaled softmax function $\sigma_\tau(w)_i = e^{w_i/\tau} / \sum_j e^{w_j/\tau}$. Note that this formulation is similar to the one proposed by Wang and Wang (2020) and reported in Equation 5.3.

5.3.3 Learning Similarities

To introduce our problem, we consider the example of k continuous random variables that take values over an open set $\Omega \subset \mathbb{R}^d$. Let $\mathcal{X}_t = \{x_1^{(t)}, \dots, x_{n_t}^{(t)}\}$ a set of points obtained as realization of the k random variables at step t. Moreover, we assume to be in a semi-supervised setting. Without loss of generality, we expect to know the associated labels of the first l points in $\mathcal{X}_t$, for each t. Each label takes value in $[k] = \{1, \dots, k\}$, and indicates from which random variable the point has been sampled. In our work, we aim to obtain at the same time a good similarity function $\delta : \Omega \times \Omega \to \mathbb{R}_+$ that permits us to discriminate the points according to the distribution they have been drawn and an optimal hierarchical clustering for

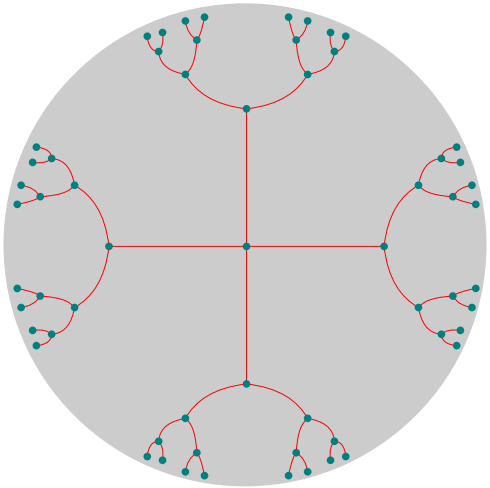


Figure 5.5 A sketch of a tree embedded in the Poincaré disk. Red curves connecting the dots are *geodesics* of the space.

each set $\mathcal{X}_t$. Our idea to achieve this goal, is to combine the continuous optimization framework proposed by Chami et al. (2020) along with deep metric learning to learn the similarities between points. Hence, we look for a similarity function $\delta_\theta : \Omega \times \Omega \to \mathbb{R}_+$ such that

$$\min_{\theta, Z \in \mathcal{Z}} C_{\text{HYPHC}}(Z; \delta_\theta, \tau) + \mathcal{L}_{\text{triplet}}(\delta_\theta; \alpha). \tag{5.8}$$

The second term of the equation above is the margin triplet loss defined as

$$\mathcal{L}_{\text{triplet}}(\delta_\theta; \alpha) = \sum_i \max(\delta_\theta(a_i, p_i) - \delta_\theta(a_i, n_i) + \alpha, 0), \tag{5.9}$$

where a_i is the anchor input, p_i is the positive input of the same class as a_i, n_i is the negative input of a different class from a_i and $\alpha > 0$ is the margin between positive and negative values.

5.3.4 Model Architecture

As explained before, we aim to learn a similarity function and at the same time find an optimal embedding for a family of point sets into the hyperbolic space, which implicitly encodes a hierarchical structure. To achieve this, our idea is to model the function δ_θ using a neural network whose parameters we fit in order to optimize the loss function defined in Equation (5.8). Our implementation consists of a neural network NN_θ that carries out a mapping $\text{NN}_\theta : \Omega \to \mathbb{R}^2$. The similarity function δ_θ is thus written as

$$\delta_\theta^c(x, y) = \cos(\angle(\text{NN}_\theta(x), \text{NN}_\theta(y))), \tag{5.10}$$

We use the cosine similarity for two reasons. The first comes from the intuition that points belonging to the same cluster will be forced to have small angles between them. As a consequence, they will be merged earlier in the hierarchy. The second reason regards the optimization process. Since the hyperbolic metric is *conformal* to the Euclidean metric, the cosine similarity allows us to use the same the RAdam (Riemannian Adam) optimizer (Bécigneul and Ganea, 2019) in Equation (5.8). Once computed the similarities, the points are all normalized at the same length to embed them into the Hyperbolic space.

The normalization length is also a trainable parameter of the model. In other words, to map the hidden features $z = NN_\theta(x) \in \mathbb{R}^2$ to $\mathbb{B}^2$, we do $\frac{\varphi z}{\|z\|}$, where φ is a trainable parameter initialized at $\varphi = 0.001$. A wider family of solutions to the problem are architectures made by two components. The first part acts as feature extraction $\mathrm{NN}^f_\theta : \Omega \to \mathcal{H}$. A natural choice would be $\mathcal{H} \subset \mathbb{R}^d$, where d is the dimension of the feature space. In this hidden space, we compute similarities between points similarly to Equation (5.10). The second part $\mathrm{NN}^e_\theta : \mathcal{H} \to \mathbb{B}^2$ aims to embed from the hidden space to the Hyperbolic Space. The idea behind this second approach is to disentangle the two components in Equation (5.8) and to optimize each loss in separated spaces. The implementation of this second solution is beyond the purpose of this thesis. We conclude this section discussing the properties that we need for our neural network. The list of characteristics will guide us during the choice of the model. Similarly to PointNet Charles et al. (2017), a basic property we would like the model to have is to be independent of the number of points in the sample and to be invariant to permutations. Furthermore, we would like the model to be robust to noise.

We have selected two architectures. The first is a Multi-Layer-Perceptron (MLP) because it has the basic properties, while the second uses Dynamic-Edge-Convolutions proposed by Wang et al. (2019b). In particular we have chosen:

1. MLP composed of four hidden layers,
2. DGCNN, a model composed of three layers of DynamicEgdeConv.

Ideally, the perfect model should also be invariant to any rigid action applied to the points in the space, such as translations or rotations, but we leave this issue for future works.

5.4 Experiments

5.4.1 Toy Datasets

In this section we report our experiments on synthetic datasets generated using Python Scikit-Learn Library Pedregosa et al. (2011). Inspired by the work of Chierchia and Perret (2020) we took into account five samples generators in Scikit-Learn to produce the following five datasets.

Circles

In each sample of this dataset the points are aligned along two concentric circles centered at the origin. It is possible to control the ratio between the two circumferences and the noise of the distributions. In Figure 5.6 we illustrate different samples obtained changing the value of the noise, while the ratio between the radii of the two circles is fixed.

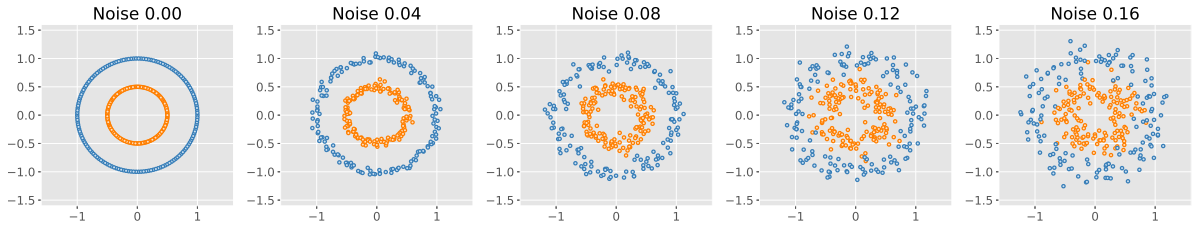


Figure 5.6 Example of circles that can be generated varying noise value.

Moons

The moons dataset is similar to the circle case. This time the points are aligned along four different moon shaped forms. Also in this case we can add noise in the sampling. Figure 5.7 shows different kinds of realisation that we obtain by changing the value of noise.

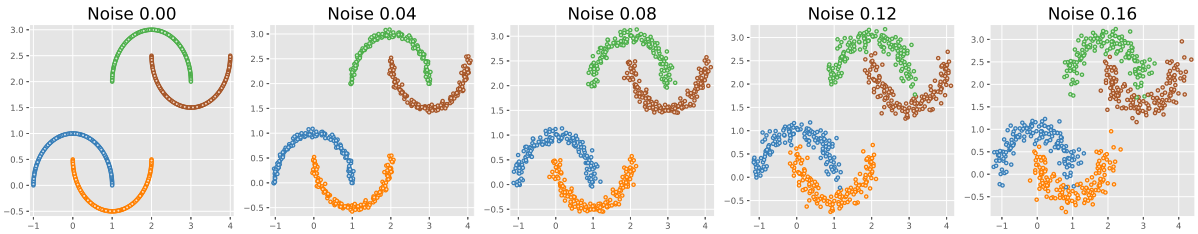


Figure 5.7 Different samples of moons obtained increasing noise value.

Blobs

In this dataset points are drawn from nine different Gaussians distributions whose centers are placed on a regular two-dimensional grid over the square $[-1,1] \times [-1,1] \subset \mathbb{R}^2$ having all the same isotropic variance as illustrated in the bottom right plot of Figure 5.8.

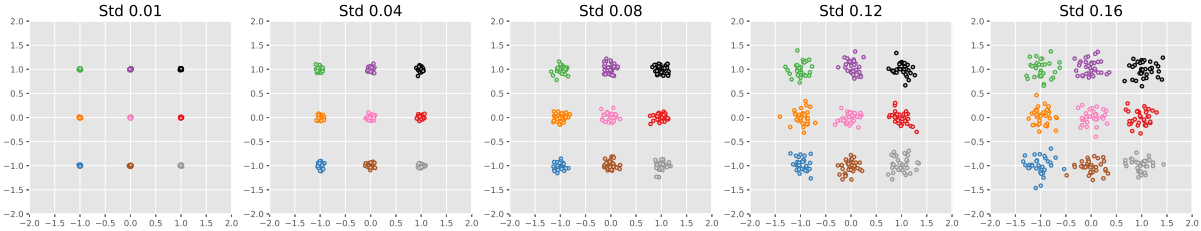


Figure 5.8 Blob Dataset is generated sampling points from nine different Gaussians centered on a regular grid having all the same standard deviation. Figure shows Gaussians that we obtain using different values for standard deviation.

Anisotropic

The anisotropic dataset is similar to blob dataset, but we replace isotropic variances with anisotropic variances, see Figure 5.9. The anisotropy has been obtained applying a rigid transformation of the plane.

Varied

Similar as blobs dataset but in this case, each Gaussian has a different value for standard deviation as illustrated in Figure 5.10. During the implementation of this dataset we decided to fix the values of the standard deviations. This means that in this case the Gaussian distributions are fixed.

5.4.2 Results on Toy Datasets

Architectures: The architectures we test are MLP and DGCNN. The dimension of hidden layers is 64. After each Linear layer, we apply a LeakyReLU defined as $\text{LeakyReLU}(x) = \max\{x, \eta x\}$ with a

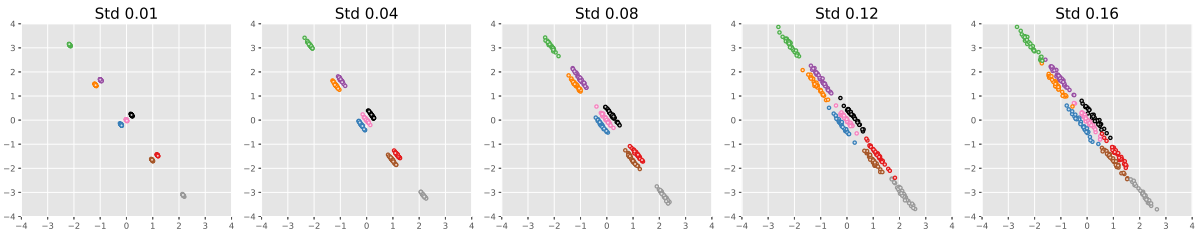


Figure 5.9 Anisotropic Dataset is generated sampling points from nine different anisotropic Gaussians centered on a regular grid. The samples shown have been obtained using different values for standard deviation.

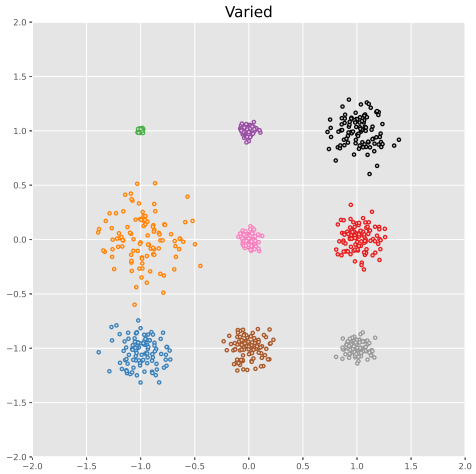


Figure 5.10 Varied Dataset is generated sampling points from nine different Gaussians centered on a regular grid. Gaussians used in this case have different standard deviation.

negative slope $\eta = 0.2$. In addition, we use Batch Normalization (Ioffe and Szegedy, 2015) to speed up and stabilize convergence.

Experiments setup: For each dataset we generate, the training set is made of 100 samples and the validation set of 20 samples. The test set contains 200 samples. In addition, each sample in the datasets contains a random number of points that may vary from 200 to 300 and the labels are known only for the 30% of the points in each sample. In Circle and Moons datasets increasing the value of noise makes the clusters mix with each other and thus the task of detection becomes more difficult. Similarly, in Blobs and Anisotropic we can increase the value of standard deviation to make the problem harder. Finally, for Varied dataset, we train and test the models because the Gaussians are fixed. Our goal is to explore how the models embed the points and separate the clusters. Moreover, we want to investigate the robustness of the models to noise. For this reason, in these datasets, we set up two different levels of difficulty according to the noise/standard deviation used to generate the sample as summarized in Table 5.2. In Circles and Moons datasets, the easiest level is represented by samples without noise, while the harder level contains samples whose noise value varies up to 0.16. In Blobs and Anisotropic datasets, we chose two different values of Gaussian standard deviation to generate the sample. In the easiest level, the standard deviation value is fixed at 0.08, while in the harder level is at 0.16. For each level of difficulty, we trained the models and compared the architectures. In addition, we used the harder level of difficulty to test all the models.

TABLE 5.2
TWO LEVELS OF DIFFICULTY USED TO GENERATE THE DATASETS. IN THE EASIER LEVEL, THE NOISE USED IS LOWER THAN IN THE HARDER.

Level of Difficulty	Dataset	Noise/Std used to generate samples
Easy	Moons	0.0
	Circles	0.0
	Blobs	0.08
	Anisotropic	0.08
Hard	Circles	$0.0 - 0.16$
	Moons	$0.0 - 0.16$
	Blobs	0.16
	Anisotropic	0.16

Metrics: Let k the number of clusters that we want to determine. For the evaluation, we consider the flat cut of the hierarchy that leads to k, and we measure the quality of the predictions using Average Rand Index (ARI), Purity, and Normalized Mutual Information Score (NMI). We have introduced these metrics previously in Section 1.2.3. In addition, we also navigate the hierarchy to find the number of clusters that maximize the Average Rand Index. Our goal is to test the ability of the two types of architectures selected to approximate the similarity function in Equation (5.10).

Visualize the embeddings: Let first discuss the results obtained on Circles and Moons. In order to understand and visualize how similarities are learned, we first trained the architectures at the easiest level of Table 5.2. Figures 5.11, 5.12, 5.14 and 5.15 illustrate the predictions carried out by models trained using samples without noise. Each row in the figures illustrates the model's prediction on a sample generated with a specific noise value. The second column from the left of sub-figures depicts hidden features in the feature space $\mathcal{H} \subset \mathbb{R}^2$. The color assigned to hidden features depends on points' labels in the ground truth. The embeddings in the Poincaré Disk (third column from the left) are obtained by normalizing the features to a learned scale as explained in Section 5.3.4. Furthermore, the fourth column of sub-figures shows the prediction obtained by extracting flat clustering from the hierarchy decoded from leaves embedding in the Poincaré Disk. Here, colors assigned to points come from predicted labels. The number of clusters is chosen in order to maximize the average rand index score. It is interesting to remark how differently the two architectures extract features. Looking at the samples without noise, it is straightforward that hidden features obtained with MLP are aligned along lines passing through the origin. Especially in the case of Circles (Figure 5.11), hidden features belonging to different clusters are mapped to opposite sides with respect to the origin, and after rescaling hidden features are clearly separated in the hyperbolic space. Indeed, picking cosine similarity in Equation 5.10 we were expecting this kind of solution. On the other hand, the more noise we add, the closer to the origin hidden features are mapped. This leads to a less clear separation of points on the disk. Unfortunately, we cannot find a clear interpretation of how DGCNN maps points to hidden space. However also in this case, the more noise we add, the harder it is to discriminate between points of different clusters.
During the experiments on Blobs and Anisotropic datasets, we conducted a similar analysis, in order to grasp how the models extract the hidden features. Also in this case, we first trained models at the easiest

level of Table 5.2, and tested on different values of noise. Figures 5.17, 5.18, 5.20 and 5.20 show how MLP and DGCNN embed points. Similarly, Figure 5.23 shows the behaviour of the two models on a sample in Varied dataset. This time, it is fascinating to see that both models act similarly. Indeed, in all three cases, the two models align the hidden features along directions coming out from the origin. Additionally, as we increase the standard deviation MLP pushes hidden features towards the origin, likewise, it does in Moons and Circle datasets. As before this causes the clusters to overlap on the Poincaré Disk leading to a lesser precise separation between diverse groups. Another insight to underline is that the margin value α in Triplet Loss (5.9) plays an important role in the good convergence of the models. The shown results have been achieved for $\alpha = 0.2$. In Figure 5.24, we illustrate some predictions obtained MLP models trained with different values of α. In those cases, we can clearly perceive how models make some clusters align along the same direction in the hidden space. The value $\alpha = 0.2$ has been chosen after several tests, trying to avoid the alignment effect in the hidden space.

Comparison with classical HC methods: In Figures 5.13, 5.16, 5.19 and 5.22 we compare models trained at easier level of Table 5.2 against classical methods such as Single/Complete/Average Linkage and Ward's method on Circles, Moons, Blobs and Anisotropic respectively. We report scores obtained at different values of noise. For each level of noise, we generate a test set containing 20 samples. The lines in the plots represent the average score obtained by the methods, while the bars represent scores' standard deviation. The plots show the degradation of the performance of models as we add noise to samples.

Results on Circles say that Single Linkage is the method that performs the best for small values of noise. However, MLP shows better robustness to noise. For high levels of noise, MLP is the best method. On the other hand, DGCNN exhibits a low efficacy also on low levels of noise. Other classical methods do not achieve good scores on this dataset. A similar trend can also be observed in Moon dataset. Note that, in this case, MLP is comparable with Single Linkage also on small values of noise, and its scores remain good also on higher levels of noise. DGCNN and other classical methods perform worse even in this data set. Results obtained by MLP and DGCNN on Blobs' dataset are comparable with the classical methods, even though the performances of models are slightly worse compared to classical methods for higher values of noise. On the contrary, MLP and DGCNN achieve better scores on Anisotropic dataset compared to all classical models. Overall, MLP models seem to act better than DGCNN ones in all the datasets.

Benchmark of the models: Table 5.3 reports the scores obtained by the trained models on each dataset. Each line corresponds to a model trained either at an easier or harder level of difficulty of Table 5.2. The test set used to evaluate the results contains 200 samples generated using the harder level of difficulty. Scores obtained demonstrate that models trained at the harder levels of difficulty are more robust to noise and achieve better results. As before, also in this case MLP is, in general, better than DGCNN in all the datasets considered.

TABLE 5.3
SCORES OBTAINED BY MLP AND DGCNN ON FIVE TOY DATASETS: CIRCLES, MOONS, BLOBS, ANISOTROPIC, VARIED. IN EACH DATASET THE MODELS HAVE BEEN TESTED ON THE SAME TEST SET CONTAINING 200 SAMPLES. IN CIRCLES AND MOONS, DATASET SAMPLES IN THE TEST SET HAVE BEEN GENERATED USING A NOISE VALUES THAT VARIES FROM 0.0 UP TO 0.16. IN BLOBS AND ANISOTROPIC DATASETS SAMPLES IN THE TEST SET HAVE BEEN PRODUCED FIXING TO 0.16 THE VALUE OF STANDARD DEVIATION FOR GAUSSIAN DISTRIBUTIONS. TO GENERATE A TEST SET FOR VARIED DATASET WE KEPT THE SAME STANDARD DEVIATION FOR GAUSSIANS AS TRAIN AND VALIDATION SET.

