POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science in Computer Science Engineering

Landform identification from surface networks

The case of mountain peaks identification from surface networks

Supervisor: Piero Fraternali
Co-supervisor: Nahime Torres

Master Thesis of:
Varga Florin, 864994

Academic Year 2018-2019

Aknowledgements

I would like to thank my supervisor, the Professor Piero Fraternali, for sharing his broad knowledge and guiding me through all the steps for completing this work. I would like also to highlight how this thesis wouldn't be possible without the contribution of the co-supervisor Rocio Nahime Torres which helped me completing and understanding the key points.

I dedicate this work to my family for always supporting me and for their sustain through all these years.

I thank also my friends and all the people which shared this path with me and have always encouraged me.

Contents

List of Figures

List of Tables

Sommario

La classificazione della morfologia della superficie terrestre è essenziale per la comprensione di molti processi che occorrono nel pianeta ed è parte di diversi studi multidisciplinari. Recenti studi dimostrano come l'estrazione delle forme morfologiche sia un'area di studio in continua evoluzione, supportata anche dall'incremento della disponibilità di dati ad alta risoluzione, tra cui i Modelli Digitali di Elevazione (DEM). Un DEM è una rappresentazione della superficie terrestre attraverso una griglia i cui valori sono cono costituiti dall'elevazione rilevata nell'area di interesse. Questi modelli possono essere sfruttati per la individuazione delle proprietà di interesse sia da algoritmi euristici che da quelli basati sull'apprendimento automatico dai dati. Per poter raggiungere buoni livelli sia di precisione che di efficienza, la scelta del tipo di rappresentazione dei dati è di vitale importanza nella progettazione dei metodi che sfruttano i DEM. Una possibile raffigurazione sono le cosiddette *surface network*, ovvero reti delle superfici, che hanno dimostrato di essere efficaci in diversi studi topologici. Diversamente dai precedenti modelli di apprendimento basati dui dati, che codificano i DEM come immagini e applicano le ormai consolidate tecniche convolutive, in questa tesi esploriamo l'applicabilità del Machine Learning sui grafi per il riconoscimento automatico delle forme morfologiche dalle reti delle superfici. In particolare vedremo i metodi per l'identificazione delle sommità delle montagne, che imparano dalle surface networks e da una base dati definita come "gold standard", cioè contente le coordinate geografiche di montagne note. Il modello è stato allenato e testato con i Modelli Digitali di Elevazione e le montagne note della Svizzera.

Abstract

The classification of the Earth surface into landforms is essential for understanding many physical processes that occur in the planet and is the focus of multi-disciplinary studies. In the recent years, automatic landform extraction has emerged as a promising research area, supported by the increasing availability of high resolution Digital Elevation Models (DEMs). DEMs comprise a grid of elevation values, to which different heuristic or data-driven algorithms can be applied for characterizing the terrain features of interest. The choice of the data representation is of primary importance in the design of DEM data analysis approaches, to achieve a landform extraction method that is both efficient and precise. One such representation, the *surface network*, has proved effective for many topological studies. Unlike previous data-driven leaning-based methods, which encode DEMs as images and apply standard convolution operators, in this paper we explore the suitability of Machine Learning on graphs for the automatic recognition of landforms from surface networks. We discuss a method for identifying mountain summits, which learns from a surface network and from a gold standard data set containing the coordinates of peaks in a region. The model has been trained and tested with Switzerland DEM and mountain summit data.

Chapter 1

Introduction

Analyzing and mapping the Earth surface is an important task for a great variety of sciences, such as hydrology, morphometry and morphology, and for applications such as environment monitoring and urban planning. With the proliferation of digital imagery and of its derived products, such as Digital Elevation Models (DEMs), the automatic analysis of the Earth surface with computer-aided tools has become the de facto standard and several research works have proposed methods for extracting landforms from DEM data automatically [1] [2][3] [4]. Among the landforms, mountains have attracted substantial research efforts, due to their prominent role in hydrogeological risk and water supply. The problem of characterizing mountains is well-known to be difficult, due to the lack of a single definition of what a mountain is, which sparkled different approaches to mountain analysis [5, 6].

Mountain peaks identification from DEM data is a specific case of landform detection: given the DEM representation of an area of the Earth surface, the goal is to identify the coordinates of the points that belong to a mountain landform. A further restriction is *summit identification*, which aims at determining the coordinates of a single point representing the summit of the mountain.

Computer-aided mountain summit identification can help improve the accuracy and completeness of Voluntary Geographical Information Systems (VGIS), which depend on the contribution of volunteers for the quality and

quantity of their data. For example, at the moment the popular Open Street Map (OSM) system contains $\approx 506,097$ mountain peaks, of which 36,25% miss the altitude value. Machine intelligence could be exploited to improve the situation: rather than waiting for the spontaneous contribution of volunteers, one could push automatically extracted candidate mountain summits, with their coordinates and altitude, to a crowds of volunteers, soliciting them to verify such candidate entities and add the valid ones to the VGIS.

A common characteristic of state-of-the-art mountain identification algorithms is that they rely on manually selected features (e.g., altitude, slope, curvature, local relief, elevation, prominence, isolation, etc), apply heuristic rules, and require parameters to be manually configured by the user. The selection of the features to use and of the parameters values to obtain the most accurate identification is a non trivial task, especially when multiple parameters are involved.