Dataset	k	Noise/Cluster std	Model	Hidden	Temp	Margin	Ari@k $\pm$ s.d	Purity@k $\pm$ s.d	Nmi@k $\pm$ s.d	Ari $\pm$ s.d
Circles	2	0.0	MLP	64	0.1	1.0	0.871 ± 0.153	0.965 ± 0.0427	0.846 ± 0.167	0.896 ± 0.123
Circles	2	$[0.0 - 0.16]$	MLP	64	0.1	1.0	$\mathbf{0.919 \pm 0.18}$	$\mathbf{0.972 \pm 0.0848}$	$\mathbf{0.895 \pm 0.187}$	$\mathbf{0.948 \pm 0.0755}$
Circles	2	0.0	DGCNN	64	0.1	1.0	0.296 ± 0.388	0.699 ± 0.188	0.327 ± 0.356	0.408 ± 0.362
Circles	2	$[0.0 - 0.16]$	DGCNN	64	0.1	1.0	0.852 ± 0.243	0.947 ± 0.116	0.826 ± 0.247	0.9 ± 0.115
Moons	4	0.0	MLP	64	0.1	1.0	0.895 ± 0.137	0.927 ± 0.108	0.934 ± 0.0805	0.955 ± 0.0656
Moons	4	$[0.0 - 0.16]$	MLP	64	0.1	1.0	$\mathbf{0.96 \pm 0.0901}$	$\mathbf{0.971 \pm 0.0751}$	$\mathbf{0.972 \pm 0.049}$	$\mathbf{0.989 \pm 0.017}$
Moons	4	0.0	DGCNN	64	0.1	1.0	0.718 ± 0.247	0.807 ± 0.187	0.786 ± 0.191	0.807 ± 0.172
Moons	4	$[0.0 - 0.16]$	DGCNN	64	0.1	1.0	0.917 ± 0.123	0.942 ± 0.0992	0.941 ± 0.0726	0.966 ± 0.0455
Blobs	9	0.08	MLP	64	0.1	0.2	0.911 ± 0.069	0.939 ± 0.057	0.953 ± 0.025	0.958 ± 0.022
Blobs	9	0.16	MLP	64	0.1	0.2	$\mathbf{0.985 \pm 0.0246}$	$\mathbf{0.992 \pm 0.0198}$	$\mathbf{0.99 \pm 0.0115}$	$\mathbf{0.992 \pm 0.00821}$
Blobs	9	0.08	DGCNN	64	0.1	0.2	0.856 ± 0.0634	0.891 ± 0.0583	0.931 ± 0.025	0.921 ± 0.0401
Blobs	9	0.16	DGCNN	64	0.1	0.2	0.894 ± 0.0694	0.92 ± 0.0604	0.95 ± 0.0255	0.948 ± 0.0336
Aniso	9	0.08	MLP	64	0.1	0.2	0.86 ± 0.0696	0.904 ± 0.0631	0.922 ± 0.0291	0.925 ± 0.0287
Aniso	9	0.16	MLP	64	0.1	0.2	$\mathbf{0.952 \pm 0.0503}$	$\mathbf{0.968 \pm 0.044}$	$\mathbf{0.972 \pm 0.0189}$	$\mathbf{0.976 \pm 0.0133}$
Aniso	9	0.08	DGCNN	64	0.1	0.2	0.713 ± 0.0835	0.793 ± 0.0727	0.844 ± 0.0401	0.795 ± 0.0652
Aniso	9	0.16	DGCNN	64	0.1	0.2	0.84 ± 0.0666	0.879 ± 0.0595	0.922 ± 0.0274	0.914 ± 0.0436
Varied	9	-	MLP	64	0.1	0.2	$\mathbf{0.997 \pm 0.0057}$	$\mathbf{0.999 \pm 0.00252}$	$\mathbf{0.998 \pm 0.00448}$	$\mathbf{0.998 \pm 0.00365}$
Varied	9	-	DGCNN	64	0.1	0.2	0.98 ± 0.0335	0.986 ± 0.0298	0.989 ± 0.0122	0.991 ± 0.0102

5.5 Conclusions

In this chapter, we have studied the metric learning problem for the retrieval of 3D textured shapes and to perform hierarchical clustering where the number of nodes per graph in the training set can vary. In the first case, an original approach is proposed and compared against different methods. Our contribution finished in the Top-3 over twenty solutions proposed by eight different teams in the international Shrec'20 challenge. In the second part, we have trained MLP and DGCNN architectures on five Toy datasets by using our proposed protocol. The key point of the proposed framework is its ability to work on samples with a varying number of points. The quantitative results show that overall MLP performs better than DGCNN. The comparison with the classic methods proves the flexibility of the solution proposed to the different cases analyzed, and the results obtained confirm higher robustness to noise. Finally, inspecting the hidden features, we have perceived how MLP tends to project points along lines coming out from the origin. To conclude, the results obtained are promising and we believe that it is worth testing this solution also on other types of datasets such as 3D point clouds.

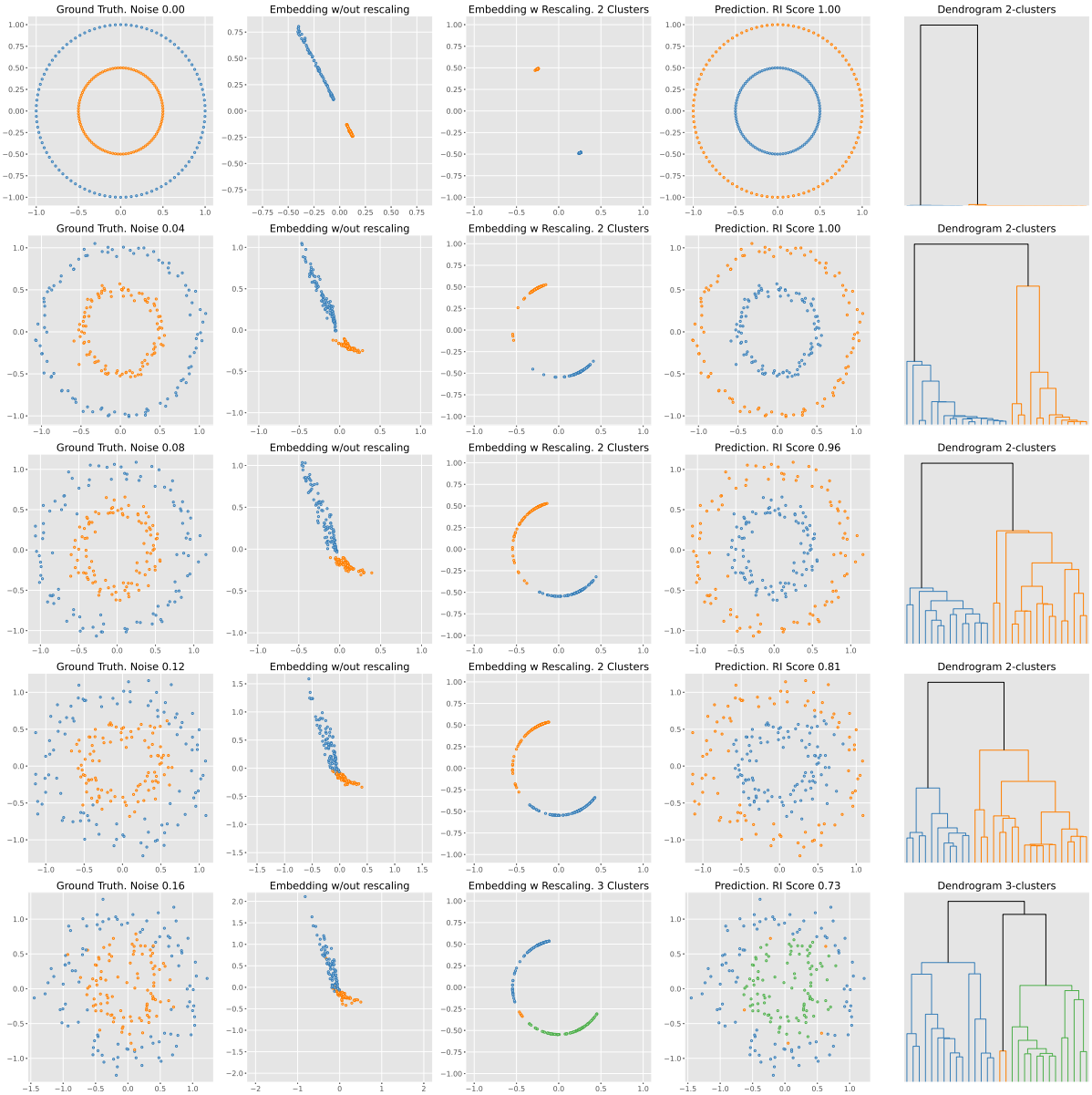


Figure 5.11 Effect of noise on predictions in the circle database. The model used for prediction is a MLP trained using a dataset without noise. From top to bottom, each row is a case with an increasing level of noise. In the first column the input points, while in the second column we illustrate hidden features. Points are colored according to ground truth. The third column illustrates the hidden features after projection to Poincaré Disk. Fourth column shows predicted labels, while the fifth column shows associated dendrograms. Colors in the last three columns are assigned according to predicted labels.

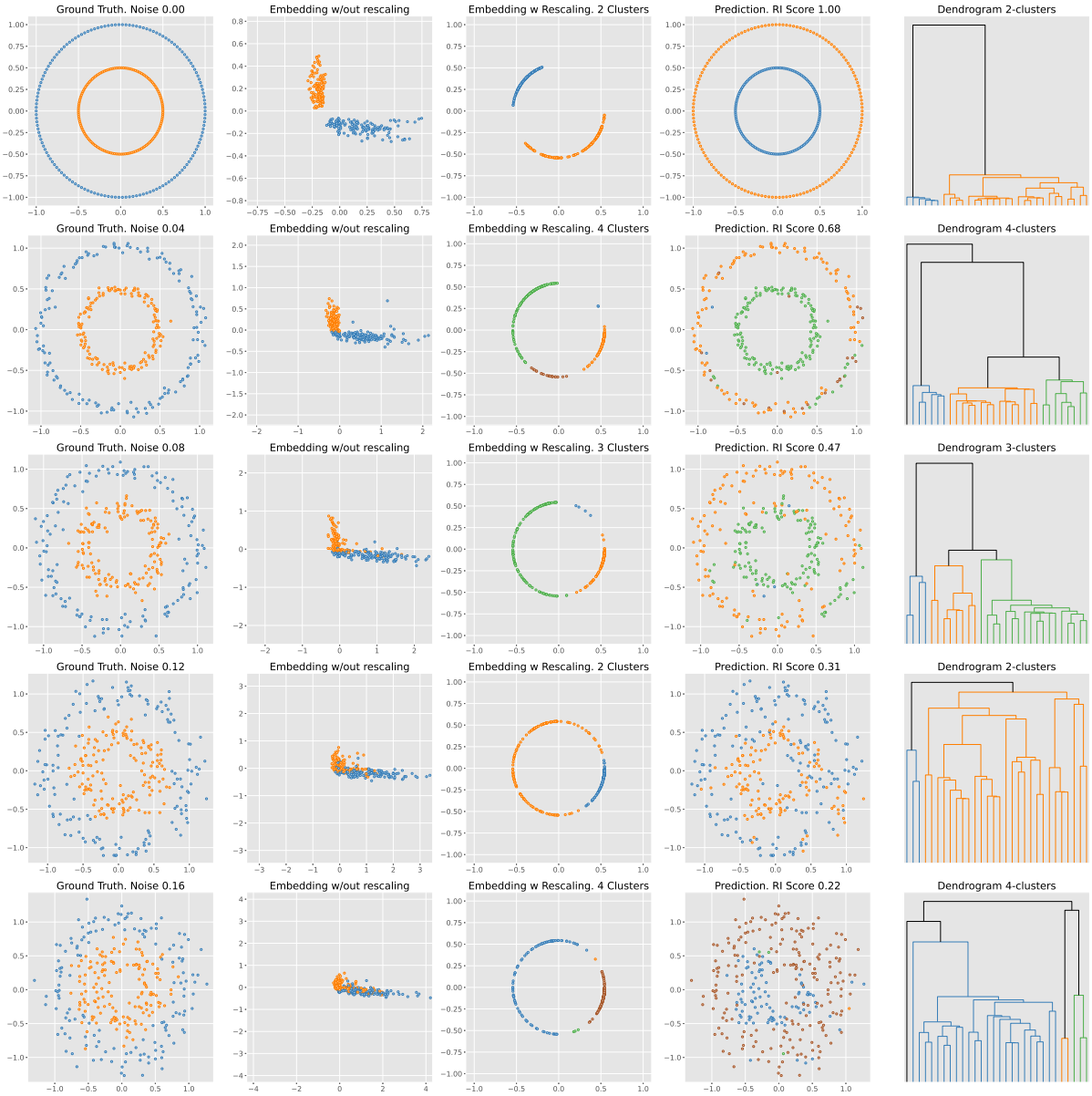
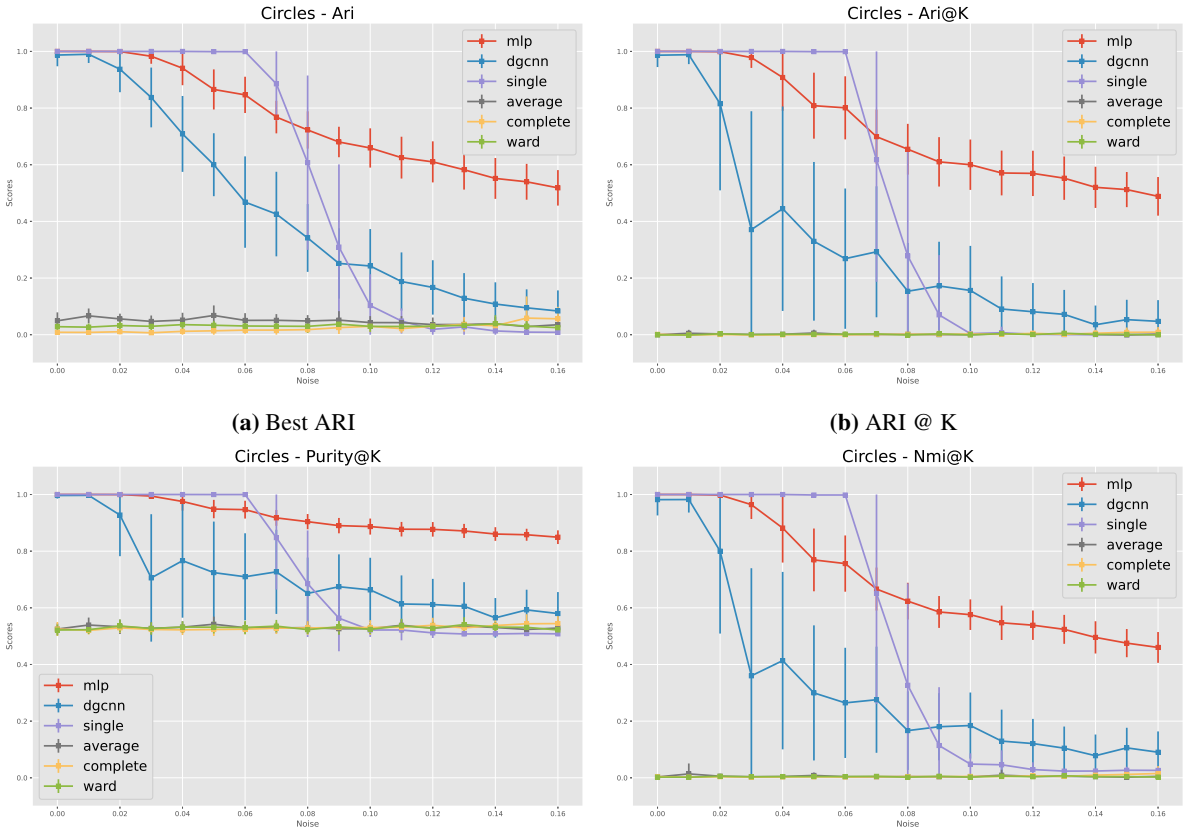


Figure 5.12 Effect of noise on predictions in circle database. The model used for prediction is a DGCNN trained using a dataset without noise. From top to bottom, each row is a case with an increasing level of noise. In the first column the input points, while in the second column we illustrate hidden features. Points are colored according to ground truth. The third column illustrates the hidden features after projection to Poincaré Disk. Fourth column shows predicted labels, while the fifth column shows associated dendrograms. Colors in the last three columns are assigned according to predicted labels.



(a) Best ARI

(b) ARI @ K

(c) Purity @ K

(d) NMI @ K

Figure 5.13 Robustness to noise of models on Circles. We compare trained models against classical methods as Single Linkage, Average Linkage, Complete Linkage and Ward's Method. The models used have been trained on a dataset without noise. Test sets used to measure scores contain 20 samples each. Plots show mean and standard deviation of scores obtained. During the experiments on this dataset MLP has shown a higher robustness to noise compared with DGCNN. Among classical methods only single linkage perform well on these samples.

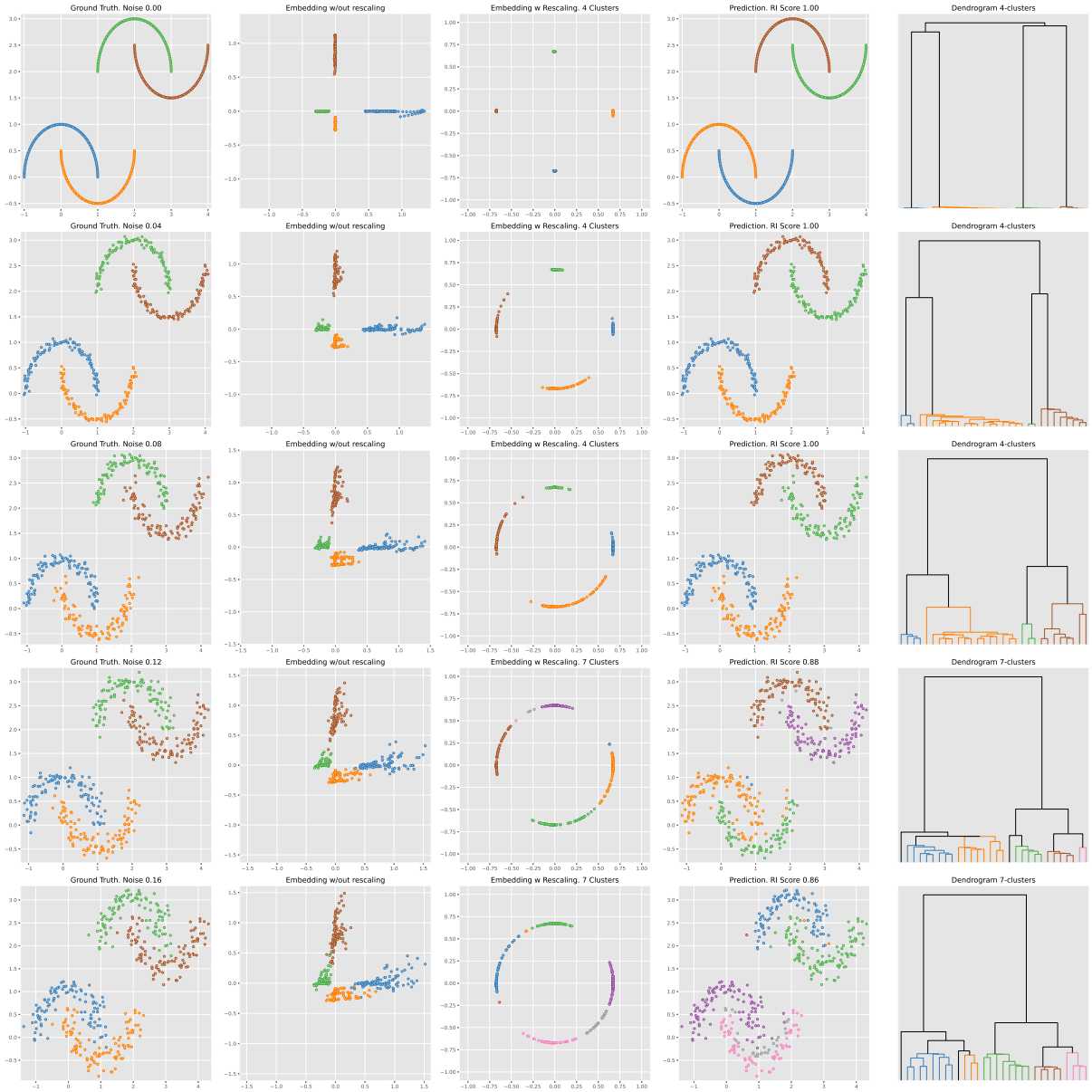


Figure 5.14 Moons dataset. The model used is a MLP trained on samples without noise. From top to bottom, each row is a case with an increasing level of noise. In the first column the input points, while in the second column we illustrate hidden features. Points are colored according to ground truth. The third column illustrates the hidden features after projection to Poincaré Disk. Fourth column shows predicted labels, while the fifth column shows associated dendrograms. Colors in the last three columns are assigned according to predicted labels.

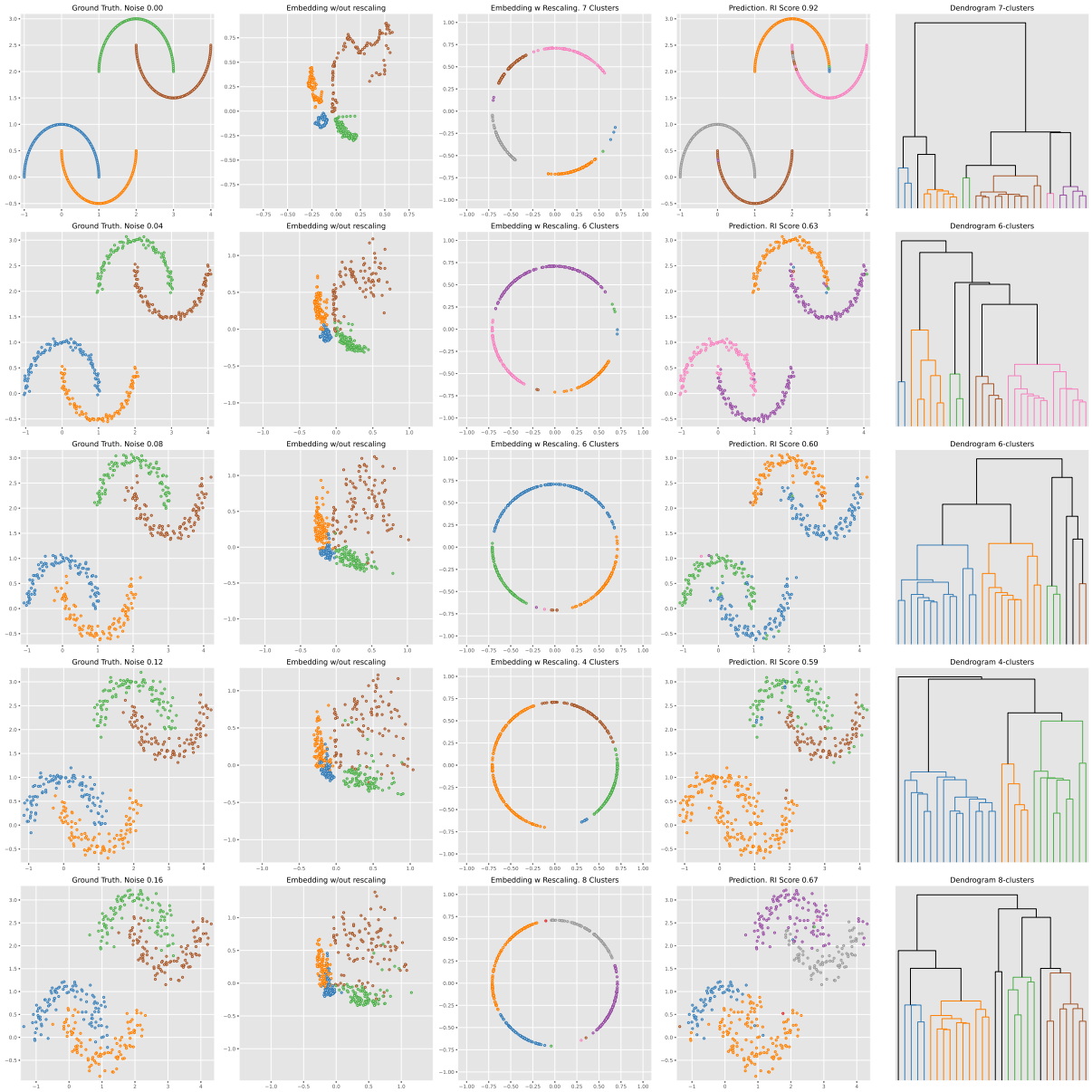
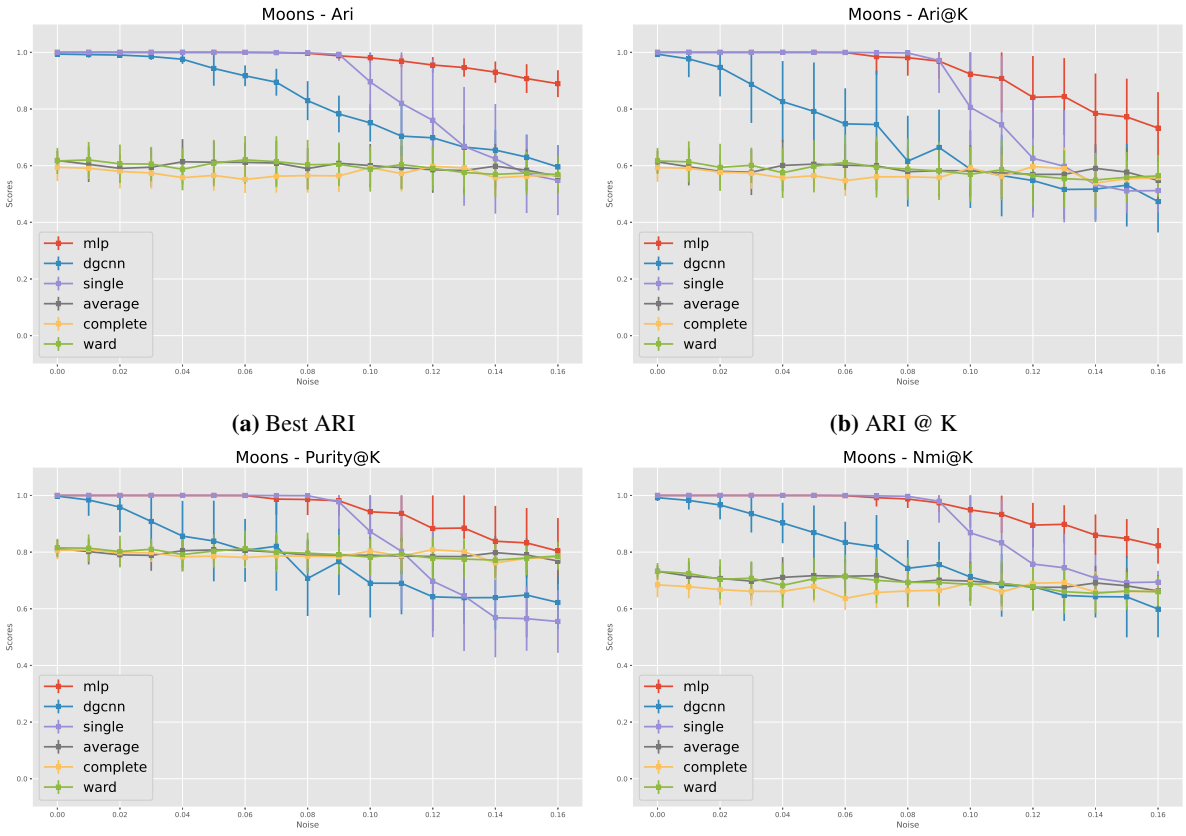


Figure 5.15 Moons dataset. The model used for prediction is a DGCNN trained on samples without noise. From top to bottom, each row is a case with an increasing level of noise. In the first column the input points, while in the second column we illustrate hidden features. Points are colored according to ground truth. The third column illustrates the hidden features after projection to Poincaré Disk. Fourth column shows predicted labels, while the fifth column shows associated dendrograms. Colors in the last three columns are assigned according to predicted labels.



(a) Best ARI

(b) ARI @ K

(c) Purity @ K

(d) NMI @ K

Figure 5.16 Robustness to noise of models on Moons. We compare trained models against classical methods as Single Linkage, Average Linkage, Complete Linkage, Ward's Method. The models used have been trained on a dataset without noise. Test sets used to measure scores contain 20 samples each. Plots show mean and standard deviation of scores obtained. During the experiments on these datasets MLP has shown a higher robustness to noise compared with the other models.

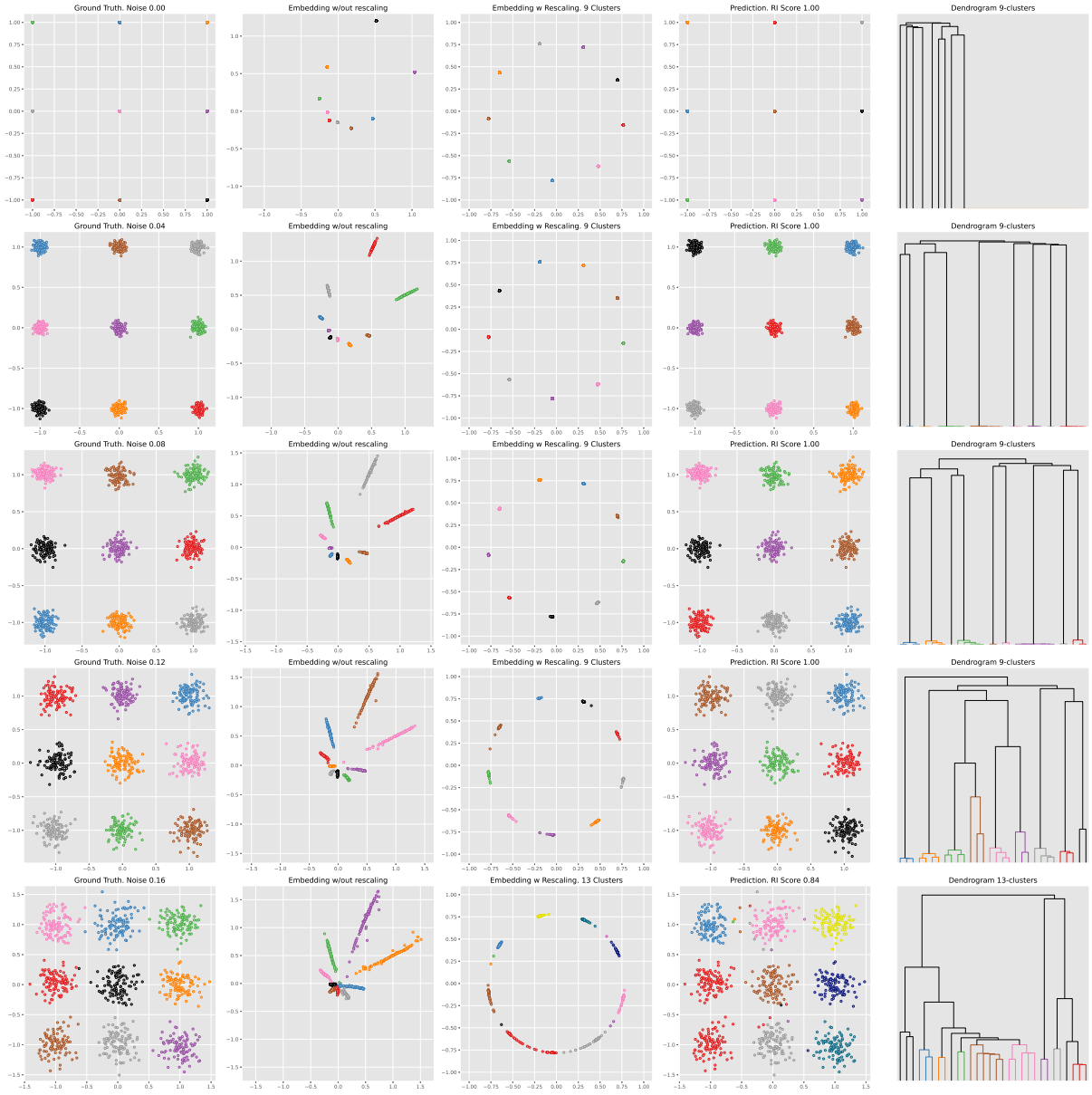


Figure 5.17 Blobs dataset. The model used for prediction is a MLP trained on samples with standard deviation value at 0.08. From top to bottom, each row is a case with an increasing value of standard deviation. In the first column the input points, while in the second column we illustrate hidden features. Points are colored according to ground truth. The third column illustrates the hidden features after projection to Poincaré Disk. Forth column shows predicted labels, while the fifth column show associated dendrograms. Colors in the last three columns are assigned according to predicted labels.

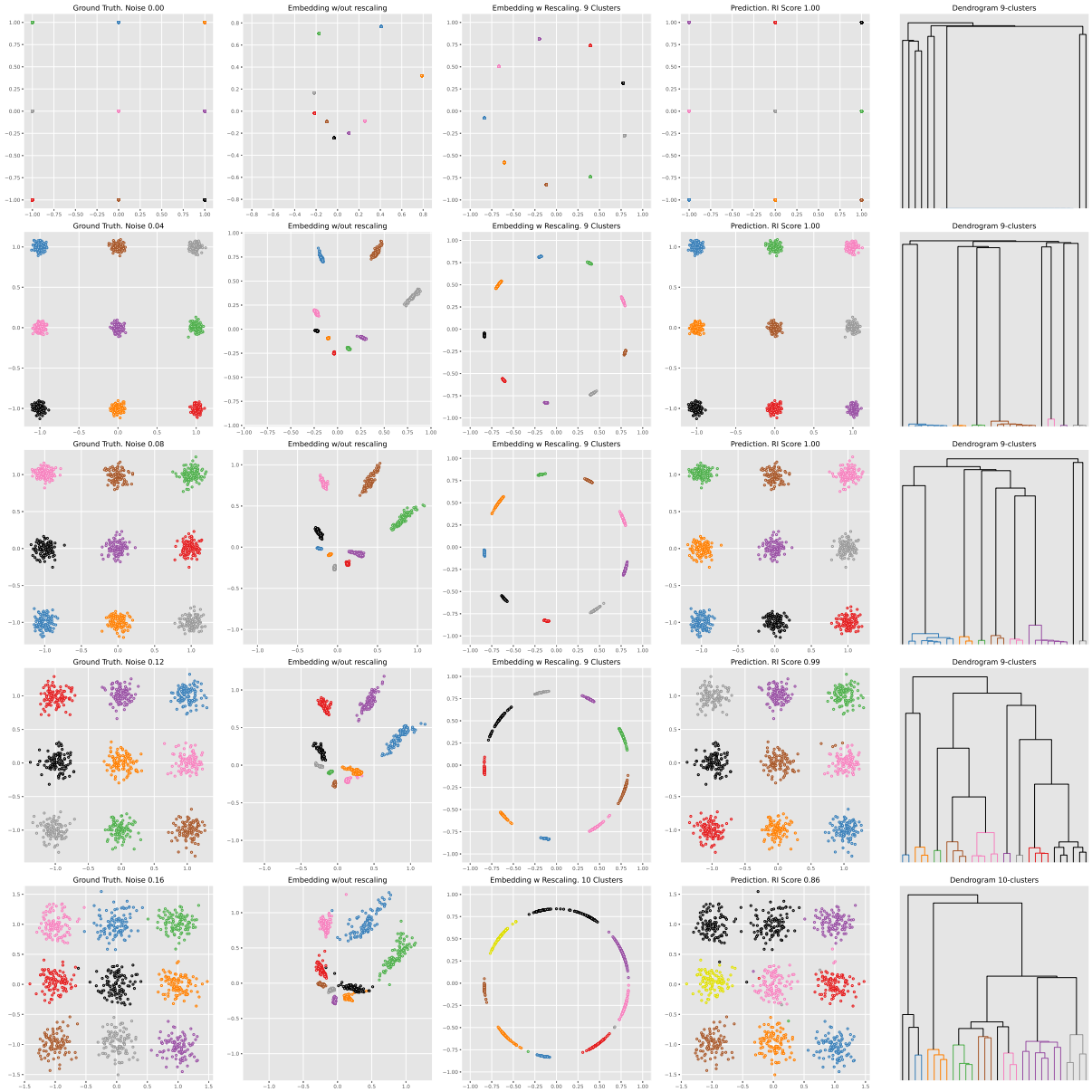
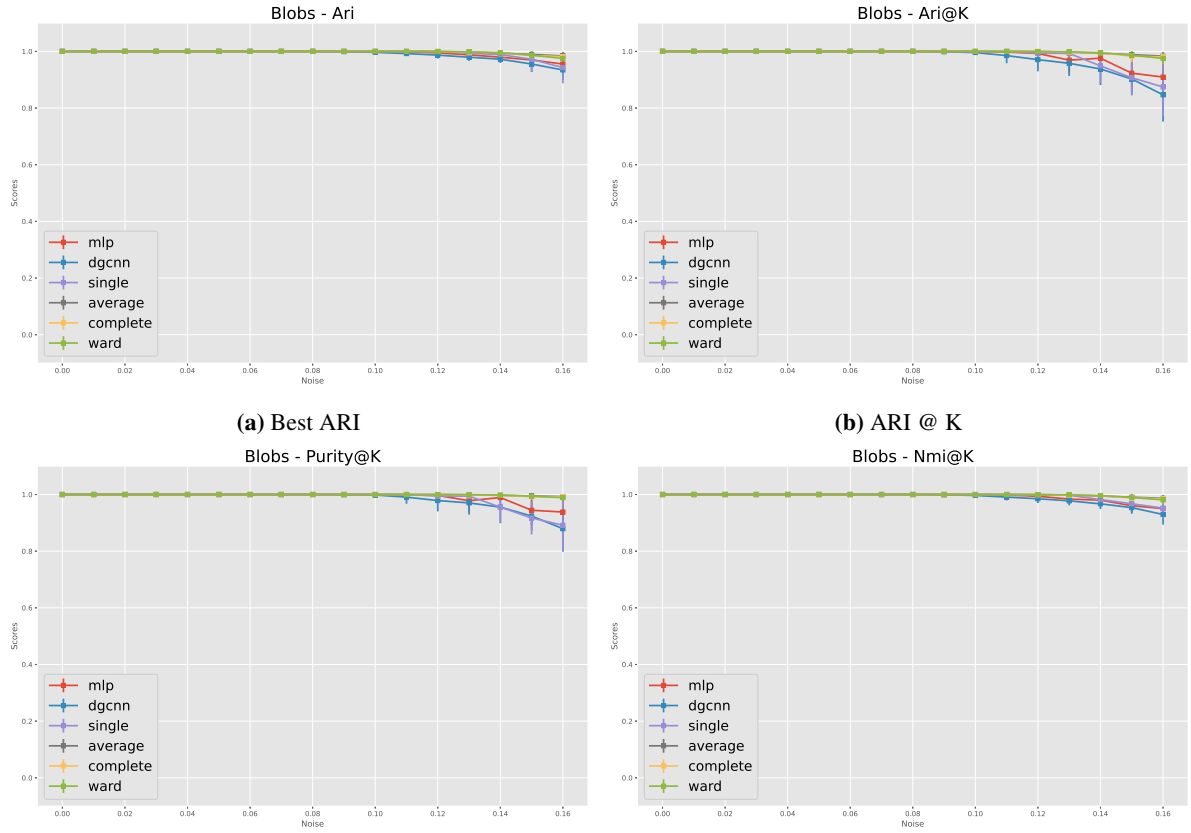


Figure 5.18 Blobs dataset. The model used for prediction is a DGCNN trained on samples with standard deviation value at 0.08. From top to bottom, each row is a case with an increasing value of standard deviation. In the first column the input points, while in the second column we illustrate hidden features. Points are coloured according to the ground truth labels. The third column illustrates the hidden features after projection to Poincaré Disk. Fourth column shows predicted labels, while the fifth column show associated dendrograms. Colors in the last three columns are assigned according to predicted labels.



(a) Best ARI

(b) ARI @ K

(c) Purity @ K

(d) NMI @ K

Figure 5.19 Robustness to noise of models on Blobs. We compare trained models against classical methods as Single Linkage, Average Linkage, Complete Linkage and Ward's Method. The models used have been trained on a dataset with Gaussian's standard deviation fixed at 0.08. Test sets used to measure scores contain 20 samples each. Plots show mean and standard deviation of scores obtained. During the experiments on these datasets MLP has shown a higher robustness to noise compared with DGCNN. In this case classical methods show better performances compared to trained models.

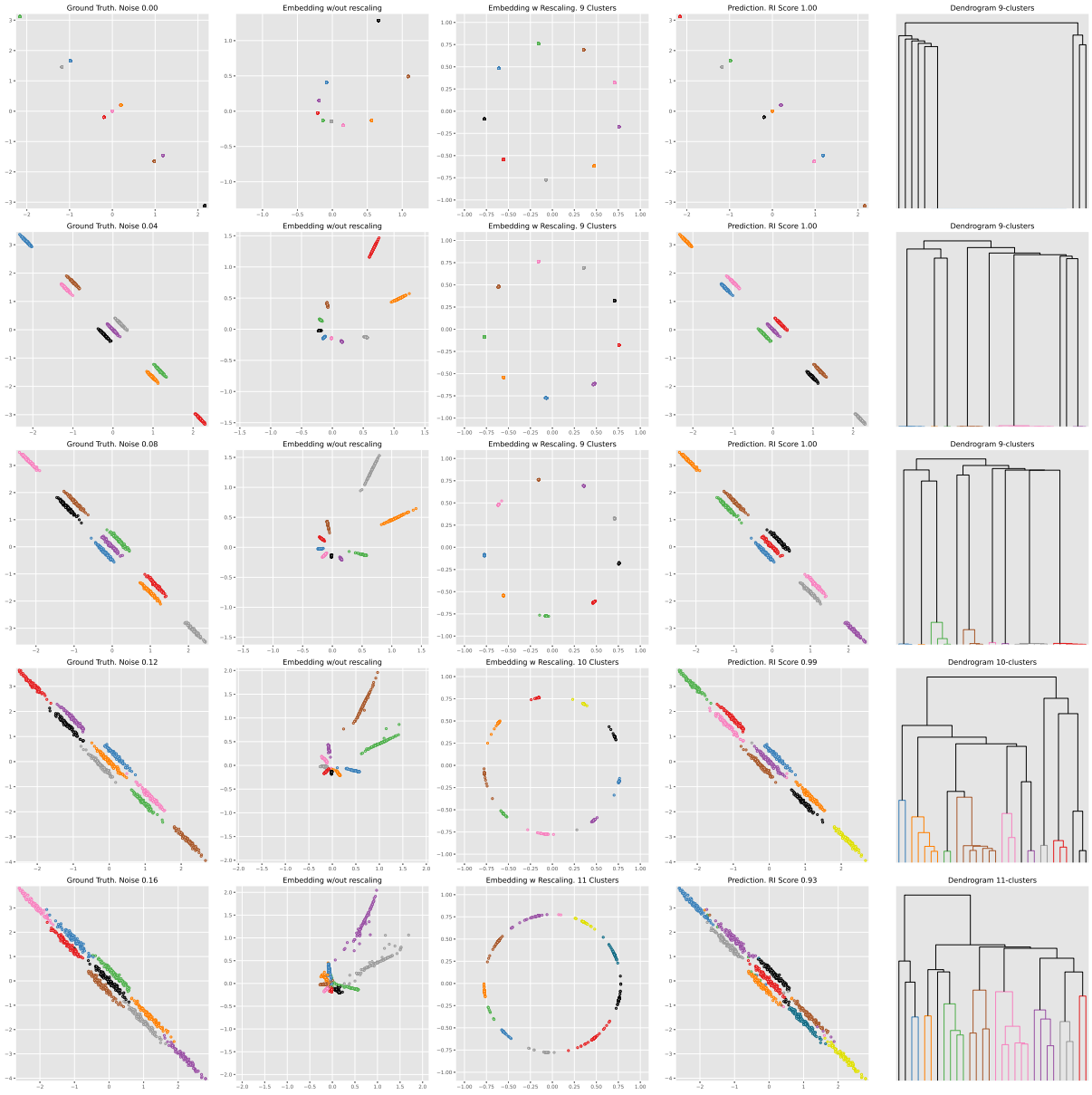


Figure 5.20 Anisotropic dataset. The model used for prediction is a MLP trained on samples with standard deviation value at 0.08. From top to bottom, each rows is a case with an increasing value of standard deviation. In the first column the input points, while in the second column we illustrate hidden features. Points are colored based on ground truth. The third column illustrate the hidden features after projection to Poincaré Disk. Forth column shows predicted labels, while the fifth column show associated dendrograms. Colors in the last three columns are assigned according to predicted labels.