In a previous work [7] the supervisors of this thesis have explored the use of data-driven learning-based methods for mountain summit identification and investigated the use of Deep Learning (DL) and Convolutional Neural Networks (CNN) [8] [9] as an alternative to heuristic algorithms that require the manual selection of landform features and parameters. The idea is to let a deep neural network learn the optimal features and parameters for recognizing mountain summits directly from the data encoded as bi-dimensional images. Under this formulation, mountain summit identification becomes a pixel-level binary classification task, whereby for each DEM cell a prediction is made whether it contains a summit or not. This requires training the network on a suitable gold standard, built from existing peak collections, so that the DL classifier learns the significant knowledge on the positioning of mountain summits embodied in the available maps and applies such knowledge to infer the localization of peaks nnot present in the cartography. At training time, the DL model is fed with positive and negative examples, extracted from the DEM data and from the coordinates of existing summits. At inference time, the input to the trained DL model is a set of images extracted from DEM data, different from those used for training, and the output is a map that assigns to each pixel/DEM location the probability of representing

a mountain summit.

In this thesis, we investigate a different learning-based approach to mountain summit identification from DEM data. Instead of treating the DEM data as a pseudo-image and applying CNN models from the image analysis domain, we extract the *surface network* of the Earth from the DEM data and address mountain summit identification as a graph node classification task. Surface networks are graph based topological data structures, which allow a representation of the geometry and topology of the Earth surface well suited to such applications as generalisation, drainage network analysis and route planning [10].

Graph Deep Learning (GDL) is a branch of Machine Learning that extends the architectures and techniques of Deep Learning beyond Euclidean domains (such as those of sequence and grid data), enabling inference on data defined in other domains such as graphs [11]. Graph-indexed data occur in many applications, where some signal is associated to nodes connected by links that express some relationship between nodes. Examples of such structures are common in applications for recommender systems, computer graphics, social network analysis, and life sciences.

The possibility or representing the Earth surface by means of a surface network, which encodes the surface salient points, some of their features, and their topological relations, raises the question of the applicability of GDL to landform identification. In this thesis we investigate such a question, with a specific focus on mountain summit detection, to understand *how GDL architectures perform on surface network data and compare to both heuristic method and learning-based methods that encode DEM data as images.*

For the graph construction we explored the method in [12] , where the DEM data is analyzed locating critical points (nodes): peaks, pits and saddles and their connection though path (edges): ridges (connect saddles with peaks) and channels (connect saddles with pits).

The idea is to let the methods to learn the optimal parameter configuration for recognizing mountain summits, by training it on a suitable gold standard.

The methods selected were Logistic Regression[13] for a baseline and

GraphSAGE[14] a more complex one that performs node classification based on the aggregation of neighbours information based on the graph topology. To generate the gold standard we exploit existing cartography (OSM and Swiss3DNames). Considering that the critical points are enriched with features derived from the surface, it was implemented also Node2Vec [15] as a step for learning representations for the topology which cannot be represented trivially in an euclidean space.

The nodes of the graph are labeled based on such gold standard to train and evaluate the proposed methods. We compare the results with other existing methods from the literature, analyzing their performance and the possible points for improvement in future work.

The contribution of this thesis can be described as follows:

- We formulate the mountain summit identification task as a node classification problem, in which traditional and graph ML models are trained by supplying to it the DEM data encoded as a graph based on surface networks.

- We experiment such methods in a mountainous region in Switzerland, using SRTM data at three degree resolution as input and peak coordinates from the OpenStreetMap (OSM) and SwissNames3D public data sets as gold standard.

- We evaluate the performance of the model, based on a distance-based heuristic and compare with other methods and we discuss the possible directions of improvement of the proposed approach.

The thesis is organized as follows. In chapter 2 we present a review of the state of the art of the heuristic methods and the Deep Learning approaches to extract landforms and, in particular, peaks summit from SRTM DEM data and evaluate the obtained results. We also introduce the reader to the topic of Graph Deep Learning, a specific family of models applying on networks.

In chapter 3 we introduce the concept of surface network and the related studies for building networks from DEMs. Then The Main Machine Learning models involved in this work are then discussed.

In chapter 4 we explain the techniques adopted for building the graph followed by our workflow for data pre-processing, peak extraction and post processing with the proposed Deep Learning approaches and the replicated heuristics methods.

In chapter 5 we illustrate the choice of parameters for each examined method and perform a quantitative and qualitative analysis on the results.

Finally, in chapter 6 we summarize our work and discuss future improvements.

Chapter 2

Related Work

In the following sections we will describe methods and techniques related to the main pourpose of this work: automatically extract peaks using DEM data. In Section 2.1 we will see more classical and heuristic methods of computer science which directly interact with DEMs for the identification of peaks. Section 2.2, instead, is an introduction to *Deep Learning*, the building block for the techniques used in the development of our workflow for learning how to extract peaks from graphs built over DEMs.

2.1 Mountain peaks extraction from DEM

One of the earliest attempts of extracting surface specific points, like peaks, from discrete terrain elevation data was done by K.Peucker and H.Douglas in [16]. They described several methods designed to detect landforms like pits, peaks, passes, ridges, ravines, and breaks, given an array of sampled, quantized terrain elevations. Their work analyzed topographic features of grid cells according to the patterns of elevation changes between neighbour cells. The results are limited, as stated in [17], because of the encountered problems with single cell pits in flat areas due to high signal-to-noise ratio. Under the bigger landform detection problem goes the subproblem of mountains peaks identification. The pioneer work [3] introduces an automated method for classifying generic terrain features extracted from DEMs. Their two-class

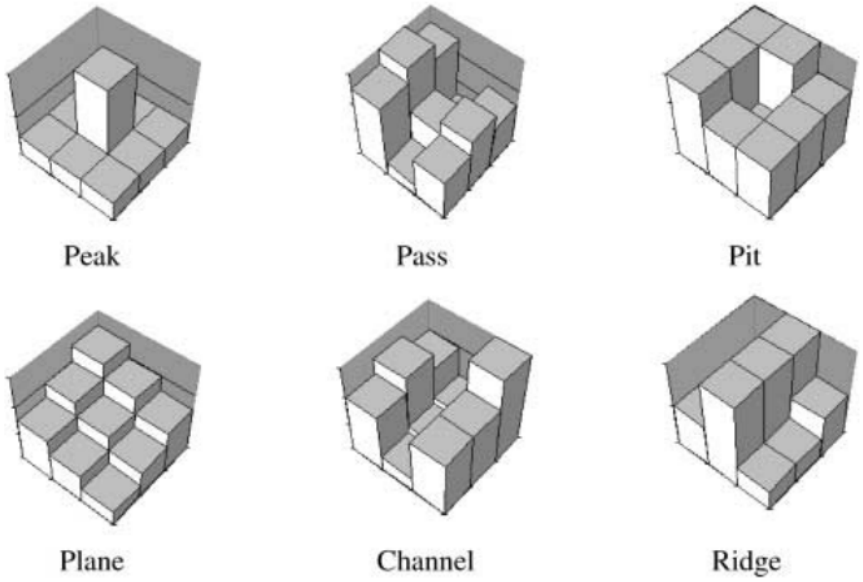

Figure 2.1: Morphometric classes
Figure illustrating the six morphometric classes that can be extracted from a raster DEM. Image courtesy Peter Fisher, Jo Wood and T. Cheng [6].

system differentiates mounts, the elevated features, and non-mounts, the remaining terrain features. Despite some limitations of the algorithm, also this work has been affected by the quality of digital data. Indeed, the authors of [17] show how DEM errors affect the computation of the derived attributes. There exist six morphometric classes that can be identified in a DEM by analyzing the eight direct neighbours elevations, and an example can be seen in Figure 2.1. Many studies and researches have been done to further extend the basic eight-neighbours method for extracting the morphometric classes. With a new perspective in [5] the authors show that based on the scale with which we analyze the terrains we can classify them differently. This goes in the direction of fuzzy set theory of terrain analysis. Peter Fisher, Jo Wood and T. Cheng [6] explored the fuzziness of multi-scale landscape morphometry where they stated that any location can be allocated to a specific class, but the class to which a location is assigned may vary considering different scales. Indeed, an area that is a channel by considering its eight direct neighbours can be part of a ridge for a larger scale, considering, for example, not

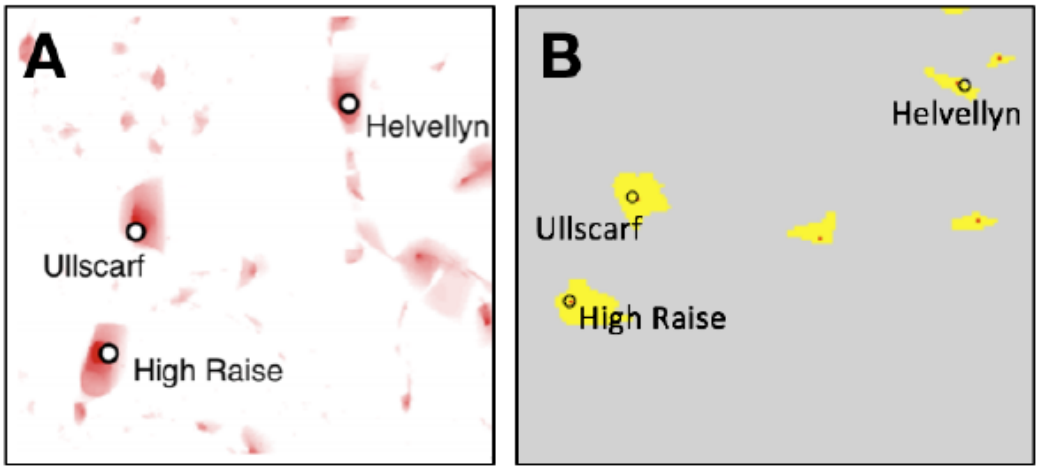

Figure 2.2: Landserf application
(A) Landserf fuzzy feature extraction for peak classification and
(B) Landserf peak classification. Examples in a small area of Lake District using OS Terrain 50 DEM