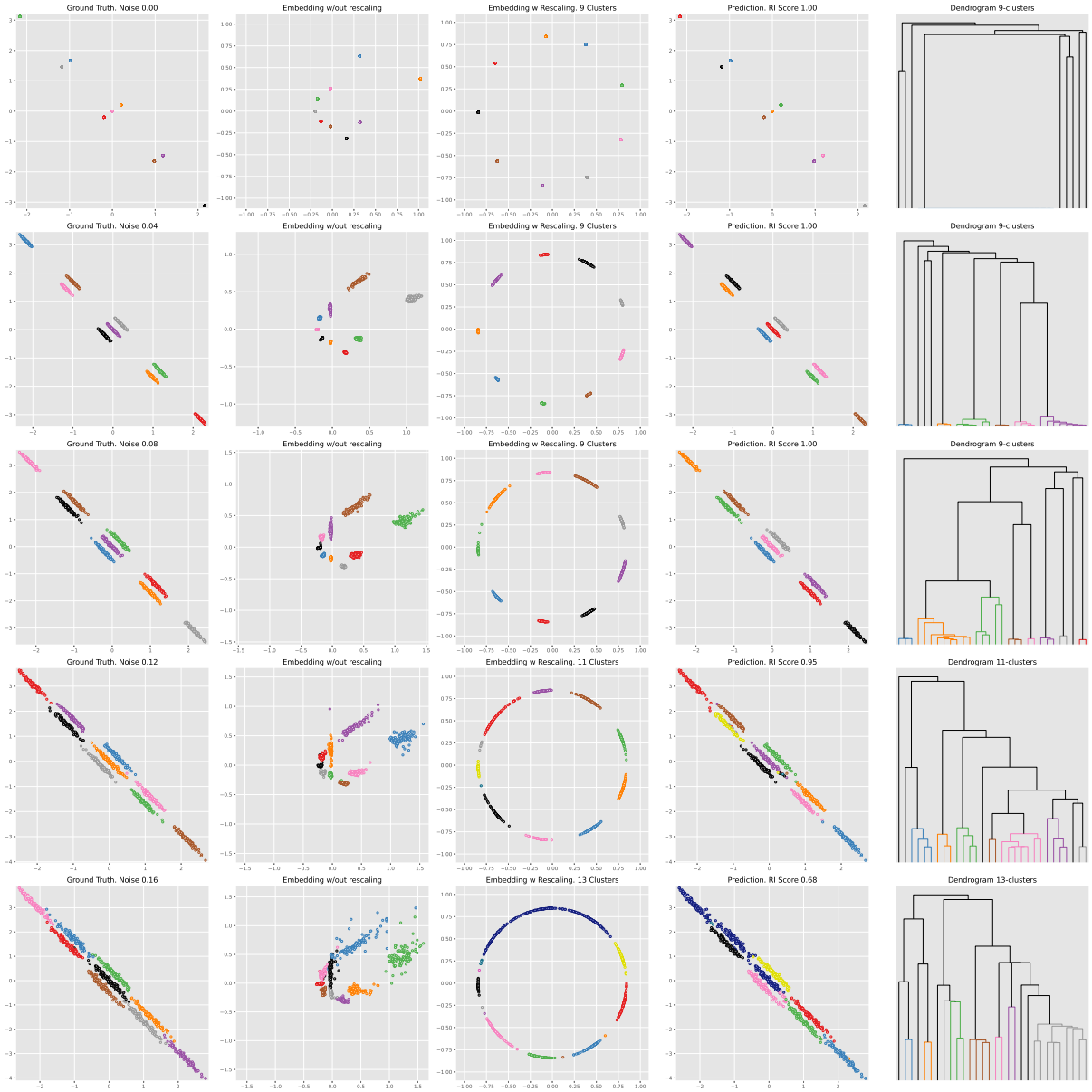
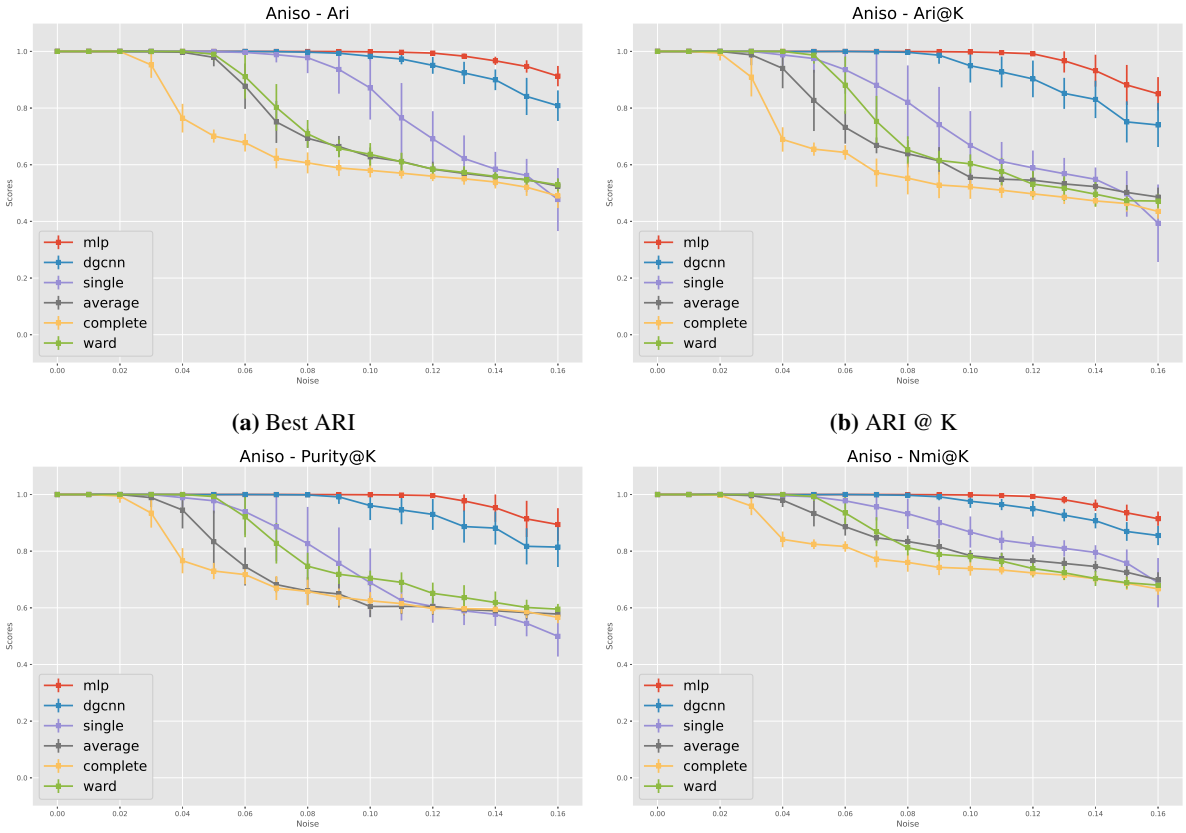


Figure 5.21 Anisotropic dataset. The model used for prediction is a DGCNN trained on samples with standard deviation value at 0.08. From top to bottom, each rows is a case with an increasing value of standard deviation. In the first column the input points, while in the second column we illustrate hidden features. Points are colored according to ground truth. The third column illustrate the hidden features after projection to Poincaré Disk. Forth column shows predicted labels, while the fifth column show associated dendrograms. Colors in the last three columns are assigned according to predicted labels.



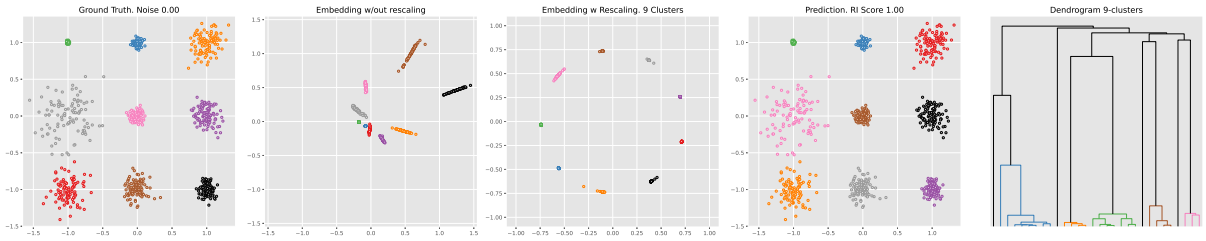
(a) Best ARI

(b) ARI @ K

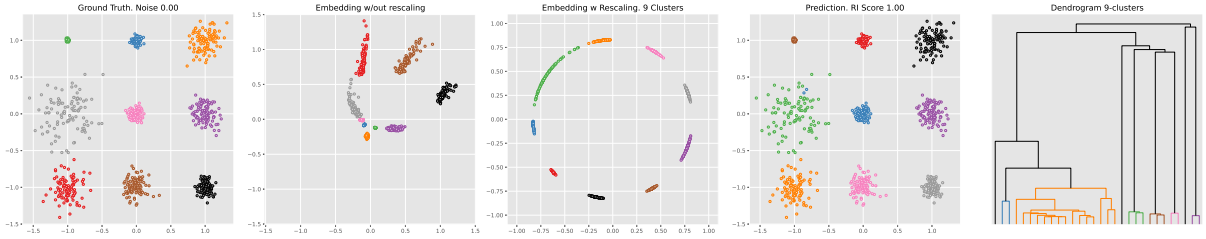
(c) Purity @ K

(d) NMI @ K

Figure 5.22 Robustness to noise of models on Anisotropic dataset. We compare trained models against classical methods as Single Linkage, Average Linkage, Complete Linkage and Ward's Method. The models used have been trained on a dataset with Gaussian's standard deviation fixed at 0.08. Test sets used to measure scores contain 20 samples each. Plots show mean and standard deviation of scores obtained. During the experiments on these datasets MLP has shown a higher robustness to noise compared with DGCNN.

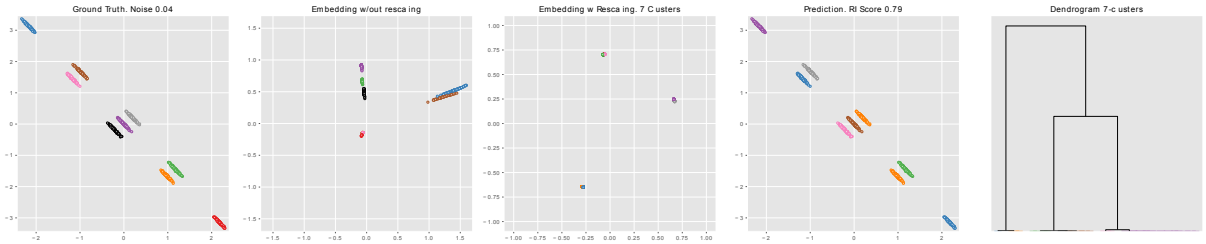


(a) MLP

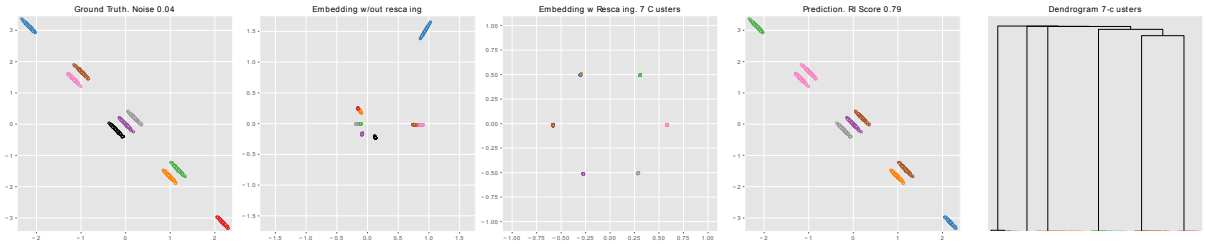


(b) DGCNN

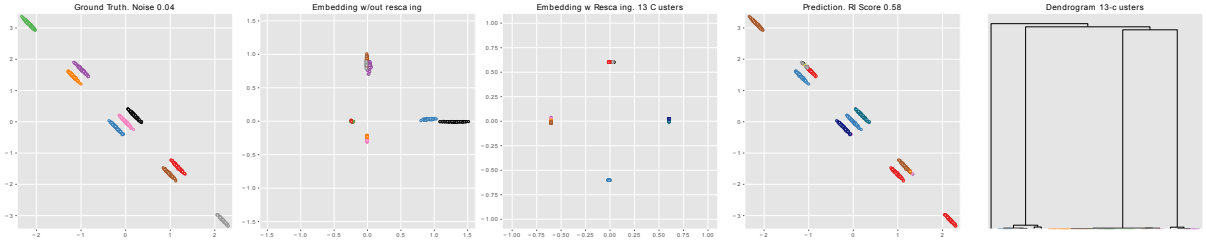
Figure 5.23 Varied dataset. (a) The model used for prediction is a MLP, while (b) the model used for prediction is a DGCNN. In the first column the input points, while in the second column we illustrate hidden features. Points are colored according to ground truth labels. The third column illustrates the hidden features after projection to Poincaré Disk. Fourth column shows predicted labels, while the fifth column show associated dendrograms. Colors in the last three columns are assigned according to predicted labels.



(a) $\alpha = 0$



(b) $\alpha = 0.5$



(c) $\alpha = 1$

Figure 5.24 The choice of value for margin α plays an important role for the quality of the results.

6 Towards Morphological Convolutions on Graphs

Resumé

Ce chapitre se concentre sur les réseaux de neurones sur des graphes. Dans la première partie du chapitre, nous donnons une introduction aux réseaux de neurones sur les graphes, en présentant les principaux concepts et en résumant l'état actuel de l'art. Dans la deuxième partie, nous proposons une extension d'un type particulier de couche sur les graphes, le *Edge-Convolution*, dans le cas de l'algèbre max-plus. L'objectif est de proposer une version morphologique de l'opération de convolution sur les graphes.

6.1 Graph Neural Networks

The great breakthrough lead by Deep Neural Networks (DNN) and in the field of Computer Vision by Convolutional Neural Network (CNN) has raised the question if it was possible to adapt these models also to different domains rather than images Bronstein et al. (2017). Thanks to their great adaptability to a wide range of applications, graphs have become an interesting field of study. First efforts to extend mathematical tools as Fourier transform to the graph domain come from the field of graph signal processing (Chung, 1997). Nonetheless, the first definition of a Graph Neural Network has been proposed as a recurrent network (Scarselli et al., 2009). In the first part of the chapter, we review the state of the art about Graph Neural Networks (GNN). The interested reader may refer to Bacciu et al. (2020) for an introduction to GNN. In the second part, we propose a novel Morphological Convolution Layer.

From now on, we assume that a weighted graph $\mathcal{G} = (V, E, W)$ is enriched with additional information on both nodes and edges. In particular, let $h_v \in \mathbb{R}^{d_1}$ be the feature vector associated to node $v \in V$ and $h_{vw} \in \mathbb{R}^{d_2}$ be the feature vector associated to edge $e = (v, w) \in E$. Generally, Deep Graph Networks (DGN) take as input a graph and generate a representation for each node v of the graph and in some cases also a representation for the entire graph. Figure 6.1 illustrates a representation of a DGN.

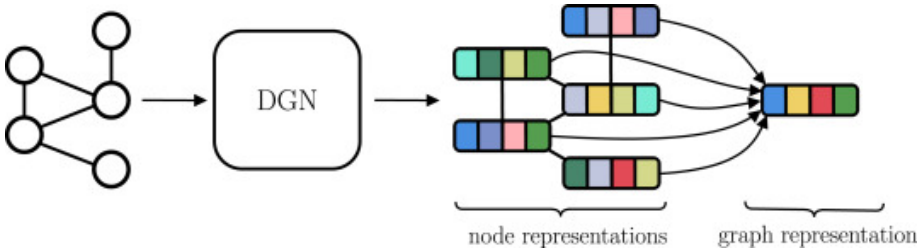


Figure 6.1 A *Deep Graph Network* takes as input a graph and produces for each node a hidden representation. Such node representations can be aggregated to generate a representation for the entire graph. Image source Bacciu et al. (2020)

6.1.1 Message Passing

The Message Passing mechanism is the building block for the construction of Graph Neural Networks. It has been introduced by Gilmer et al. (2017), as a generalization of different definitions of context information aggregation. At each layer $\ell \in \{1, \dots, L\}$ of the network, Message Passing is made by two functions, a message function M_ℓ and a vertex update function U_ℓ. The message function takes as input a hidden vector h_v^ℓ for node v and aggregates local information as

$$m_v^{\ell+1} = \underset{w \in \mathcal{N}(v)}{\square} M_\ell(h_v^\ell, h_w^\ell, h_{(v,w)}), \tag{6.1}$$

where $\square$ is any differentiable, permutation invariant function, such as sum, maximum, minimum and, average. The function $\square$ should be permutation invariant, mainly because in many cases there is not a unique way to order edges. For this reason, we look for a function that is invariant to the order in which the edges may appear. Once obtained the vector containing aggregated information, the update function returns the new hidden feature vector $h_v^{\ell+1}$ as:

$$h_v^{\ell+1} = U_l(h_v^\ell, m_v^{\ell+1}). \tag{6.2}$$

We remark that Message Passing only depends on the neighbourhood of the vertex v, thus is independent of graph size. This make space/time complexity of this operation $\mathcal{O}(|E|)$, that reduces to $\mathcal{O}(|V|)$ for sparse graphs. Later on, we will see how convolutions on a graph (Bruna et al., 2014; Defferrard et al., 2016; Kipf and Welling, 2017; Wang et al., 2019b) are defined using this mechanism. In classification problems, where the goal is to classify an input graph G, after L steps of message passing, a particular function called *READOUT* is used to compute a feature vector for the whole graph. Specifically, the readout function R is another permutation invariant function that takes as input the set of node feature vector

$$h_{\mathcal{G}} = R(\{h_v^L \mid v \in \mathcal{G}\}). \tag{6.3}$$

Readout function operates on the set of nodes features vector and must be invariant to permutation in order to be invariant by graph isomorphism.