only the adjacent cells but also the ones that are connected to their neighbours. They showed how is it possible to combine classification at different scales for finding peaks. The method is implemented in the Landserf[1] application [18]. An example where we can see the fuzzy concept of "peakness", i. e. how much a given location belongs to the class of peak, can be seen in figure 2.2 (A) where red denotes a higher value of peakness. These techniques are further explored with a qualitative work in [19] and two use cases are reported: the Ben Nevis area, containing 19 peaks, and the Ainsdale coastal sand dunes. They show that some areas that have large value of peakness are actually corresponding to real peaks present in the dataset of known peaks used by authors, while some others are not associated with any summit. The algorithm outcome is strongly influenced by its tunable parameters. The Landserf tool implements also a heuristic technique, based on the more classical method of the eight direct neighbours, which considers that a location may be considered a peak if its altitude is higher than a given treshold and there is a minimum elevation difference w.r.t its adjacent cells [18]. These

[1] http://www.landserf.org/

quantities are two tunable parameters. An example can be seen in figure 2.2 (B). The yellow areas indicate the locations which are part of the extent of a mountain, while the red color denotes the summit of a mountains. With yet another approach, presented in [20], and implemented in Landserf, the tool allows to extract peaks from DEMs also by building a so called *Metric Surface Network*. It consists, essentially, in a graph with weighted edges whose vertexes are the critical points (peaks, pits and saddles) of a surface while the edges are the critical lines (channels and ridges). Also this method is subject to parameters tuning for optimization. Still considering 3x3 windows of DEM cells for the analysis of the locations for possible summits, the author of [21] and [22] combines topographic and morphologic criteria. According to these works a point, to be considered a peak, must reside in a non-flat area, must be the highest within its eight neighbors and must have at least a certain horizontal and vertical distance from other candidate peaks. The author, with a further qualitative study in [23] analyzes in detail the shape of a peak. By evaluating DEM data of the Kamnik Alps in Slovenia they show that shapes are dependent on each other and are not universal. The shape examination improves peak detection even though it is considered still as a very complex task to be generalized and solved by only authomated methods.

Apart of studying the elevation of a cell relative to its adjacent ones, there have been developed also methods that considers the shape of a peak relative to its neighbours. Similarly to [6], in [24] the authors consider mountains peaks as fuzzy entities and define a multi-scale peaks extraction algorithm based on local properties such as topographic position, number of summits in the neighborhood, relief, relative altitude and mean slope. The algorithm returns the peak class membership of a point expressed by a value suitable to further analysis through the application of a treshold. The effect of varying the treshold and the scale is presented through a qualitative evaluation.

Other studies, like [25] of J. Jasiewicz and F. Stepinski, focused in applying pattern recognition approaches for classifying and mapping landforms. They identified the so called *geomorphon*, a simple ternary pattern that serves as base archetype for building more complex morphometric landforms. There

are 498 geomorphons that constitute a comprehensive and exhaustive set of all possible morphological terrain types including all the standard elements of landscape, as well as unfamiliar forms. This approach of classification is significantly different from classical methodologies, indeed, it uses tools of computer vision rather than tools of differential geometry. The geomorphons can be then mapped to the more classical morphometrical classes.

SAGA GIS [2] constitue another important tool in the field of landform detection. It has been usen in [26] where the authors propose a worklow for Digital Terrain Analysis (DTA) and landform reconition and extraction from DEM. They analyze the most used terrain attributes, like slope, curvature and elevation, and combine different landfrom recognition methods, like digital topography. hidrology and morphology. The results show that different landforms are better characterized by different resolutions. In particular, higher resolutions allow to distinguish between more classes in the context of Fuzzy Landform Classification.

An important work based on heuristic approaches introduced in [27] determines the *prominence* and *isolation* for the mountains, two important features characterizing peaks. Prominence is a measure of the independence of a summit and it is computed by finding the minimum vertical distance needed to descend from the peak to to ascend to a higher one. Isolation, instead, measures the minimum distance of a summit from another one with higher elevation. For each peak in the world these two values are calculated and the results compared with the PeakBagger dataset [3]. When computing the prominence for the summits a so called divide tree is built which represent the connection with the higher ones (except of the root of the tree which is the highest). These connections follow the path of descent from the peak until the lowest points, which is a saddle, before restarting to climb up a higher one. This tree can be thought like a sampling of the critical points and critical lines from the (metric) surface networks.

[2] http://www.saga-gis.org/en/index.html

[3] http://www.peakbagger.com/

2.2 Deep Learning

Artificial Intelligence (AI) is the field that studies the creation of computer systems able to mimic the human cognitive functions in order to solve non trivial problems. Machine Learning (ML) is a subfield of Artificial Intelligence that develops solutions that do not rely on explicitly programmed instructions to perform a certain task, but exploit a data-driven approach, in which patterns are learned from training data. Learning can be supervised, semi-supervised or unsupervised [28]. Furthermore, Deep Learning (DL) is a class of Machine Learning algorithms that relies on deep neural networks to learn data representations, which recently experienced great success, thanks to the vast amount of training data and to the increasing computational power available nowadays. DL models, have proved capable of achieving high quality results in a wide range of Computer Vision tasks, such as image classification, detection, localization and segmentation. In particular, it is easy to feed the above mentioned techniques, with euclidean data such as feature vectors, or images. The performance of Machine Learning algorithms heavily depend on the data they are fed with; the choice of the representation for the data on which they are applied requires important efforts on the design of preprocessing pipelines and data transformation. In cases were we have too much data engineering the relevant features or when we're in the domain of non-Euclidean data, the task of preparing the input to feed ML or DL models could be more challenging. To cope with this, there is a field called Rresentation Learning which goes in the direction of learning representations of the data that make it easier to extract useful information when building classifiers or other predictors. Deep Learning techniques are formed by the composition of multiple non-linear transformations with the goal of yielding more useful representations, as stated in [28]. The organization of the AI fields can be seen in Figure 2.3. In this section, we will do an overview of main concepts on these areas.

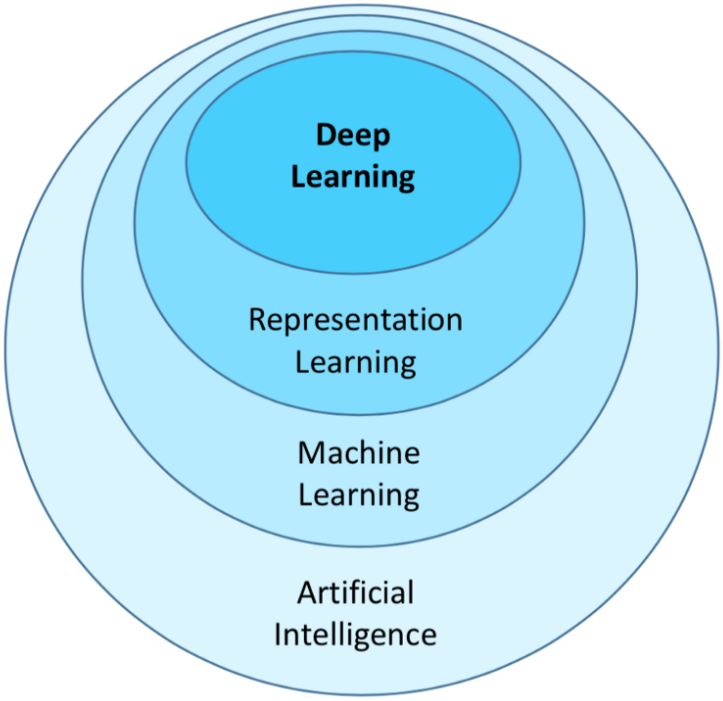

Figure 2.3: Artificial Intelligence fields
Figure showing how Artificial Intelligence can be hierarchically organized in subfields. Deep Learning can be seen as a specific field of Representation Learning, that is by itself a specific field of Machine Learning, a subset of the broader class of Artificial Intelligence methods.

2.2.1 Artificial Neural Networks

The basic model over which are built many and more complex Deep Learning models are the so called Artificial Neural Networks, which are vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm but rather a framework for many different machine learning and, consequently, deep learning algorithms to work together and process complex data inputs. Their ability of learning from examples without being programmed for a specific task made them a breakthrough in many fields. Learning can be seen as as the process of adjusting internal parts of the model in order to approximate some unknown function f^*, just by feeding the network with different examples of data. For example, for a classifier, f^* may be a function that maps an input $\mathbf{x}$ into a category y. An artificial neural network defines a mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ where the values of the parameters $\boldsymbol{\theta}$ are learned such that f^* would result as the best approximation function.

Artificial Neural Networks are based on a collection of connected units or nodes called artificial neurons whose connections, that have a shallow similarity with the biological synapses, can transmit a signal from one artificial neuron to another. Once a neuron receives a signal (usually represented by a real number) can process it and then send the outcome to the other neurons to which is connected. Generally, the output of each artificial neuron is computed by some non-linear function of the sum of its inputs, while the connections between artificial neurons are called edges. To the edges between neurons are then associated some weights that represent the strength of the connection and are the parts of the model that can be adjusted during the learning process. As we can see in Figure 2.4, each neuron i is fed with a vector of real numbers $\mathbf{x}$, i.e., the outcomes of the processing of the other neurons connected to i; in this case we refer to x_j as the output of node j feeding node i. Each input will be then multiplied by the weights w_{ij} associated to the edges that connect the neurons j to neuron i and summed with the results of the other multiplications. There is, essentially, a dot product between the vectors $\mathbf{x}$ and $\mathbf{w}$ representing, correspondingly, the input data

and the weights of the edges connecting the neurons. Then, the summation of the multiplications is used as an input to a non-linear activation function g which will produce the final signal y_i.