6.1.2 Convolution on graph

A first attempt to extend the concept of convolution on graph has been made using spectral methods Bruna et al. (2014); Defferrard et al. (2016), mainly using the Graph Fourier Transform. Basically, let L be the graph Laplacian matrix. Since L is symmetric positive semidefinite matrix it exists an orthonormal basis of eigenvectors such that $L = U\Lambda U^T$, where U is the matrix of the eigenvectors and Λ is the diagonal matrix of eigenvalues, that is $\Lambda = diag([\lambda_1, \dots, \lambda_N])$. The eigenvectors of the Graph Laplacian are used to compute the Graph Fourier Transform of a signal $x \in \mathbb{R}^N$ on a graph as $\mathcal{F}(x) = U^T x$. Since there does not exist a translation operator on the nodes' domain, the convolution operator can be defined on the Fourier domain using the Graph Fourier Transform. Given $x, y \in \mathbb{R}^N$ two signals on the graph, the convolution operator is defined as:

$$x * y = \mathcal{F}^{-1}(\mathcal{F}(x) \odot \mathcal{F}(y)) = U((U^T x) \odot (U^T y)), \tag{6.4}$$

where $\odot$ is the element-wise Hadamard product. Thus, a signal x is filtered by g_θ as

$$g_\theta * x = g_\theta(L)x = U g_\theta(\Lambda) U^T x, \tag{6.5}$$

where $g_\theta(\Lambda) = diag(\theta)$ and the parameter θ is a vector of Fourier coefficients. In order to avoid computing the complete eigendecomposition of the matrix L, Bruna et al. (2014) propose to use the first k eigenvectors of the matrix U since for practical reasons they are the ones that more influence the convolution. However, this approach has two limitations. The first is that the filters are not localized in space and the second is that their learning complexity scales with the dimension of the input graph, that is $O(N)$. *ChebNets* (Defferrard et al., 2016) proposes to use the truncated Chebyshev expansion (Hammond et al., 2011) to overcome these issues. The idea is to rewrite the filtering operation as

$$g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x, \tag{6.6}$$

where $\tilde{L} = \frac{2}{\lambda_{max}} L - I_n$ is the scaled Laplacian and $T_k(\cdot)$ is the Chebyshev polynomial of order k, that respects the following recurrent definition $T_k(x) = 2xT_{k-1}(x) - T_{k-1}(x)$ with $T_0 = 1$ and $T_1 = x$. This approach solves the spatial problem because the convolution is a K order polynomial in the Laplacian, and thus it depends only on nodes that are at maximum K steps away from the central node. Moreover, it avoids computing the eigenvectors of the graph Laplacian and the complexity of evaluating a convolution is reduced to $\mathcal{O}(K|E|)$. Finally, Kipf and Welling (2017) introduce the Graph Convolutional Network (also known as *Vanilla GCN*), truncating the above expansion at the first term, assuming that $\lambda_{max} \approx 2$ and using the *renormalization trick*:

$$H^{\ell+1} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^\ell\Theta^\ell), \tag{6.7}$$

where $\tilde{A} = A + I_N$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. The feature vector at layer ℓ is $H^\ell \in \mathbb{R}^{N \times d_\ell}$, while at layer $\ell+1$ is $H^{\ell+1} \in \mathbb{R}^{N \times d_{\ell+1}}$. Finally, $\Theta^\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ are shared trainable parameters of the network. Even though in this definition of convolution only the closest neighbours of a node are considered, the K-localization can be achieved by chaining K convolutional layers. Equation 6.7 can be rewritten using Message Passing

functions as:

$$m_v^{\ell+1} = \sum_{w \in \mathcal{N}(v)} \tilde{A}_{vw} (deg(v) deg(w))^{-1/2} h_w^{\ell}, \tag{6.8}$$

and the update function is

$$h_v^{\ell+1} = \sigma(\langle \theta^{\ell}, m_v^{\ell+1} \rangle), \tag{6.9}$$

where θ^{ℓ} is the vector of parameters of the network. One property of this definition of convolution is isotropy. Differently from convolutions on images, where we have a notion of up, down, left and right, in the graph domain we do not have an explicit notion of direction and thus all the neighbours are treated equally. Similar to Vanilla GCN, there exist two other isotropic definitions of convolution of graph, that are Graph SAGE (Hamilton et al., 2018) and GraphConv (Morris et al., 2019). Both of them are inspired by the Weisfeiler-Lehman (WL) isomorphism test, which is a classic algorithm to test if two graphs are isomorphic. The idea is that if two graphs are isomorphic, then the features extracted by a network must be the same on the two graphs. They propose to learn a set of functions that learn how to aggregate information from a node's local neighbourhood. Each layer is defined as

$$h_v^{\ell+1} = \sigma\left(\Theta_1^{\ell} h_v^{\ell} + \underset{w \in \mathcal{N}(v)}{\square} \Theta_2^{\ell} h_w^{\ell} \right), \tag{6.10}$$

where $\Theta_1^{\ell}, \Theta_2^{\ell}$ are trainable parameters, and $\square$ can be a summation of a maximum of $\Theta_2^{\ell} h_w^{\ell}$ or a Long-Short Term Memory of h_w^{ℓ} (Hochreiter and Schmidhuber, 1997).

Edge Convolution

Another family of convolutions still based on the Message Passing technique are Edge Convolutions (Atzmon et al., 2018; Wang et al., 2019b). Differently from the previous case, this second family is an anisotropic version of convolution on graphs. In this particular class of convolutions, the update function is such that $h_v^{\ell+1} = m_v^{\ell+1}$. In other words, Edge convolution is defined as

$$h_v^{\ell+1} = \underset{w \in \mathcal{N}(v)}{\square} M_{\ell}(h_v^{\ell}, h_w^{\ell}, h_{(v,w)}). \tag{6.11}$$

A first particular example of this class is PointNet Charles et al. (2017), that can be seen as a convolution on a graph where the set of edges is empty, *i.e.* $E = \emptyset$ and the function $M_{\ell}(h_v^{\ell}, h_w^{\ell}, h_{(v,w)}) = M_{\ell}(h_v^{\ell})$. In this case, the convolution encodes only global shape information and does not consider local neighbourhood structure. Other examples use a notion of local patch on a graph, in which a convolution operation can be defined (Atzmon et al., 2018; Monti et al., 2017; Simonovsky and Komodakis, 2017; Veličković et al., 2018). In these cases, the convolution is defined as

$$h_v^{\ell+1} = \sum_{w \in \mathcal{N}(v)} f_{\theta}(h_w^{\ell}) \cdot g(u(h_v^{\ell}, h_w^{\ell})), \tag{6.12}$$

where f_{θ} is a function in the parameters of the network often implemented as a MLP, g is a Gaussian kernel and u computes the pairwise Euclidean distance. In their work Wang et al. (2019b) they propose

for the case of a graph $\mathcal{G}$ that is the $k-NN$ graph of a set of features $\mathcal{X} = \{h_1^\ell, \dots, h_n^\ell\} \subset \mathbb{R}^d$

$$h_v^{\ell+1} = \max_{w \in \mathcal{N}(v)} \mathrm{ReLU}(\theta_m \cdot (h_w^\ell - h_v^\ell) + \phi_m \cdot h_v^\ell), \tag{6.13}$$

where the function inside ReLU can be implemented using a Multi-Layer Perceptron (MLP) and the weights θ_m and ϕ_m are shared among the nodes. This last definition combines global shape structure information captured by the vectors h_v^ℓ with the local neighbourhood information captured by $h_v^\ell - h_w^\ell$.

6.1.3 Attention on GNN

Another mechanism that allows the implementation of anisotropic convolutions is attention. This last one has been introduced in the context of Natural Language Processing (NLP). In the context of NLP, attention mechanism allows putting more importance on a certain relationship between two words in a sentence or close context. The idea takes inspiration from human visual attention that focuses a certain region on the foreground while perceiving the rest in the background in "low resolution". When an attention mechanism is used to compute a representation of a single sequence, it is commonly referred to as *self-attention*. Among others, Vaswani et al. (2017) used *self-attention* in their Transformer Architecture, obtaining state-of-the-art performances on the machine translation task. Taking inspiration from this last work, Veličković et al. (2018) introduced an attention-based architecture called Graph Attention Network (GAT) to perform node classification of graph structured data. The input of a layer is a set of node features $\{h_1, \dots, h_N\} \in \mathbb{R}^D$ and the output is a new set of node features $\{h'_1, \dots, h'_N\} \in \mathbb{R}^{D'}$. Firstly, a linear transformation $W : \mathbb{R}^D \to \mathbb{R}^{D'}$ is applied to every node. Then a shared self-attention mechanism $a : \mathbb{R}^{D'} \times \mathbb{R}^{D'} \to \mathbb{R}$ computes attention coefficients

$$c_{ij} = a(Wh_i, Wh_j),$$

for each edge $e_{ij} \in E$. The coefficient c_{ij} can be explained as the *importance of node j's features to node i*. In order to fair compare all the coefficients of the same node i these are renormalized using a softmax function as follows:

$$\alpha_{ij} = \mathrm{softmax}_j(c_{ij}) = \frac{\exp(c_{ij})}{\sum_{v_k \in \mathcal{N}(v_i)} \exp(c_{ik})}$$

Specifically, in the proposed implementation of GAT architecture, the self-attention function is implemented as a feed-forward neural network, parametrized by a vector $\vec{a} \in \mathbb{R}^{D'}$ and applying a LeakyReLU non-linearity afterwards defined as $\mathrm{LeakyReLU}(x) \coloneqq \max(\epsilon x, x)$, where $0 \leq \epsilon \leq 1$. Thus, the expanded formulation of normalized self-attention coefficients is

$$\alpha_{ij} = \frac{\exp\Big(\mathrm{LeakyReLU}\Big(\vec{a}^T[Wh_i \| Wh_j]\Big)\Big)}{\sum_{v_k \in \mathcal{N}(v_i)} \exp\Big(\mathrm{LeakyReLU}\Big(\vec{a}^T[Wh_i \| Wh_k]\Big)\Big)},$$

where $\|$ has to be meant as the concatenation operation. Finally, the new feature vectors are computed as

$$h'_i = \sigma\Big(\sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij} W h_j\Big),$$

where σ is a non-linear function. Figure 6.2 illustrates the self-attention mechanism for a single node.

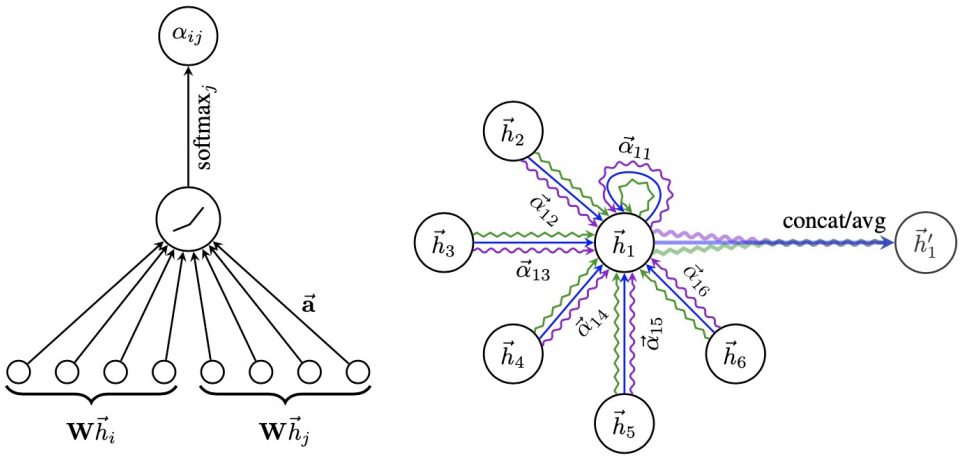


Figure 6.2 Left: Attention Mechanism $a(Wh_i, Wh_j)$ implemented as a feedforward neural network. **Right:** A schematic illustration representing the self-attention mechanism for a single node v_1 having as feature vector h_1. (Image source Veličković et al. (2018))

Multi-headed attention is implemented using K different self-attention functions $a_1, \ldots, a_K$ and then concatenating the output features as follows:

$$h'_i = \Big\|_{k=1}^{K} \sigma\Big(\sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij}^k W^k h_j \Big).$$

Please note that the output feature h'_i is a KD' (rather than D') dimensional vector for each node.
A similar strategy is employed by Wang et al. (2019a) in their proposed Graph Attention Convolution (GAT). This architecture has been proposed to be used in the context of point cloud semantic segmentation task. Nodes of the graph are points $\{p_1, \ldots, p_N\}$ of the point cloud. In this version, also initial point positions are used in the activation function and attention scores are computed component-wise, that means that the components of an attention vector $\alpha_{ij} = (\alpha_{ij,1}, \ldots, \alpha_{ij,D'}) \in \mathbb{R}^{D'}$ are computed as

$$\alpha_{ij,d} = \frac{\exp\Big(c_{ij,d}\Big)}{\sum_{v_k \in \mathcal{N}(v_i)} \exp\Big(c_{ik,d}\Big)},$$

and the features vector $c_{ij} \in \mathbb{R}^{D'}$ are obtained using a feedforward network $M_\alpha : \mathbb{R}^{3+D} \to \mathbb{R}^{D'}$ as $c_{ij} = M_\alpha(\Delta p_{ij} \| \Delta h_{ij})$. Compared to GAT, this strategy allows incorporating local spatial relationships between neighbouring points. For the curious reader, Joshi (2020) proposes an interesting analogy between Graph Neural Networks and Transformer architectures.

6.1.4 Graph pooling

Pooling is a widely used operation in CNN architectures. The image domain is partitioned in non-overlapping rectangles, on which a non-linear aggregation function (e.g. max) is applied. The main advantages of this operation are to reduce the size of the feature maps and to increase the *receptive field* of the network. Different approaches to adapt this operation in the context of graphs have been proposed during the years. Clearly, the main challenge comes from the fact that the underlying graph is not a regular

grid. Among all the adopted strategies to tackle the problem, we distinguish between two. The first relies on learning a soft-assignment matrix $S \in \mathbb{R}^{(N,k)}$ where N is the number of nodes in the graph and k is the number of nodes in the graph after the pooling. Methods that employ this kind of approach are for example DiffPool (Ying et al., 2018), or MinCutPool (Bianchi et al., 2020). Both methods design specific loss functions to find the optimal assignment. The main drawback of this kind of method is that the number of nodes in the graph is fixed intentionally. The second approach is based on learning a function to score nodes and then selecting the k most important nodes according to the scoring function, as in Top-k pooling (Gao and Ji, 2021), or in SAG-Pool (Knyazev et al., 2019; Lee et al., 2019). A small variation of this last approach is EdgePool Diehl (2019a,b) that aims to learn to score functions for the edges that measure similarity between endpoint nodes. Successively, the edges are sorted by their score, and then it contracts the edges with the highest score and whose endpoints have not yet been part of a contracted edge.

6.2 Towards Morphological Graph Convolutions

In this section, we propose to extend the Edge-Convolution layer on the max-plus algebra. Our goal is to propose a morphological version of Graph Convolutions. The first definitions of morphological operators on graph have been proposed in Vincent (1989). Later Cousty et al. (2013) extended on subgraphs the basic morphological operators. Another approach has been proposed by Velasco-Forero and Angulo (2015), and Blusseau et al. (2018), that reviewed the morphological operators with the graph signal processing approach and redefined dilation and erosion operators as max-plus version of convolution on graphs. The developments that we propose in this chapter assume somehow that the reader is familiar with the work of Velasco-Forero and Angulo (2015) and max-plus algebra. For this reason, we recall the basic definitions that will be useful later on.

Definition 6.1 (Max-Plus Semi-ring)**.** Let $(\mathbb{R}_{\max}, \oplus, \otimes)$ be the triple such that $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$ and $\oplus$, $\otimes$ are two binary operations defined as $x \oplus y = \max(x, y)$ and $x \otimes y = x + y$. $(\mathbb{R}_{\max}, \oplus, \otimes)$ is an idempotent semi-ring because it respects the following properties:

- The neutral element for $\oplus$ is $-\infty$, and 0 is the neutral element for $\otimes$.
- Both operations are associative and commutative.
- It holds $c \otimes (a \oplus b) = c + \max(a, b) = \max(a + c, a + b) = (a \otimes c) \oplus (b \otimes c)$ for every triple $a, b, c \in \mathbb{R}_{\max}$
- For every $a \in \mathbb{R}_{\max}$ it holds $a \otimes -\infty = \infty \otimes a = a - \infty = -\infty$.
- For every $a \in \mathbb{R}_{\max}$ it holds $a \oplus a = \max(a, a) = a$.

Definition 6.2 (Conservative Morphological Weight Matrix)**.** A weight matrix $W \in \mathbb{R}_{\max}^{n,m}$ is a *conservative morphological weight matrix* if $-\infty \leq w_{ij} \leq 0$ and $w_{ii} = 0$ for any $1 \leq i \leq n$ and $1 \leq j \leq m$.

The conservative morphological weight matrix can be seen as the adjacent weight matrix of a weighted graph $\mathcal{G} = (V, E)$, where $w_{ij} = -\infty$ if $(v_i, v_j) \notin E$. In this context, note that values $w_{ij}^{(p)}$ in the power matrix W^p (intended in a max-plus sense) are greater than $-\infty$ if and only if there exists in $\mathcal{G}$ a path from vertex v_i and vertex v_j in at most p hops.

Definition 6.3 (Dilation and erosion)**.** Let $h : V \to \mathbb{R}_{\max}$ a signal over the nodes of the graph. The dilation δ_W of signal h on the graph $\mathcal{G} = (V, E)$:

$$\delta_W(h)_i = \bigvee_{j=1}^{N} h_j + w_{i,j} \coloneqq (W \oplus h)_i, \tag{6.14}$$

where W is a conservative morphological weight matrix. The erosion is the dual adjoint operator of the dilation and is defined as:

$$\varepsilon_W(h)_i = \bigwedge_{j=1}^{N} h_j + W^*[i,j] = \bigwedge_{j=1}^{N} h_j - w_{j,i} \coloneqq (W^* \ominus h)_i, \tag{6.15}$$

where W^* is the adjoint weight matrix defined as $W^*[i,j] = -W[j,i]$.

Remark that the definition of dilation and erosion above include transformations by flat, non-flat, adaptive and non-local structuring elements. In the following section, our goal is to define a graph layer that learns an optimal morphological weight matrix.

6.2.1 MaxPlus Edge Convolution

Sharing Morphological Weights

In the definition of dilation and erosion operators, we remark that the morphological weight matrix can be seen as analogous in the max-plus sense of convolution filters. However, we have found the following drawbacks with the above definition.

- The size of the matrix W depends on the size $N = |V|$ of the input graph.
- An optimal W^* may not take into account local graph topology.

For this purpose, similarly to the classic case, we would like to learn a shared morphological weight matrix. In order to simplify our problem, we decide to restrict ourselves to the case in which the graph $\mathcal{G}$ is the k-NN graph of set of vertices V. Thanks to this hypothesis, we can reduce the representation of the morphological weight matrix to a vector $w = (w_1, \ldots, w_k) \in \mathbb{R}^k$ of size k. In this case, the dilation operator with a shared morphological weight w can be written as:

$$(w \oplus h)_i = \bigvee_{k'=1}^{k} h_{j_{k'}} + w_{k'}, \tag{6.16}$$

where $\{j_1, \ldots j_k\}$ is the set of indices of the k neighbours of the vertex v_i.

Multidimensional Case

Since so far, we have contemplated only the case of a one-dimensional signal defined on the graph nodes. In the general case, we need to treat multi-dimensional features vectors $h \in \mathbb{R}^D$. We can easily extend the operators to the case of a signal $h \in \mathbb{R}^D$ stacking D weights vectors and obtaining a matrix $W = [w_1, \ldots, w_D]$. In this case, the operator $\delta_W : \mathbb{R}^D_{\max} \to \mathbb{R}^D_{\max}$ maps a D dimensional signal to another

D dimensional signal, with the following expression:

$$\delta_W(h)_{i,d} = \bigvee_{k'=1}^{k} h_{j_{k'},d} + w_{k',d}, \tag{6.17}$$

where $\{j_1, \dots j_k\}$ are indices of the k neighbours of the vertex v_i and d is the d-th feature of the feature vector, *i.e.* $d \in \{1, \dots D\}$. In Figure 6.3, we illustrate an example of dilation applied on a two-dimensional graph. The signal is the vector of points coordinates, thus in this case is a two-dimensional vector. In the example, we illustrate the effect of the number k of neighbours used to build the k-NN graph on the output. The weight matrix used is obtained stacking the same vector twice, that is $W = [w, w]$. On the second row, the plots show the input signal (red points) and the output signal (blue points). As the reader can see, increasing the number k of neighbours makes the output signal to concentrate towards the upper-right corner of the space.

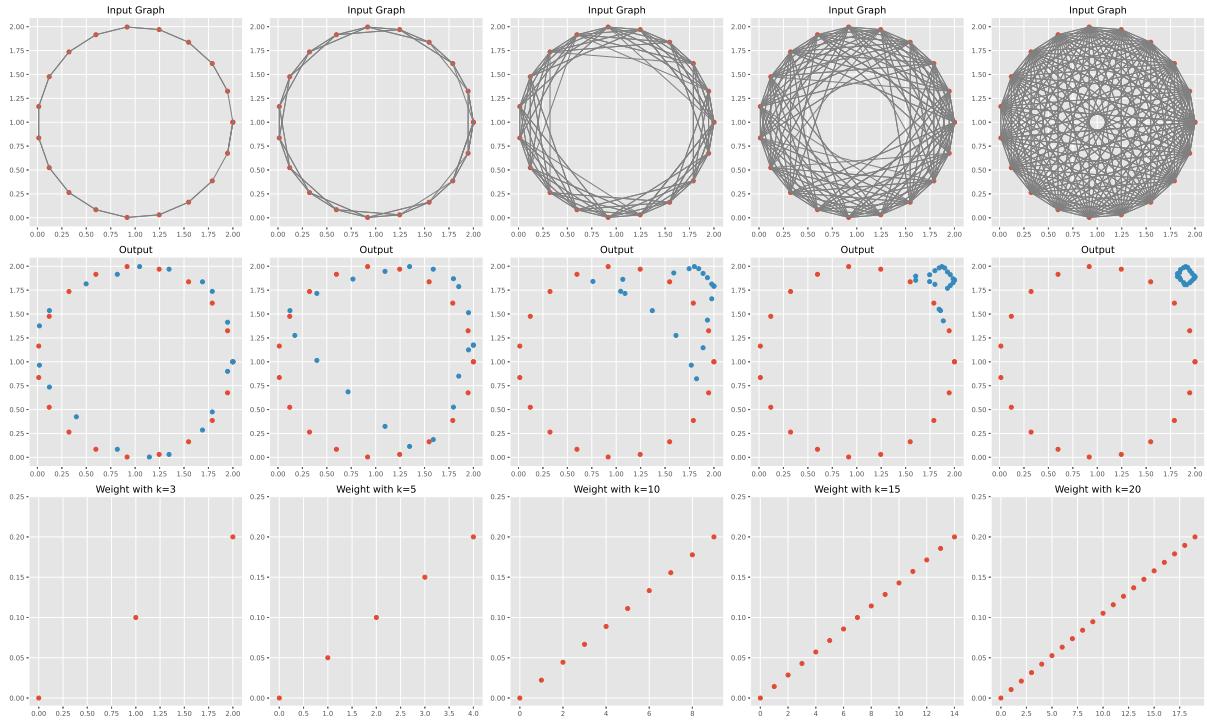
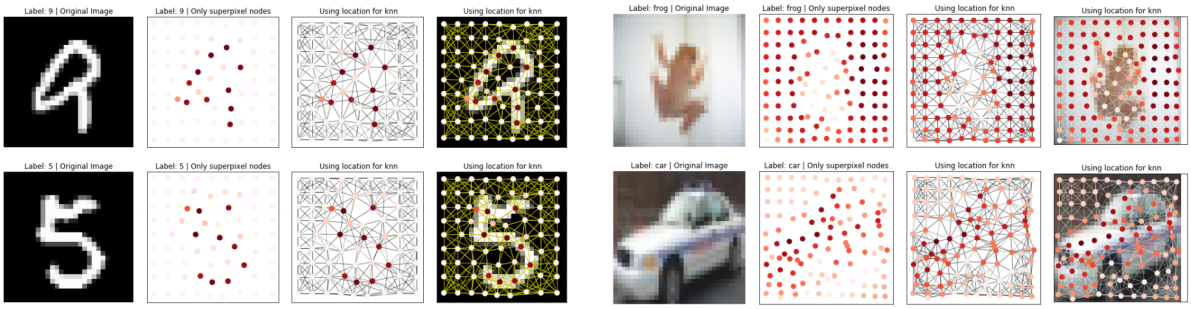


Figure 6.3 An example of multidimensional dilation on a graph in which we make vary the number of neighbours that we take into account. The input nodes are points aligned on a circle. In the first row, we show the input graphs. In each case, we increase the number of neighbours, k, that we consider building the graph. The second row illustrates input points (red) and output points (blue). The third row illustrates the absolute values of weights used for each example. The weight matrices have size $k \times 2$. To build the matrices W, we stacked the same vector $w \in \mathbb{R}^k$ twice.