As an example of non-linear activation function we may consider the binary step (which is also present in Figure 2.4) and write the output signal y_i as:

$$y_i = g(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \cdot \mathbf{w} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

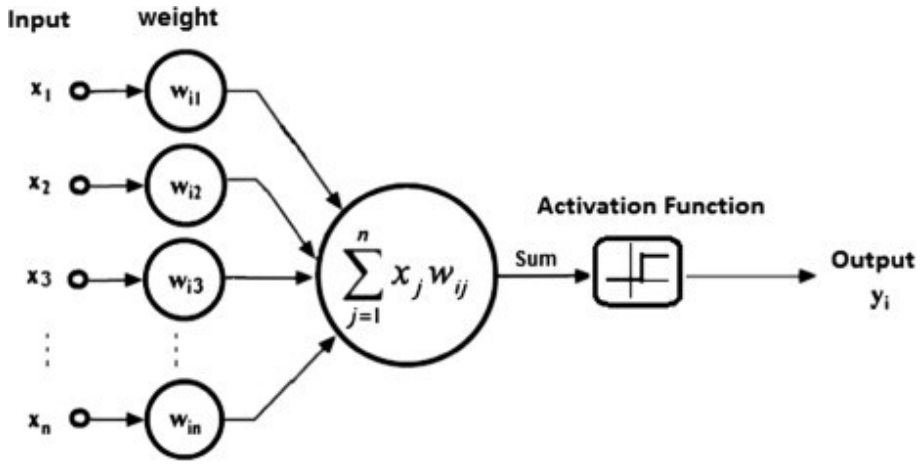

Figure 2.4: Artificial Neuron

Figure taken from [29] depicting the computational structure and the flow of information between neurons in an Artificial Neural Network.

One of the most well known example of ANN are the Feedforward Neural Networks, also called Multilayer Perceptrons (MLPs). As stated in [30], these models are called feedforward because information flows from $\mathbf{x}$, which generally represents the data, through the intermediate computations used to define f, and finally to the output $\mathbf{y}$. MLPs are acyclic, i.e they do not include feedback connections.

Feedforward Neural Networks are called networks because they are typically represented by composing together many different functions. An example may be $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$ where there are three functions ($f^{(1)}$,

$f^{(2)}$ and $f^{(3)}$ connected in a chain. Usually we say that the functions constitute the layers and we refer to $f^{(1)}$ as the first layer, $f^{(2)}$ as the second layer and so on.

In Figure 2.5 we can see a general architecture of a (Deep) FNN. The difference between FNN and the deep ones consist in the number of layers. Deep FNN are in general composed by many more layers than the normal ones. It is important to highlight that the functions that constitute the chain can be cosidered as computational layers composed by different numbers of neurons. Essentially the stacked layers of neurons can be associated to the chain of funcions, and each layer's functionality is given by the combination of the processing units, i.e. the neurons, that constitute the layer. The layers can be then organized as: *input layers* (L_1), where the network is fed with the examples (usually constituted by vectors of real numbers), *hidden layers* (L_2, L_3, L_4), which usually receive the outcome of the computations of the input layers, and the *output layers* (L_5) that are fed with the signals coming from the hidden layers and whose output constitute the model response to the data. For example, if the model is trying to classify images, each different y_i representing the output may indicate the probability for the input picture to belong to a given class (cat, person, building, car, etc...). Organizing the neurons in layers allows to have powerful models that are able to learn really complex functions, or at least quite close approximations to an ideal f^*. Also notice that this particular kind of Deep Feedforward Neural Network is a Fully-Connected one. As we can still see from Figure 2.5, except of the input and output layers, each neuron from each layer is connected to all the neurons from the previous layer and to all the neurons of the next layer. The exception of the input and output layer is given by the fact that, as we said, the input layer receives the example so its neurons have no incoming edges from other neurons, while the output layer receives only the signals from the computations from the previous layer but is not feeding other layers since its result constitute the final approximation of the function f^*. The overall length of the chain gives the depth of the model, the term from which arose "deep learning" name.

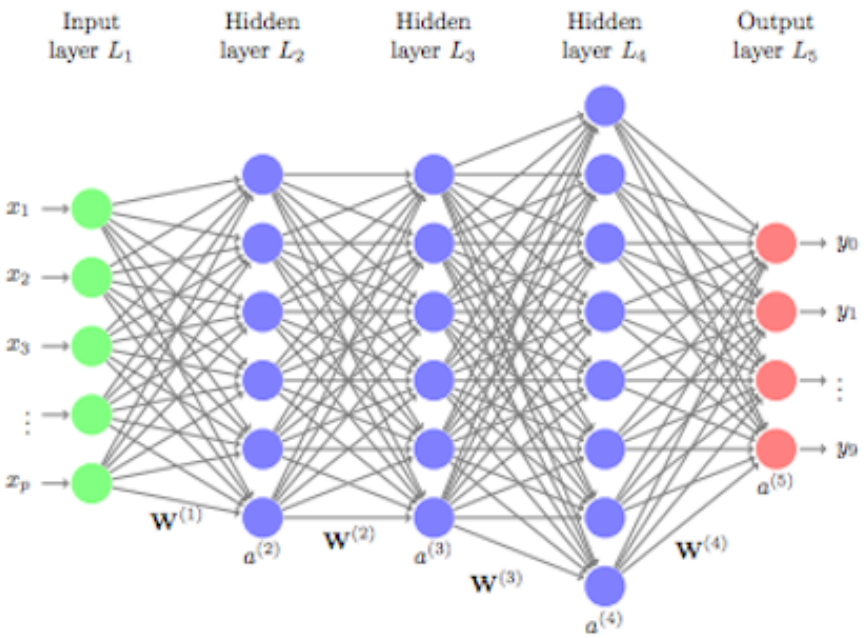

Figure 2.5: A Deep Feedforward Neural Network

Figure taken from [31] showing the computational structure of a Deep (Fully-Connected) Feedforward Neural Network and the Layers that can be distinguished between input layers L_1, hidden layers L_2, L_3,L_4 and output layers L_5.

2.2.2 The learning process

As we said, the parts of the neural networks models which are usually modified to better approximate the objective function f^* are the edges connecting the neurons and their associated weights. Again as an example consider a supervised learning where human labeled data is used for training an artificial neural network that is classifying the content of images. The learning process is mainly constituted by three phases, the *forward propagation*, the *loss computation* and the *backward propagation* (often named just *backpropagation*). The first one occurs by feeding the network with examples, passing them across all the network and applying the transformations of each neuron. The outcome of the output layers can be interpreted as the network's prediction for the content of the image. After the output is calculated for each example that is given to the model, an error, usually called loss, is computed between what is the real content of the image and what is the neural network predicting. Here the last phase, the backpropagation, takes part. The loss is then propagated through all the neurons of the hidden layers and it is used to adjust the weights of the edges to reduce to the minimum the error. This process can be visualised in Figure 2.6. The objective is to make the loss as close as possible to zero the next time we will use the network for a prediction. For doing so, the *gradient descent* technique it is used; it changes the weights of the edges with small increments by calculating the derivative (or gradient) of the loss function. Gradient descent tunes the weights in order to descend towards a *global minimum* of the loss function of what is predicted against which are the real values of the examples. If we think at the neural network as a composition of functions, gradient descent tries to find the best $\boldsymbol{\theta}^*$ for the function $f(\mathbf{x}; \boldsymbol{\theta})$ representing the network in order to minimize the loss between the ideal f^* and the actual approximating function $f(\mathbf{x}; \boldsymbol{\theta})$.

2.2.3 Deep Learning on GIS

The latest improvements of Artificial Intelligence and Deep Learning made them suitable for replacing specific task algorithms. Also in the field of Computer Vision, for tasks such as image classification, detection, localization and

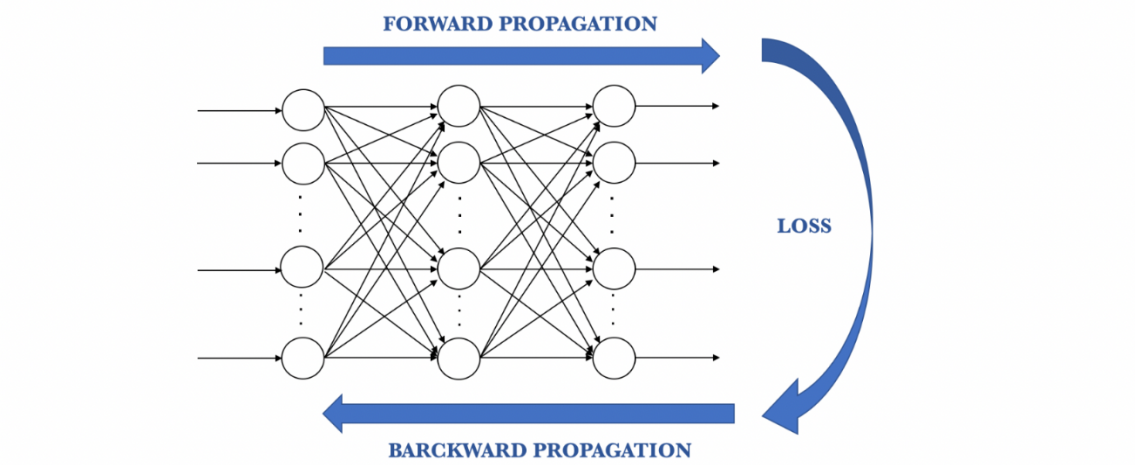

Figure 2.6: Learning phases of a Neural Network

Figure representing the learning phases of an Artificial Neural Network. (1) Forward propagation: the training data is passed across all the neural network and the outcome of the output layers represents the network's prediction. (2) Loss: represents how far the prediction is from the objective f^*. (3) Backpropagation: the propagation of the loss to all the neurons in the hidden layers that contribute to the output. Image courtesy Jordi Torres [32]

segmentations [33], the advancements of Deep Learning allowed a remarkable improvement of the performances compared to traditional methods. Specific models of Deep Learning, the Convolutional Neural Networks (CNN), have proved a great ability in dealing with images. As stated in [34] they have been used in several works of geoscience and remote sensing as an important tool for the analysis of aerial images. Specifically, in [35] and [36], CNNs have been applied for aerial images segmentation, tackling land cover and objects mappings, in which each pixel is assigned a given class (e.g. road, car, vegetation, building, etc). Artificial Intelligence has been proved also of being effective for the analysis of DEM data. Models such as the Multilayer Perceptron have been applied in [37] for classifying the above-ground objects by having as main target the separation of the high-standing structures (trees and building) from their surrounding terrain. In [38] the authors used DL on Airbone laser scanning (ALS) point cloud data for extracting digital terrain models (DTMs). Their method allows to classify points by using an image-like classification approach. Indeed, they map the relative height difference of each point with respect to its neighbours (in a square window) to an image.