Defining the layer

Now we have all the elements to define a MaxPlus Edge Convolution (MPEdge) as a variation of edge convolution on the max plus algebra. In a MPEdge transformation, we first apply an edge-convolution



(a) MNIST **(b)** CIFAR10

Figure 6.4 Samples images and super-pixels graph. Nodes correspond to superpixels of images obtained using SLIC Algorithm. In MNIST, graphs have at most 75 nodes, while at most 150 nodes in CIFAR-10. (Image Source Dwivedi et al. (2020))

followed by a convolution on the max-plus algebra:

$$h_v^{\ell+1} = \sigma\left(\delta_{\Theta^\ell}(f^\ell_{\theta_m,\phi_m}(h_v^\ell, h_w^\ell - h_v^\ell))\right) = \sigma\left(\bigvee_{w\in\mathcal{N}(v)} f^\ell_{\theta_m,\phi_m}(h_v^\ell, h_w^\ell - h_v^\ell) + \Theta^\ell\right), \tag{6.18}$$

where Θ^ℓ is the trainable morphological weight matrix and $f^\ell_{\theta_m,\phi_m}$ is the function used in Edge-Conv (6.13), defined as:

$$f^\ell_{\theta_m,\phi_m}(h_v^\ell, h_w^\ell - h_v^\ell) = \mathrm{ReLU}(\theta_m \cdot (h_w^\ell - h_v^\ell) + \phi_m \cdot h_v^\ell), \tag{6.19}$$

where θ_m and ϕ_m are trainable parameters as well. The Edge-Conv is used to map an input feature vector $h_v^\ell \in \mathbb{R}^F$ to an output $h' \in \mathbb{R}^D$. Successively, we apply the multidimensional dilation operator δ_{Θ^ℓ} to the signal h' of a permutation invariant aggregation function.

6.2.2 Benchmarking MPEdge Convolutions

Following the benchmark proposed in Dwivedi et al. (2020), we compared MPEdge against the state-of-the-art on the task of graph classification. In the same manner, we trained and tested our models on the SLIC super-pixels Achanta et al. (2012) versions of MNIST and CIFAR10 datasets. To build these datasets, the original images are converted to graphs using super-pixels that are small regions of homogeneous intensity in the images. Super-pixels are the nodes of the graph, and each node is connected to its k nearest neighbours. Given x_v, x_w the 2-D coordinates of super-pixels v and w, their similarity is defined as:

$$\delta(v,w) = \exp\left(-\frac{\|x_v - x_w\|^2}{\sigma_x^2}\right), \tag{6.20}$$

where σ_v is the scale parameter defined as the average distance x_k of the k nearest neighbours for each node. In our experiments, we chose $k = 9$. Graphs in MNIST dataset have at most 75 nodes. Input vertex features are the coordinates of the super pixel and its mean intensity (2D coordinates + 1D intensity = 3D features). Similarly, graphs in CIFAR10 have at most 150 nodes, while features used are the super-pixels coordinates and the average super-pixel intensity in each channel (2D coordinates + 3D RGB = 5D features). In Figure 6.4, we show a sample from MNIST and CIFAR10 respectively. For further information about the datasets, the reader may refer to Appendix A.2 in Dwivedi et al. (2020).

Experiment Setup: For the comparison we consider the following GNNs: GCN (Kipf and Welling, 2017), GraphSAGE (Hamilton et al., 2018), GraphConv (Morris et al., 2019), EdgeConv/DynamicEdgeConv Wang et al. (2019b) and MPEdge. We also included as baseline the graph agnostic MLP, which does not consider any connection between nodes, but it simply updates the nodes features as $h_v^{\ell+1} = \sigma\left(W^{\ell} h_v^{\ell}\right)$. We used the same architecture to test the different approaches. As shown in Figure 6.5 the model has three *Conv* layers that are the GNN we analyse. After this step, a global max pooling is applied, and the prediction is achieved using an MLP. The dimension of hidden features is fixed at 128 in all the steps. For each model, we perform five runs with different seeds, and we evaluate the multi-label classification accuracy on test and training sets. The results are averaged over the five runs. Remember that in DynamicEdgeConv the graph is updated after each convolution step. That means that the set of k nearest neighbours of a node changes from layer to layer of the network. At each level, in fact, the new set is computed using the node embeddings obtained after the convolution. Since MPEdge is based on EdgeConv, we aim to explore if MPEdge can also benefit from the dynamic update of the graph. For this reason, we consider two models. The first, called MPEdge, in which the connections between nodes are fixed. The second, named MPDynEdge, updates the graph after each convolution as in DynamicEdgeConv.

Our goal is to examine the effects brought by the trainable morphological parameters on the performance of the network. For this reason, we focus our study on EdgeConv, DynamicEdgeConv, MPEdge and MPDynEdge. In Figure 6.6, we show the training accuracy of the models. Each curve corresponds to a different run, while the colours refer to the GNN employed. As the reader can see, on the MNIST dataset the MPEdge based models reach on average higher accuracy compared to EdgeConv based models. At the same time, on CIFAR10, the two layers achieve comparable scores.

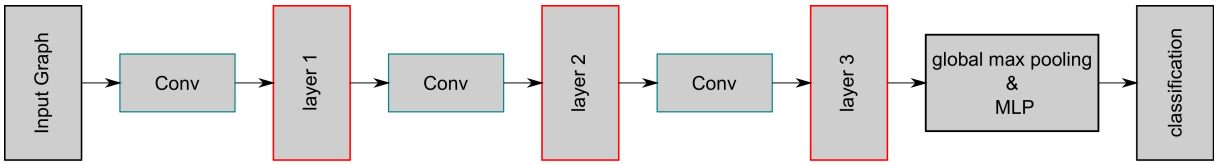
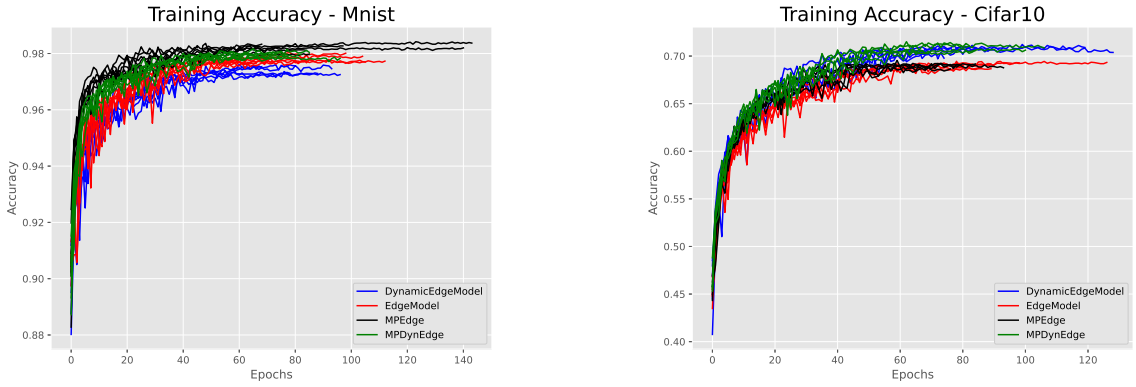


Figure 6.5 The architecture of the model used for the tests. The *Conv* step is set according to the GNN class analysed.

The experiments in Table 6.1 evaluate the different GNNs for the graph classification task on MNIST and CIFAR10. Results for MNIST show that all the models achieve high scores both in train and test sets. The MPEdge convolutions perform a bit better compared to all the others. This is confirmed also if we look at Figure 6.7a, which illustrates the accuracy scores obtained by the different GNN. Each dot is a different run. Moving our analysis to CIFAR10, note that in this case, all the models overfit the training set. In fact, there is a big gap between test and train accuracy. However, as said before, the results obtained by MPEdge are comparable with EdgeConv. Test accuracy scores, shown in Figure 6.7b, does not exhibit a model that clearly outperforms the other on CIFAR10. In this case, DynamicEdgeConv and MPDynEdge reach the highest accuracy score. Overall, results suggest that morphological parameters could be helpful to accentuate the anisotropic properties of the convolutions. At the same time, Figure 6.8 shows that MPEdge models are slower compared to all the other methods.



(a) Training Accuracy on MNIST

(b) Training Accuracy on CIFAR10

Figure 6.6 Accuracy during training. Colours are assigned according to the GNN. Each curve corresponds to a different run. (a) MNIST: On average, accuracy of MPEdge layers is higher compared to EdgeConv layers. (b) CIFAR10: in this case, the accuracy of MPEdge layers is comparable with EdgeConv layer.

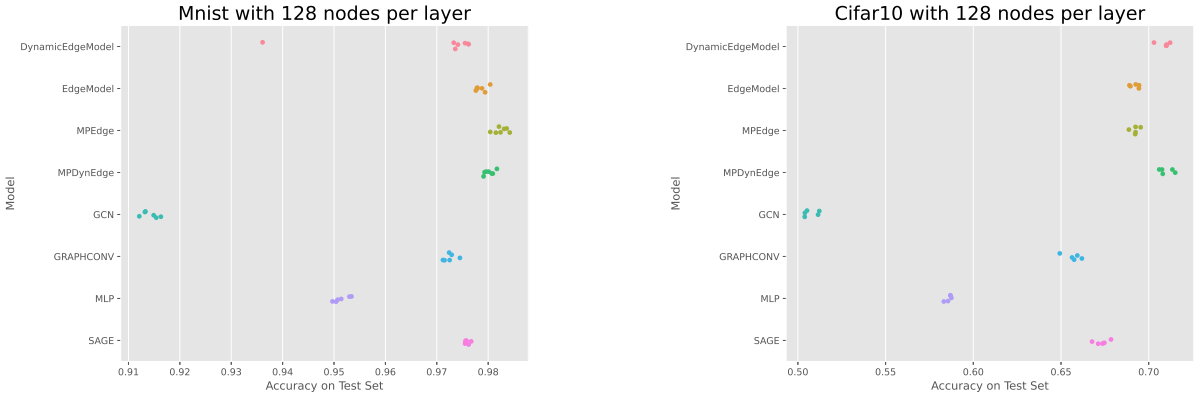
6.2.3 Conclusions

In this chapter, we have introduced a straightforward extension of edge convolution layer based on principles of max-plus algebra. Despite its simplicity, the experiments performed on Graph Classification task show that the proposed MPEdge convolution achieves comparable results and in the case of MNIST better results than the state of the art. Results obtained are promising and in our opinion, it is worth doing further tests of the MPEdge on other tasks such as link prediction or node classification in future works. We should note that this is a first step in the way toward the use of more complex morphological operators for graph processing in the context of deep learning.

TABLE 6.1

COMPARISON WITH STATE-OF-THE-ART CONVOLUTION LAYERS ON GRAPH CLASSIFICATION TASK. WE USED THE MULTI-LABEL CLASSIFICATION ACCURACY AS AN EVALUATION METRIC. BEST MODELS IN RED.

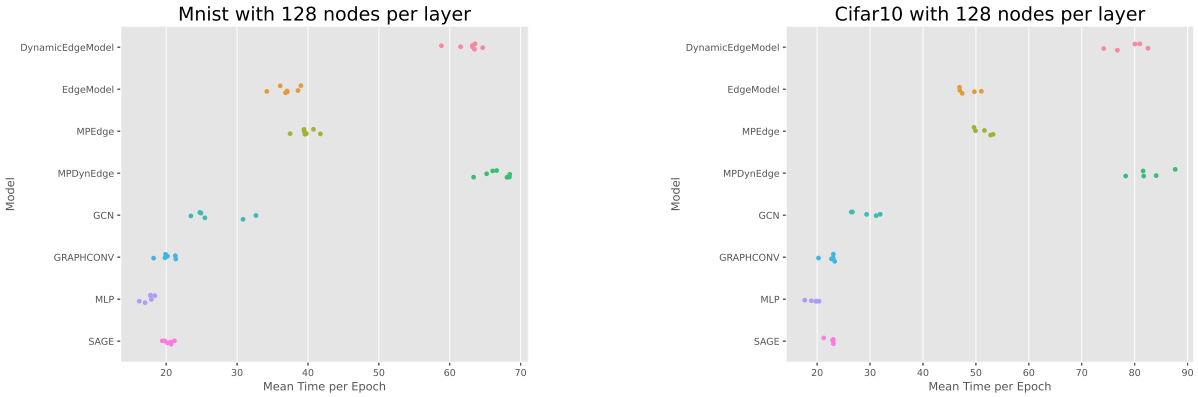
		GRAPH CLASSIFICATION									
		MNIST					CIFAR10				
Model	**L**	**#Param**	**Test Acc ± s.d.**	**Train Acc ± s.d.**	**#Epoch**	**Epoch/Total**	**#Param**	**Test Acc ± s.d.**	**Train Acc ± s.d.**	**#Epoch**	**Epoch/Total**
MLP	3	35594	0.951 ± 0.001	0.955 ± 0.001	123	17.5s/0.59hr	35850	0.586 ± 0.002	0.620 ± 0.004	120	19.3s/0.64hr
EdgeConv	3	68746	0.979 ± 0.001	0.999 ± 0.001	93	37s/0.95hr	69258	0.692 ± 0.003	0.879 ± 0.007	98	48.4s/1.32hr
DynEdgeConv	3	68746	0.969 ± 0.015	0.989 ± 0.022	73	62.6s/1.27hr	69258	$\mathbf{0.709 \pm 0.004}$	0.868 ± 0.014	100	78.8s/2.19hr
MPEdge	3	72202	$\mathbf{0.982 \pm 0.001}$	0.999 ± 0.002	95	39.7s/1.04hr	72714	0.692 ± 0.002	0.872 ± 0.023	80	51.4s/1.15hr
MPDynEdge	3	72202	0.980 ± 0.001	0.999 ± 0.001	81	66.5s/1.50hr	72714	$\mathbf{0.710 \pm 0.004}$	0.851 ± 0.026	89	82.6s/2.04hr
GCN	3	35594	0.914 ± 0.002	0.924 ± 0.002	130	27s/0.98hr	35850	0.507 ± 0.004	0.534 ± 0.007	129	29s/1.03hr
GraphConv	3	68746	0.973 ± 0.001	0.989 ± 0.002	115	20s/0.64hr	69258	0.657 ± 0.005	0.750 ± 0.009	137	22.5s/0.85hr
GraphSAGE	3	68746	0.976 ± 0.000	0.993 ± 0.002	117	20s/0.66hr	69258	0.673 ± 0.004	0.787 ± 0.012	115	22.6s/0.73hr



(a) Test Accuracy on MNIST

(b) Test Accuracy on CIFAR10

Figure 6.7 Accuracy scores on test sets (Higher is better). Each dot is a different run.



(a) Meantime per Epoch on MNIST

(b) Mean Time per Epoch on CIFAR10

Figure 6.8 Mean Time per epoch (Lower is better). Each dot is a different run. DynamicEdgeConv and MPDynEdge are slower because they update the k-NN graph after each convolution.

Conclusions

Hereunder, we list the contributions of this thesis followed by a discussion on future perspectives.

- **Ground Detection.** In the context of point cloud scanning from road environments, we have discussed the importance of ground detection as the first step for more complex tasks as for example object detection and tracking. The high variation of point cloud density represents an issue for classical λ-flat zones based algorithm for ground detection. Therefore, we discussed two strategies to cope with this problem. The first strategy aims to interpolate the information using a polar grid defined on the BEV image. The second approach builds a graph on 3D using the inherent spherical vision of the scanner. Moreover, it combines elevation and point normals to define a weight function on the graph edges helpful to locate horizontal surfaces. The two methods have been tested against the state-of-the-art methods on the SemanticKITTI dataset. For this study, we selected supervised (U-Net) and unsupervised approaches such as RANSAC, CSF. The scores obtained by our methods are comparable with the state-of-the-art. Even though U-Net performs the best in almost all metrics considered, we believe that the low number of hyperparameters required and the interpretability are the strong points of our approaches. Moreover, simpler hardware is required. This is also an important advantage for mass market applications. Finally, it is worth noting a low percentage of points from classes such as cars, pedestrians or buildings are misclassified as ground. The main confusion of propagation methods remains between ground and low vegetation.
- **Road Detection.** We carried out a study on the performance of road detection on low-resolution point clouds. Results have shown a reduction of the performances as expected. Given the intrinsic flatness of the surface, we propose to integrate information with point normals. We use an extremely simple normal calculation approach. It can be efficiently computed starting from the Spherical View image with point radial information. We demonstrate that the use of normals increases the performances in the full resolution case and mitigates the deterioration of the performances while reducing the resolution.
- **MST on Data Streams.** We proposed two novel algorithms for the problem of compute and update a MST for a stream of data. We consider the case of an image decomposed in tiles received over the time. The proposed solutions are based on the decomposition of the edges of the graph in two parts: *stable* and *unstable*. The stable part is composed of edges that will belong to the MST in the future, whatever the content of upcoming tiles. On the other hand, the unstable part contains edges that could be removed from MST as new information arrives. The advantage of this decomposition

is the reduced memory footprint of the algorithm. This allows to treat images of bigger sizes compared to the naive approach. The correctness of the proposed algorithms is proved in the case of morphological segmentation of remote sensing images.

- **Metric Learning for surface retrieval.** We illustrate the solution submitted for the SHREC'20 Track: Retrieval and classification of the surface patches with similar geometric relief. The goal is to estimate the similarities between two objects based only on the patterns on their surfaces, without considering the global shape. Our idea is to look for points of the mesh where the curvature of the object is almost flat, in order to extract a local patch that characterizes the texture of the surface. For each object, we extract multiple patches. Successively, a siamese-network is trained to learn similarities between different patches. Finally, the global similarity between two objects is defined as the minimum value among all similarities between any two patches of the two objects. Our solution was ranked second in the international challenge.
- **Learning Similarity.** In the context of Hierarchical Clustering (HC), we analyze the case in which we have a collection of realizations of a random distribution, and we want to extract a hierarchical clustering for each sample in the collection. The size of samples varies and is not known in advance. Our idea to solve this problem takes into account the continuous relaxation formulation of the Dasgupta functional proposed in Chami et al. (2020), that approximates a hierarchical tree via an embedding of the input points into the Hyperbolic space $\mathbb{B}^2$. However, this formulation assumes to know in advance the number of points and the pairwise-similarities between them. To overcome this issue, the proposed solution aims to learn a similarity function on the input space along with the embedding to the Hyperbolic space. Hence, a triplet loss term is added to the embedding loss. This approach has been validated on five illustrative datasets. In addition, we carried out a comparison with classical HC algorithms. The results obtained are promising and the proposed method showed in many cases good robustness to noise and higher adaptability to different datasets compared with the classical approaches.
- **Morphological Convolutions.** We proposed MPEC a first *Morphological Convolutional Layer* on Graph Neural Network (GNN). The definition is based on EdgeConv, and it uses a morphological weight matrix to implement a trainable permutation invariant aggregate function. The newly proposed layer has been tested on two graph classification benchmarks against the state-of-the-art GNNs. Overall the results are comparable with the state of the art and on MNIST dataset, MPEC shows better performances.

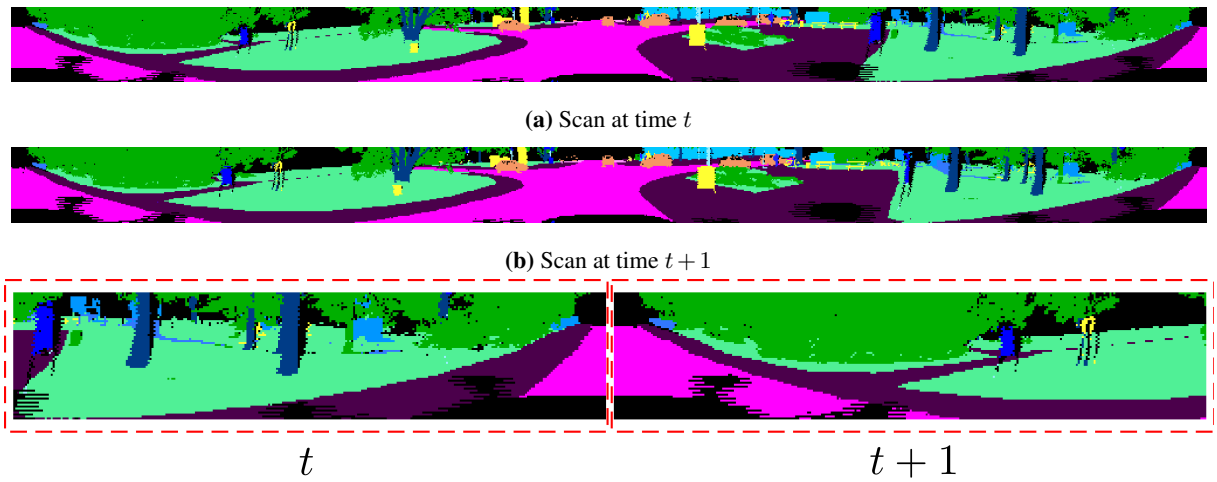
Perspectives

In the following we list the perspectives based on the current work.

- **Point Cloud Sampling** A first direction could be to analyse the effect of temporal aggregation of LIDAR scans on reduced spatial resolution due to subsampling the vertical angle. Furthermore, in Spherical View we upsampled the information inside the network, just before the prediction layer. It could be interesting to upsample the input Spherical View range images, and evaluate the performance of the U-Net model on the original 64×2048 sized Spherical View range image. The up-sampling can be trained end-to-end to achieve successful reconstruction of 64×2048 image

from sub-sampled 32×2048 or 16×2048 images. We have focused our study on road segmentation mainly to limit and understand the effect of subsampling on a geometrically simple case. Future studies could evaluate performance of key classes such as cars, pedestrians, to determine the loss in accuracy on account of subsampling of point clouds.

- **λ-flat-zones on 3D graph** In this method proposed in Section 3.3, in the weight function we measure the ratio between the elevation of the point and the vertical orientation of its normal. Thanks to this ratio horizontal surfaces are contained in big quasi flat zones, while vertical surfaces are shattered in small components. However, we could integrate in the definition of the weight function also information about local neighborhoods of the two points. Namely, it could be interesting to integrate in the weight function information such as planarity, linearirty or sphericity. Since, we use the PCA of the local neighborhood to estimate normals, we can obtain this information without any extra computation cost. Moreover, another important step of the method is the merging of quasi flat zones. Here, RANSAC algorithm is used to evaluate if two quasi flat zones are comparable. Other criteria, based on the maximum, minimum and mean elevation of the two components could be considered.
- **Application of MST Streaming to Point Clouds from Road Enviromnents.** In Chapter 4, we have shown three applications of the algorithm for MST in streaming to segmentation of Remote Sensing images. Morphological segmentation of point-clouds from road environments could be another interesting application for this method. The approach could exploit the Spherical View representation to temporally aggregate multiple scans. Using this representation, we can navigate through the scans along the time as if we were unfolding a photographic film, where each frame of the film corresponds to a single scan (Figure 6.9). In this context we could use for example the streaming version of λ-flat-zone algorithm to find a segmentation of the scene.



(a) Scan at time t

(b) Scan at time $t+1$

(c) Zoom on the frontier between t and $t+1$.

Figure 6.9 (a) Scan at time t and (b) the successive frame at time $t+1$. (c) We can image the two scans as two successive frames of a pelicular film.

- **MST Streaming** Using the algorithms introduced in Chapter 4, it could be possible to implement streaming version of other morphological segmentation algorithms, such as for example Stochastic Watershed (Angulo and Jeulin, 2007).

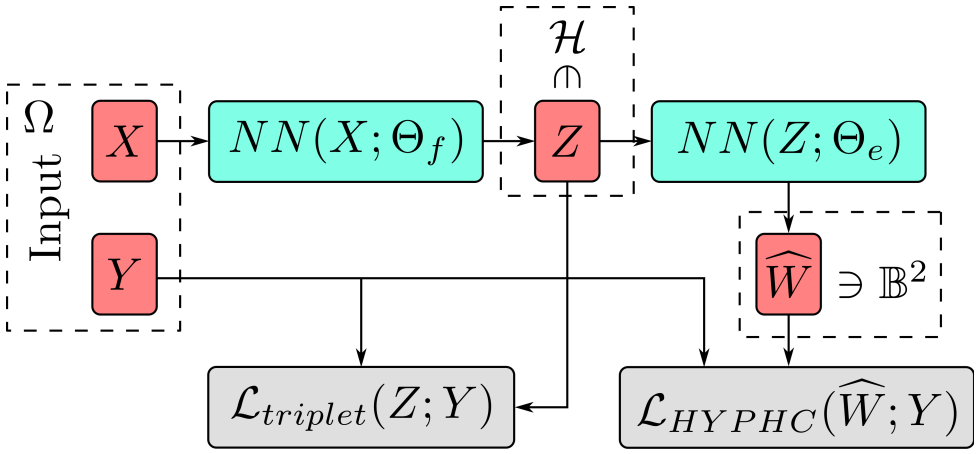


Figure 6.10 Framework that we want to explore is composed of two neural networks (cyan blocks). The first embeds input points into an hidden space $\mathcal{H}$. The optimization of the parameters of the first network is done using a triplet loss. Triplets are generated using labels Y. The second network embed hidden features to Poincaré Disk. In this case parameters are optimized using HypHC Loss defined in Chami et al. (2020).

- **Learning similarities for HC** In Section 5.3.4 of Chapter 5, we have discussed about a wider family of solutions to our problem to investigate. This framework is composed of two neural networks as shown in Figure 6.10. The first network $\mathrm{NN}_{\Theta_f} : \Omega \to \mathcal{H}$ acts as feature extractor. The idea is to map input features into an hidden space $\mathcal{H}$ on which we look for the optimal similarity function. The second network $\mathrm{NN}_{\Theta_e} : \mathcal{H} \to \mathbb{B}^2$ embed the hidden features into the Poincaré Disk. The idea is to disentangle the two components of the loss function and optimize the two components in each space.
- **Learning similarities for HC** The solution proposed in Chapter 5 could be applied to point clouds. In particular, it could be interesting to integrate the proposed framework to already existing DNN solutions for point cloud segmentation. In particular it could be interesting integrate our solution to DGCNN (Wang et al., 2019b).
- **Morphological Graph Network** In Chapter 6, we proposed MPEdge, a first *morphological* graph convolution. The test done on benchmark dataset for Graph classification are promising. Nonetheless, a wider test on other task such as node classification or link prediction should be performed.

References

R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, Nov. 2012. ISSN 0162-8828, 2160-9292. doi: 10.1109/TPAMI.2012.120. URL http://ieeexplore.ieee.org/document/6205760/.

J. Angulo and D. Jeulin. Stochastic watershed segmentation. In *ISMM (1)*, pages 265–276, 2007.

D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 1027–1035, New Orleans, Louisiana, Jan. 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the fourth annual symposium on Computational geometry - SCG '88*, pages 252–257, Urbana-Champaign, Illinois, United States, 1988. ACM Press. ISBN 9780897912709. doi: 10.1145/73393.73419. URL http://portal.acm.org/citation.cfm?doid=73393.73419.

M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *ACM Transactions on Graphics*, 37(4):1–12, Aug. 2018. ISSN 0730-0301, 1557-7368. doi: 10.1145/3197517.3201301. URL https://dl.acm.org/doi/10.1145/3197517.3201301.

D. Bacciu, F. Errica, A. Micheli, and M. Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, Sept. 2020. ISSN 08936080. doi: 10.1016/j.neunet.2020.06.006. URL https://linkinghub.elsevier.com/retrieve/pii/S0893608020302197.

L. Bao, Y. Song, Q. Yang, H. Yuan, and G. Wang. Tree Filtering: Efficient Structure-Preserving Smoothing With a Minimum Spanning Tree. *IEEE Transactions on Image Processing*, 23(2):555–569, Feb. 2014. ISSN 1057-7149, 1941-0042. doi: 10.1109/TIP.2013.2291328. URL http://ieeexplore.ieee.org/document/6665130/.

J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9296–9306, Seoul, Korea (South), Oct. 2019. IEEE. ISBN 9781728148038. doi: 10.1109/ICCV.2019.00939. URL https://ieeexplore.ieee.org/document/9010727/.

S. Beucher and C. Lantuéjoul. Use of watersheds in contour detection. volume 132, 01 1979.

F. M. Bianchi, D. Grattarola, and C. Alippi. Spectral Clustering with Graph Neural Networks for Graph Pooling. In *International Conference on Machine Learning*, pages 874–883. PMLR, Nov. 2020. URL http://proceedings.mlr.press/v119/bianchi20a.html.

N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph theory, 1736-1936*. Clarendon Press, Oxford [Oxfordshire] ; New York, 1986. ISBN 9780198539162.

J. L. Blanco and P. K. Rai. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. https://github.com/jlblancoc/nanoflann, 2014.

S. Blusseau, S. Velasco-Forero, J. Angulo, and I. Bloch. Tropical and Morphological Operators for Signals on Graphs. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1198–1202, Athens, Oct. 2018. IEEE. ISBN 9781479970612. doi: 10.1109/ICIP.2018.8451395. URL https://ieeexplore.ieee.org/document/8451395/.

O. Borůvka. "O jistém problému minimálním" (Über ein Minimalproblem.). Práce moravské přirodovědecké společnosti 3 (1926), 37-58 (1926)., 1926.

D. A. Brannan, M. F. Esplen, and J. Gray. *Geometry*. Cambridge University Press, Cambridge ; New York, 2nd ed edition, 2012. ISBN 9781107647831.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017. ISSN 1558-0792. doi: 10.1109/MSP.2017.2693418.

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]*, May 2014. URL http://arxiv.org/abs/1312.6203. arXiv: 1312.6203.

C. Burstedde, L. C. Wilcox, and O. Ghattas. p4est : Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, Jan. 2011. ISSN 1064-8275, 1095-7197. doi: 10.1137/100791634. URL http://epubs.siam.org/doi/10.1137/100791634.

G. Bécigneul and O.-E. Ganea. Riemannian Adaptive Optimization Methods. *arXiv:1810.00760 [cs, stat]*, Feb. 2019. URL http://arxiv.org/abs/1810.00760. arXiv: 1810.00760.

H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, Seattle, WA, USA, June 2020. IEEE. ISBN 9781728171685. doi: 10.1109/CVPR42600.2020.01164. URL https://ieeexplore.ieee.org/document/9156412/.

L. Caltagirone, S. Scheidegger, L. Svensson, and M. Wahde. Fast LIDAR-based road detection using fully convolutional neural networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1019–1024, Los Angeles, CA, USA, June 2017. IEEE. ISBN 9781509048045. doi: 10.1109/IVS.2017.7995848. URL http://ieeexplore.ieee.org/document/7995848/.

G. Carlsson and F. Mémoli. Characterization, Stability and Convergence of Hierarchical Clustering Methods. *The Journal of Machine Learning Research*, 11:1425–1470, Aug. 2010. ISSN 1532-4435.

I. Chami, A. Gu, V. Chatziafratis, and C. Ré. From Trees to Continuous Embeddings and Back: Hyperbolic Hierarchical Clustering. *arXiv:2010.00402 [cs, stat]*, Oct. 2020. URL http://arxiv.org/abs/2010.00402. arXiv: 2010.00402.

A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012 [cs]*, Dec. 2015. URL http://arxiv.org/abs/1512.03012. arXiv: 1512.03012 version: 1.

C. Y. Chao. Graphs and Algorithms (M. Gondran and M. Minoux). *SIAM Review*, 28(4):583–584, Dec. 1986. ISSN 0036-1445, 1095-7200. doi: 10.1137/1028176. URL http://epubs.siam.org/doi/10.1137/1028176.

R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, Honolulu, HI, July 2017. IEEE. ISBN 9781538604571. doi: 10.1109/CVPR.2017.16. URL http://ieeexplore.ieee.org/document/8099499/.

G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large Scale Online Learning of Image Similarity Through Ranking. *The Journal of Machine Learning Research*, 11:1109–1135, Mar. 2010. ISSN 1532-4435.

L. Chen, J. Yang, and H. Kong. Lidar-histogram for fast road and obstacle detection. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1343–1348, Singapore, Singapore, May 2017. IEEE. ISBN 9781509046331. doi: 10.1109/ICRA.2017.7989159. URL http://ieeexplore.ieee.org/document/7989159/.

G. Chierchia and B. Perret. Ultrametric fitting by gradient descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124004, Dec. 2020. ISSN 1742-5468. doi: 10.1088/1742-5468/abc62d. URL https://iopscience.iop.org/article/10.1088/1742-5468/abc62d.

F. R. K. Chung. *Spectral graph theory*. Number no. 92 in Regional conference series in mathematics. Published for the Conference Board of the mathematical sciences by the American Mathematical Society, Providence, R.I, 1997. ISBN 9780821803158.

T. H. Cormen, editor. *Introduction to algorithms*. MIT Press, Cambridge, Mass, 3rd ed edition, 2009. ISBN 9780262033848 9780262533058. OCLC: ocn311310321.

C. Couprie, L. Grady, L. Najman, and H. Talbot. Power Watershed: A Unifying Graph-Based Optimization Framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(7):1384–1399, July 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.200. URL http://ieeexplore.ieee.org/document/5639015/.

J. Cousty, G. Bertrand, L. Najman, and M. Couprie. Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(8):1362–1374, Aug. 2009. ISSN 1939-3539. doi: 10.1109/TPAMI.2008.173.

J. Cousty, L. Najman, F. Dias, and J. Serra. Morphological filtering on graphs. *Computer Vision and Image Understanding*, 117(4):370–385, Apr. 2013. ISSN 10773142. doi: 10.1016/j.cviu.2012.08.016. URL https://linkinghub.elsevier.com/retrieve/pii/S107731421200183X.

J. Cousty, L. Najman, Y. Kenmochi, and S. Guimarães. Hierarchical Segmentations with Graphs: Quasi-flat Zones, Minimum Spanning Trees, and Saliency Maps. *Journal of Mathematical Imaging and Vision*, 60(4):479–502, May 2018. ISSN 0924-9907, 1573-7683. doi: 10.1007/s10851-017-0768-7. URL http://link.springer.com/10.1007/s10851-017-0768-7.

S. Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 118–127, Cambridge MA USA, June 2016. ACM. ISBN 9781450341325. doi: 10.1145/2897518.2897527. URL https://dl.acm.org/doi/10.1145/2897518.2897527.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3844–3852, Barcelona, Spain, Dec. 2016. Curran Associates Inc. ISBN 9781510838819.

I. del Pino, V. Vaquero, B. Masini, J. Solà, F. Moreno-Noguer, A. Sanfeliu, and J. Andrade-Cetto. Low Resolution Lidar-Based Multi-Object Tracking for Driving Applications. In A. Ollero, A. Sanfeliu, L. Montano, N. Lau, and C. Cardeira, editors, *ROBOT 2017: Third Iberian Robotics Conference*, volume 693, pages 287–298. Springer International Publishing, Cham, 2018. ISBN 9783319708324 9783319708331. doi: 10.1007/978-3-319-70833-1_24. URL http://link.springer.com/10.1007/978-3-319-70833-1_24.

J. Demantké, C. Mallet, N. David, and B. Vallet. DIMENSIONALITY BASED SCALE SELECTION IN 3D LIDAR POINT CLOUDS. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-5/W12:97–102, Sept. 2012. ISSN 2194-9034. doi: 10.5194/isprsarchives-XXXVIII-5-W12-97-2011. URL https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XXXVIII-5-W12/97/2011/.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the *EM* Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, Sept. 1977. ISSN 00359246. doi: 10.1111/j.2517-6161.1977.tb01600.x. URL https://onlinelibrary.wiley.com/doi/10.1111/j.2517-6161.1977.tb01600.x.

F. Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019a.

F. Diehl. Edge Contraction Pooling for Graph Neural Networks. *arXiv:1905.10990 [cs, stat]*, May 2019b. URL http://arxiv.org/abs/1905.10990. arXiv: 1905.10990.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959. ISSN 0029-599X, 0945-3245. doi: 10.1007/BF01386390. URL http://link.springer.com/10.1007/BF01386390.

V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking Graph Neural Networks. *arXiv:2003.00982 [cs, stat]*, July 2020. URL http://arxiv.org/abs/2003.00982. arXiv: 2003.00982 version: 3.

L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, Sept. 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000022288.19776.77. URL http://link.springer.com/10.1023/B:VISI.0000022288.19776.77.

M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981. ISSN 0001-0782, 1557-7317. doi: 10.1145/358669.358692. URL https://dl.acm.org/doi/10.1145/358669.358692.

J. H. Friedman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, Sept. 1977. ISSN 0098-3500, 1557-7295. doi: 10.1145/355744.355745. URL https://dl.acm.org/doi/10.1145/355744.355745.

J. Fritsch, T. Kühnl, and A. Geiger. A new performance measure and evaluation benchmark for road detection algorithms. pages 1693–1700, Oct. 2013. doi: 10.1109/ITSC.2013.6728473. ISSN: 2153-0017.

O. Gallo, R. Manduchi, and A. Rafii. CC-RANSAC: Fitting planes in the presence of multiple surfaces in range data. *Pattern Recognition Letters*, 32(3):403–410, Feb. 2011. ISSN 01678655. doi: 10.1016/j.patrec.2010.10.009. URL https://linkinghub.elsevier.com/retrieve/pii/S0167865510003557.

H. Gao and S. Ji. Graph U-Nets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2021.3081010. URL https://ieeexplore.ieee.org/document/9432709/.

A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012. doi: 10.1109/CVPR.2012.6248074. ISSN: 1063-6919.

L. Gigli, B. R. Kiran, T. Paul, A. Serna, N. Vemuri, B. Marcotegui, and S. Velasco-Forero. Road Segmentation on low resolution Lidar point clouds for autonomous vehicles. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-2-2020:335–342, Aug. 2020a. ISSN 2194-9050. doi: 10.5194/isprs-annals-V-2-2020-335-2020. URL https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/V-2-2020/335/2020/.

L. Gigli, S. Velasco-Forero, and B. Marcotegui. On minimum spanning tree streaming for hierarchical segmentation. *Pattern Recognition Letters*, 138:155–162, Oct. 2020b. ISSN 01678655. doi: 10.1016/j.patrec.2020.07.006. URL https://linkinghub.elsevier.com/retrieve/pii/S016786552030252X.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for Quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1263–1272, Sydney, NSW, Australia, Aug. 2017. JMLR.org.

Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep Learning for 3D Point Clouds: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2020.3005434. URL https://ieeexplore.ieee.org/document/9127813/.

T. Hackel, J. D. Wegner, and K. Schindler. Fast semantic segmentation of 3D point clouds with strongly varying density. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-3:177–184, June 2016. ISSN 2194-9050. doi: 10.5194/isprs-annals-III-3-177-2016. URL https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/III-3/177/2016/.

T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys. SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-1/W1:91–98, May 2017. ISSN 2194-9050. doi: 10.5194/isprs-annals-IV-1-W1-91-2017. URL https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-1-W1/91/2017/.

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs. *arXiv:1706.02216 [cs, stat]*, Sept. 2018. URL http://arxiv.org/abs/1706.02216. arXiv: 1706.02216.

D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, Mar. 2011. ISSN 10635203. doi: 10.1016/j.acha.2010.04.005. URL https://linkinghub.elsevier.com/retrieve/pii/S1063520310000552.

C. He, J. Chang, and X. Chen. Using the Triangle Inequality to Accelerate TTSAS Cluster Algorithm. In *2010 International Conference on Electrical and Control Engineering*, pages 2507–2510, Wuhan, June 2010. IEEE. ISBN 9781424468805 9781424468812. doi: 10.1109/iCECE.2010.620. URL https://ieeexplore.ieee.org/document/5630828/.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE. ISBN 9781467388511. doi: 10.1109/CVPR.2016.90. URL http://ieeexplore.ieee.org/document/7780459/.

J. Hernandez and B. Marcotegui. Point cloud segmentation towards urban ground modeling. In *2009 Joint Urban Remote Sensing Event*, pages 1–5, May 2009. doi: 10.1109/URS.2009.5137562. ISSN: 2334-0932.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, Nov. 1997. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1997.9.8.1735. URL https://direct.mit.edu/neco/article/9/8/1735-1780/6109.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques - SIGGRAPH '92*, pages 71–78, Not Known, 1992. ACM Press. ISBN 9780897914796. doi: 10.1145/133994.134011. URL http://portal.acm.org/citation.cfm?doid=133994.134011.

Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11105–11114, Seattle, WA, USA, June 2020. IEEE. ISBN 9781728171685. doi: 10.1109/CVPR42600.2020.01112. URL https://ieeexplore.ieee.org/document/9156466/.

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, Dec. 1985. ISSN 0176-4268, 1432-1343. doi: 10.1007/BF01908075. URL http://link.springer.com/10.1007/BF01908075.

S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456, Lille, France, July 2015. JMLR.org.

C. Jardine, N. Jardine, and R. Sibson. The structure and construction of taxonomic hierarchies. *Mathematical Biosciences*, 1(2):173–179, 1967. doi: 10.1016/0025-5564(67)90032-6.

M. Jaritz, R. D. Charette, E. Wirbel, X. Perrotton, and F. Nashashibi. Sparse and Dense Data with CNNs: Depth Completion and Semantic Segmentation. In *2018 International Conference on 3D Vision (3DV)*, pages 52–60, Verona, Sept. 2018. IEEE. ISBN 9781538684252. doi: 10.1109/3DV.2018.00017. URL https://ieeexplore.ieee.org/document/8490955/.

V. Jarník. O jistém problému minimálním (Über ein Minimalproblem.). Práce moravské přirodovědecké společnosti 6, 57-63 (1931)., 1931.

Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug. 2000. ISSN 01628828. doi: 10.1109/34.868688. URL http://ieeexplore.ieee.org/document/868688/.

S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, Sept. 1967. doi: 10.1007/BF02289588.

C. Joshi. Transformers are graph neural networks. *The Gradient*, 2020.

D. Jungnickel. *Graphs, networks and algorithms*. Number volume 5 in Algorithms and computation in mathematics. Springer, Berlin Heidelberg New York Dordrecht London, fourth edition edition, 2013. ISBN 9783642322785 9783642322778 9783642436642.

W. Kim. Parallel clustering algorithms: survey. *Parallel Algorithms, Spring*, 34:43, 2009.

T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*, Feb. 2017. URL http://arxiv.org/abs/1609.02907. arXiv: 1609.02907.

J. Kleinberg. An impossibility theorem for clustering. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, pages 463–470, Cambridge, MA, USA, Jan. 2002. MIT Press.

B. Knyazev, G. W. Taylor, and M. Amer. Understanding Attention and Generalization in Graph Neural Networks. *Advances in Neural Information Processing Systems*, 32, 2019. URL https://papers.nips.cc/paper/2019/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html.

J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–48, Jan. 1956. ISSN 0002-9939. doi: 10.1090/S0002-9939-1956-0078686-7. URL http://www.ams.org/jourcgi/jour-getitem?pii=S0002-9939-1956-0078686-7.

L. Landrieu and M. Simonovsky. Large-Scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, Salt Lake City, UT, June 2018. IEEE. ISBN 9781538664209. doi: 10.1109/CVPR.2018.00479. URL https://ieeexplore.ieee.org/document/8578577/.

J. Lee, I. Lee, and J. Kang. Self-Attention Graph Pooling. In *International Conference on Machine Learning*, pages 3734–3743. PMLR, May 2019. URL http://proceedings.mlr.press/v97/lee19c.html.

T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, Oct. 2017. doi: 10.1109/ICCV.2017.324. ISSN: 2380-7504.

S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, Mar. 1982. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489. URL http://ieeexplore.ieee.org/document/1056489/.

Y. Lyu, L. Bai, and X. Huang. ChipNet: Real-Time LiDAR Processing for Drivable Region Segmentation on an FPGA. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(5):1769–1779, May 2019. ISSN 1549-8328, 1558-0806. doi: 10.1109/TCSI.2018.2881162. URL https://ieeexplore.ieee.org/document/8580596/.

G. J. McLachlan and T. Krishnan. *The EM algorithm and extensions*. 2008. ISBN 9780470191606 9780470191613. URL http://www.123library.org/book_details/?id=21298. OCLC: 228427283.

D. J. Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic., 1980.

X.-L. Meng and D. Van Dyk. The EM Algorithm-an Old Folk-song Sung to a Fast New Tune. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(3):511–567, 1997. ISSN 13697412. doi: 10.1111/1467-9868.00082. URL https://onlinelibrary.wiley.com/doi/10.1111/1467-9868.00082.

F. Meyer. Minimum Spanning Forests for Morphological Segmentation. In J. Serra and P. Soille, editors, *Mathematical Morphology and Its Applications to Image Processing*, Computational Imaging and Vision, pages 77–84. Springer Netherlands, Dordrecht, 1994. ISBN 9789401110402. doi: 10.1007/978-94-011-1040-2_11. URL https://doi.org/10.1007/978-94-011-1040-2_11.

F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, Sept. 1990. ISSN 10473203. doi: 10.1016/1047-3203(90)90014-M. URL https://linkinghub.elsevier.com/retrieve/pii/104732039090014M.

A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4213–4220, Macau, China, Nov. 2019. IEEE. ISBN 9781728140049. doi: 10.1109/IROS40897.2019.8967762. URL https://ieeexplore.ieee.org/document/8967762/.

K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 909–918, Long Beach, CA, USA, June 2019. IEEE. ISBN 9781728132938. doi: 10.1109/CVPR.2019.00100. URL https://ieeexplore.ieee.org/document/8954053/.

N. Monath, M. Zaheer, D. Silva, A. McCallum, and A. Ahmed. Gradient-based Hierarchical Clustering using Continuous Representations of Trees in Hyperbolic Space. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 714–722, Anchorage, AK, USA, July 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330997. URL https://doi.org/10.1145/3292500.3330997.

F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, Honolulu, HI, July 2017. IEEE. ISBN 9781538604571. doi: 10.1109/CVPR.2017.576. URL http://ieeexplore.ieee.org/document/8100059/.

C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609, July 2019. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33014602. URL https://aaai.org/ojs/index.php/AAAI/article/view/4384.

E. Moscoso Thompson, S. Biasotti, A. Giachetti, C. Tortorici, N. Werghi, A. S. Obeid, S. Berretti, H.-P. Nguyen-Dinh, M.-Q. Le, H.-D. Nguyen, M.-T. Tran, L. Gigli, S. Velasco-Forero, B. Marcotegui, I. Sipiran, B. Bustos, I. Romanelis, V. Fotis, G. Arvanitis, K. Moustakas, E. Otu, R. Zwiggelaar, D. Hunter, Y. Liu, Y. Arteaga, and R. Luxman. SHREC 2020: Retrieval of digital surfaces with similar geometric reliefs. *Computers & Graphics*, 91:199–218, Oct. 2020. ISSN 00978493. doi: 10.1016/j.cag.2020.07.011. URL https://linkinghub.elsevier.com/retrieve/pii/S0097849320301138.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, Cambridge, Mass., 2012. ISBN 9780262305242 9780262304320. URL http://site.ebrary.com/id/10597102. OCLC: 810414751.

L. Najman, J. Cousty, and B. Perret. Playing with Kruskal: Algorithms for Morphological Trees in Edge-Weighted Graphs. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C.

Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, C. L. L. Hendriks, G. Borgefors, and R. Strand, editors, *Mathematical Morphology and Its Applications to Signal and Image Processing*, volume 7883, pages 135–146. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 9783642382932 9783642382949. doi: 10.1007/978-3-642-38294-9_12. URL http://link.springer.com/10.1007/978-3-642-38294-9_12.

Y. Nakagawa, H. Uchiyama, H. Nagahara, and R.-I. Taniguchi. Estimating Surface Normals with Depth Image Gradients for Fast and Accurate Registration. In *2015 International Conference on 3D Vision*, pages 640–647, Lyon, France, Oct. 2015. IEEE. ISBN 9781467383325. doi: 10.1109/3DV.2015.80. URL http://ieeexplore.ieee.org/document/7335535/.

M. Naumov and T. Moon. Parallel spectral graph partitioning. Technical report, NVIDIA Technical Report, NVR-2016-001, 2016.

V. Olman, Fenglou Mao, Hongwei Wu, and Ying Xu. Parallel Clustering Algorithm for Large Data Sets with Applications in Bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):344–352, Apr. 2009. ISSN 1545-5963. doi: 10.1109/TCBB.2007.70272. URL http://ieeexplore.ieee.org/document/4524229/.

F. Oniga, S. Nedevschi, M. M. Meinecke, and T. B. To. Road Surface and Obstacle Detection Based on Elevation Maps from Dense Stereo. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 859–865, Sept. 2007. doi: 10.1109/ITSC.2007.4357734. ISSN: 2153-0017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research*, 12(null): 2825–2830, Nov. 2011. ISSN 1532-4435.

R. C. Prim. Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal*, 36(6):1389–1401, Nov. 1957. ISSN 00058580. doi: 10.1002/j.1538-7305.1957.tb01515.x. URL https://ieeexplore.ieee.org/document/6773228.

W. M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. ISSN 0162-1459. doi: 10.2307/2284239. URL https://www.jstor.org/stable/2284239.

O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, volume 9351, pages 234–241. Springer International Publishing, Cham, 2015. ISBN 9783319245737 9783319245744. doi: 10.1007/978-3-319-24574-4_28. URL http://link.springer.com/10.1007/978-3-319-24574-4_28.

X. Roynard, J.-E. Deschaud, and F. Goulette. Fast and robust segmentation and classification for change detection in urban point clouds. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B3:693–699, June 2016. ISSN 2194-9034. doi: 10.5194/isprs-archives-XLI-B3-693-2016. URL https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B3/693/2016/.

X. Roynard, J.-E. Deschaud, and F. Goulette. Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557, May 2018. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364918767506. URL http://journals.sagepub.com/doi/10.1177/0278364918767506.

F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, Jan. 2009. ISSN 1045-9227, 1941-0093. doi: 10.1109/TNN.2008.2005605. URL http://ieeexplore.ieee.org/document/4700287/.

R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, June 2007. ISSN 0167-7055, 1467-8659. doi: 10.1111/j.1467-8659.2007.01016.x. URL https://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2007.01016.x.

A. Serna and B. Marcotegui. Urban accessibility diagnosis from mobile laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 84:23–32, Oct. 2013. ISSN 09242716. doi: 10.1016/j.isprsjprs.2013.07.001. URL https://linkinghub.elsevier.com/retrieve/pii/S0924271613001585.

A. Serna and B. Marcotegui. Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:243–255, July 2014. ISSN 09242716. doi: 10.1016/j.isprsjprs.2014.03.015. URL https://linkinghub.elsevier.com/retrieve/pii/S0924271614000872.

M. Simonovsky and N. Komodakis. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29–38, Honolulu, HI, July 2017. IEEE. ISBN 9781538604571. doi: 10.1109/CVPR.2017.11. URL http://ieeexplore.ieee.org/document/8099494/.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, Apr. 2015. URL http://arxiv.org/abs/1409.1556. arXiv: 1409.1556 version: 6.

K. Sirohi, R. Mohan, D. Büscher, W. Burgard, and A. Valada. Efficientlps: Efficient lidar panoptic segmentation. *arXiv preprint arXiv:2102.08009*, 2021.

P. Soille. Constrained connectivity for hierarchical image partitioning and simplification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1132–1145, July 2008. ISSN 1939-3539. doi: 10.1109/TPAMI.2007.70817.

P. Soille. Preventing Chaining through Transitions While Favouring It within Homogeneous Regions. In P. Soille, M. Pesaresi, and G. K. Ouzounis, editors, *Mathematical Morphology and Its Applications to Image and Signal Processing*, volume 6671, pages 96–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 9783642215681 9783642215698. doi: 10.1007/978-3-642-21569-8_9. URL http://link.springer.com/10.1007/978-3-642-21569-8_9.

Soilán, Sánchez-Rodríguez, Río-Barral, Perez-Collazo, Arias, and Riveiro. Review of Laser Scanning Technologies and Their Applications for Road and Railway Infrastructure Monitoring. *Infrastructures*, 4(4):58, Sept. 2019. ISSN 2412-3811. doi: 10.3390/infrastructures4040058. URL https://www.mdpi.com/2412-3811/4/4/58.

Sun Chung and A. Condon. Parallel implementation of Bouvka's minimum spanning tree algorithm. In *Proceedings of International Conference on Parallel Processing*, pages 302–308, Honolulu, HI, USA, 1996. IEEE Comput. Soc. Press. ISBN 9780818672552. doi: 10.1109/IPPS.1996.508073. URL http://ieeexplore.ieee.org/document/508073/.

A. Taeihagh and H. S. M. Lim. Governing autonomous vehicles: emerging responses for safety, liability, privacy, cybersecurity, and industry risks. *Transport Reviews*, 39(1):103–128, Jan. 2019. ISSN 0144-1647, 1464-5327. doi: 10.1080/01441647.2018.1494640. URL https://www.tandfonline.com/doi/full/10.1080/01441647.2018.1494640.

H. Thomas, F. Goulette, J.-E. Deschaud, B. Marcotegui, and Y. LeGall. Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods. In *2018 International Conference on 3D Vision (3DV)*, pages 390–398, Verona, Sept. 2018. IEEE. ISBN 9781538684252. doi: 10.1109/3DV.2018.00052. URL https://ieeexplore.ieee.org/document/8490990/.

H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6410–6419, Seoul, Korea (South), Oct. 2019. IEEE. ISBN 9781728148038. doi: 10.1109/ICCV.2019.00651. URL https://ieeexplore.ieee.org/document/9010002/.

L. T. Triess, D. Peter, C. B. Rist, and J. M. Zollner. Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1116–1121, Las Vegas, NV, USA, Oct. 2020. IEEE. ISBN 9781728166735. doi: 10.1109/IV47402.2020.9304631. URL https://ieeexplore.ieee.org/document/9304631/.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6000–6010, Long Beach, California, USA, Dec. 2017. Curran Associates Inc. ISBN 9781510860964.

M. Velas, M. Spanel, M. Hradis, and A. Herout. CNN for very fast ground segmentation in velodyne LiDAR data. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 97–103, Torres Vedras, Apr. 2018. IEEE. ISBN 9781538652213. doi: 10.1109/ICARSC.2018.8374167. URL https://ieeexplore.ieee.org/document/8374167/.

S. Velasco-Forero and J. Angulo. Nonlinear Operators on Graphs via Stacks. In F. Nielsen and F. Barbaresco, editors, *Geometric Science of Information*, volume 9389, pages 654–663. Springer International Publishing, Cham, 2015. ISBN 9783319250397 9783319250403. doi: 10.1007/978-3-319-25040-3_70. URL http://link.springer.com/10.1007/978-3-319-25040-3_70.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *arXiv:1710.10903 [cs, stat]*, Feb. 2018. URL http://arxiv.org/abs/1710.10903. arXiv: 1710.10903 version: 3.

L. Vincent. Graphs and mathematical morphology. *Signal Processing*, 16(4):365–388, Apr. 1989. ISSN 01651684. doi: 10.1016/0165-1684(89)90031-5. URL https://linkinghub.elsevier.com/retrieve/pii/0165168489900315.

N. X. Vinh, J. Epps, and J. Bailey. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *The Journal of Machine Learning Research*, 11: 2837–2854, Dec. 2010. ISSN 1532-4435.

A.-V. Vo, L. Truong-Hong, D. F. Laefer, and M. Bertolotto. Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100, June 2015. ISSN 09242716. doi: 10.1016/j.isprsjprs.2015.01.011. URL https://linkinghub.elsevier.com/retrieve/pii/S0924271615000283.

U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec. 2007. ISSN 1573-1375. doi: 10.1007/s11222-007-9033-z. URL https://doi.org/10.1007/s11222-007-9033-z.

G. Vosselman, editor. *Airborne and terrestrial laser scanning*. Whittles Publ, Caithness, repr edition, 2011. ISBN 9781439827987 9781904445876.

D. Wang and Y. Wang. An improved cost function for hierarchical cluster trees. *Journal of Computational Geometry*, pages 283–331 Pages, Aug. 2020. doi: 10.20382/JOCG.V11I1A11. URL https://jocg.org/index.php/jocg/article/view/3101.

L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan. Graph Attention Convolution for Point Cloud Semantic Segmentation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10288–10297, Long Beach, CA, USA, June 2019a. IEEE. ISBN 9781728132938. doi: 10.1109/CVPR.2019.01054. URL https://ieeexplore.ieee.org/document/8954040/.

Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics*, 38(5):1–12, Nov. 2019b. ISSN 0730-0301, 1557-7368. doi: 10.1145/3326362. URL https://dl.acm.org/doi/10.1145/3326362.

M. Weinmann, S. Urban, S. Hinz, B. Jutzi, and C. Mallet. Distinctive 2D and 3D features for automated large-scale scene analysis in urban areas. *Computers & Graphics*, 49:47–57, June 2015. ISSN 00978493. doi: 10.1016/j.cag.2015.01.006. URL https://linkinghub.elsevier.com/retrieve/pii/S0097849315000072.

B. Y. Wu and K.-M. Chao. *Spanning Trees and Optimization Problems*. Chapman and Hall/CRC, 0 edition, Jan. 2004. ISBN 9780203497289. doi: 10.1201/9780203497289. URL https://www.taylorfrancis.com/books/9780203497289.

D. Xu and Y. Tian. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, 2 (2):165–193, June 2015. ISSN 2198-5804, 2198-5812. doi: 10.1007/s40745-015-0040-1. URL http://link.springer.com/10.1007/s40745-015-0040-1.

R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3): 645–678, May 2005. ISSN 1941-0093. doi: 10.1109/TNN.2005.845141.

Y. Xu, V. Olman, and D. Xu. Minimum spanning trees for gene expression data clustering. *Genome Informatics*, 12:24–33, 2001.

D. Yan, L. Huang, and M. I. Jordan. Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 907, Paris, France, 2009. ACM Press. ISBN 9781605584959. doi: 10.1145/1557019.1557118. URL http://portal.acm.org/citation.cfm?doid=1557019.1557118.

R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 4805–4815, Montréal, Canada, Dec. 2018. Curran Associates Inc.

C. Yuksel. Sample Elimination for Generating Poisson Disk Sample Sets. *Computer Graphics Forum*, 34 (2):25–32, May 2015. ISSN 01677055. doi: 10.1111/cgf.12538. URL https://onlinelibrary.wiley.com/doi/10.1111/cgf.12538.

R. B. Zadeh and S. Ben-David. A Uniqueness Theorem for Clustering. *arXiv:1205.2600 [cs]*, May 2012. URL http://arxiv.org/abs/1205.2600. arXiv: 1205.2600.

F. Zanoguera and F. Meyer. On the implementation of non-separable vector levelings. *Proc. of VIth ISMM, Sydney, CSIRO*, pages 369–377, 2002.

M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, June 2010. doi: 10.1109/CVPR.2010.5539957. ISSN: 1063-6919.

W. Zhang, J. Qi, P. Wan, H. Wang, D. Xie, X. Wang, and G. Yan. An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation. *Remote Sensing*, 8(6):501, June 2016. ISSN 2072-4292. doi: 10.3390/rs8060501. URL http://www.mdpi.com/2072-4292/8/6/501.

Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, Boston, MA, USA, June 2015. IEEE. ISBN 9781467369640. doi: 10.1109/CVPR.2015.7298801. URL http://ieeexplore.ieee.org/document/7298801/.

RÉSUMÉ

Les graphes sont de puissantes structures mathématiques représentant un ensemble d'objets et les relations sous-jacentes entre eux. Ils sont de plus en plus populaires, en particulier dans l'analyse hiérarchique des images ou des nuages de points 3D. Les hiérarchies sont très répandues car elles nous permettent d'organiser efficacement l'information et d'analyser les problèmes à différents niveaux de détail. Dans cette thèse, nous abordons les sujets suivants :

De nombreuses approches hiérarchiques morphologiques s'appuient sur l'arbre de poids minimum (MST). Nous proposons un algorithme pour le calcul du MST en streaming reposant sur une stratégie de décomposition des graphes. Grâce à cette décomposition, des images plus grandes peuvent être traitées ou peuvent bénéficier d'une information partielle fiable alors que l'image entière n'est pas encore disponible. Les récents développements du lidar permettent d'acquérir des nuages de points 3D précis et à grande échelle. De nombreuses applications, telles que la surveillance des infrastructures, l'urbanisme, la conduite autonome, l'agriculture de précision, pour n'en citer que quelques-unes, sont en cours de développement. Nous introduisons un algorithme de détection du sol et le comparons à l'état de l'art. L'impact de la réduction de la densité des nuages de points avec des scanners à faible coût est étudié. Enfin, dans de nombreuses méthodes hiérarchiques, les similarités entre les points sont données en entrée. Cependant, la métrique utilisée pour calculer les similitudes influence la qualité des résultats. Nous abordons l'apprentissage de la métrique comme un outil complémentaire qui contribue à améliorer la qualité des hiérarchies. Nous démontrons les capacités de ces méthodes dans deux contextes. Le premier, une classification de la texture des surfaces 3D, classée deuxième dans une tâche du défi international SHREC'20. Le second permet d'apprendre la fonction de similarité ainsi que la hiérarchie optimale, dans une formulation continue dans l'espace des caractéristiques.

MOTS CLÉS

Théorie de graphes, Hiérarchies, Segmentation, Traitement d'images, Apprentissage.

ABSTRACT

Graphs are powerful mathematical structures representing a set of objects and the underlying links between pairs of objects somehow related. They are becoming increasingly popular in data science in general and in particular in image or 3D point cloud analysis. Among the wide spectra of applications, they are involved in most of the hierarchical approaches. Hierarchies are particularly important because they allow us to efficiently organize the information required and to analyze the problems at different levels of detail. In this thesis, we address the following topics.

Many morphological hierarchical approaches rely on the Minimum Spanning Tree (MST). We propose an algorithm for MST computation in streaming based on a graph decomposition strategy. Thanks to this decomposition, larger images can be processed or can benefit from partial reliable information while the whole image is not completely available.

Recent LiDAR developments are able to acquire large-scale and precise 3D point clouds. Many applications, such as infrastructure monitoring, urban planning, autonomous driving, precision forestry, environmental assessment, archaeological discoveries, to cite a few, are under development nowadays. We introduce a ground detection algorithm and compare it with the state of the art. The impact of reducing the point cloud density with low-cost scanners is studied, in the context of an autonomous driving application.

Finally, in many hierarchical methods similarities between points are given as input. However, the metric used to compute similarities influences the quality of the final results. We exploit metric learning as a complementary tool that helps to improve the quality of hierarchies. We demonstrate the capabilities of these methods in two contexts. The first one, a texture classification of 3D surfaces. Our approach ranked second in a task organized by SHREC'20 international challenge. The second one learning the similarity function together with the optimal hierarchical clustering, in a continuous feature-based hierarchical clustering formulation.

KEYWORDS

Graph Theory, Hierarchical clustering, Segmentation, Image processing, Machine Learning, Point Clouds