Digital Elevation Models in some areas of the Earth lack of good resolution. To overcome this problem in [38] the authors proposed the so called DEM super resolution, a technique that improves the resolution for a DEM on basis of some learning examples. With a different approach in [39] and [40] there have been suggested techniques which involve the synthetic generation of terrain images by using a specific model of DL, the Deep Generative Adversarial Neural Networks (GANs).

Recently, the authors in [41] proposed the usage of CNN for extracting peaks from DEMs by creating patches of dimension 31x31x3 representing parts of the physical region delimited by the DEM. Each patch is a square of 31x31 pixels, where every pixel is a cell of the DEM raster containing the elevations. The patches containing the elevations are then treated like images, which makes them really suitable for models like CNN. The authors showed how is it possible to use Deep Learning methods to train a model with terrain data represented as DEMs capable of identifying mountain summits.

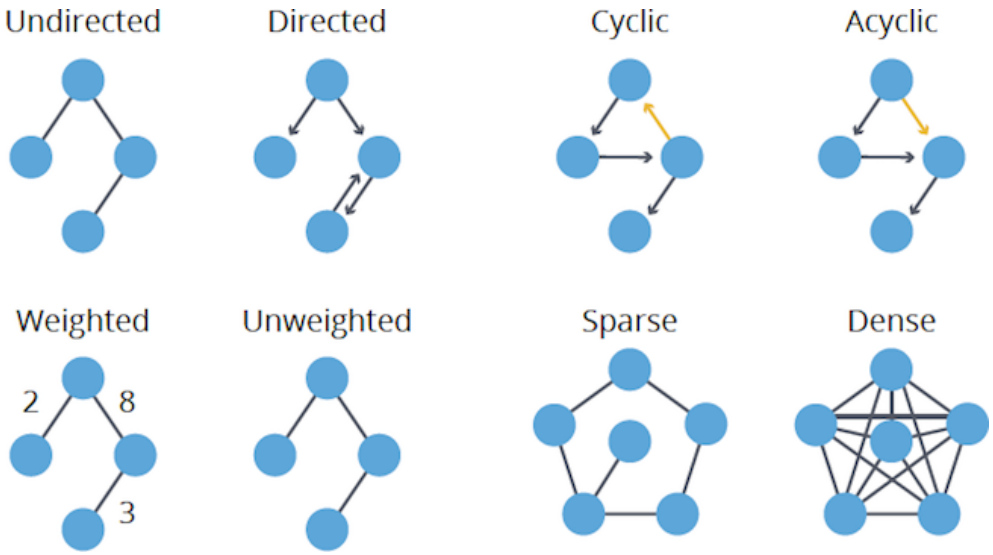

Figure 2.7: Diagrammatic representation of the types of graphs
The blue circles represent the vertexes (or the nodes), while the gray lines represent the connections among the nodes, i. e. the edges. The figure shows how can be defined 8 categories of graphs based on the type of edges.

Based on their work, we are extending the application DL for extracting peaks from digital elevation models by building graphs, specifically surface networks, over the DEMs and then applying Deep Learning on the graphs.

2.2.4 Deep Learning on Graphs

In mathematics, and more specifically in graph theory, a graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called an arc or line) [42]. Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges. An example of diagrams can be seen in Figure 2.7.

Graphs naturally exist in a wide diversity of real world scenarios, e.g., social graph in social media networks, citation graph in research areas, user interest graph in electronic commerce area, knowledge graph, etc. In Figure

2.7 we can see a categorization of graphs based on their type of edges. For an *undirected* graph, an unordered pair of nodes that specify a line joining these two nodes are said to form an edge. For a *directed* graph, the edge is an ordered pair of nodes. If an edge is representing a relation in a family, for example a member *father* of another member, the nodes represent the members while the directed edge represents in which direction is going the relation. In the context of directed graphs there can be made a distinction between *cyclic* and *acyclic* graphs, i.e if the graphs do contain a cycle or not. A cycle of a graph G is a subset of the edge set of G that forms a path such that the first node of the path corresponds to the last. A *weighted* graph is a graph in which a number (the weight) is assigned to each edge. Such weights might represent for example costs, lengths or capacities, depending on the problem at hand. The *unweighted* graphs, instead, do not have a cost associated or they all have the same cost usually set to 1. Also it can be distinguished between sparse or dense graphs. In the last ones there is an edge between all the possible pairs of vertexes in the graph, while in the sparse this is not happening.

There exist also graphs with features associated to the nodes, i.e. some numerical or categorical attribute representing properties in the domain of the graph. Figure 2.8 shows a diagram representing such kind of graphs. For example if the graphs are representing sentences and the nodes are words, instances of attributes may be the length of the word, its position in the sentence and the word class (noun, verb, adjective, etc).

Being able to analyze properly these kind of graphs means being able to make good use of the information hidden in the structure of the graphs. An increased attention has been devoted to the topic in the last few decades [43]. In particular, the utilization of Machine Learning on graphs became an important and ubiquitus task with applications ranging on very heterogeneous fields [44]. Examples of ambits of implementation go from classifying the role of a protein in a biological interaction graph, to predicting the role of a person in a collaboration network, from recommending new friends to a user in a social network to predicting new therapeutic applications of existing drug molecules whose structure can be represented as a graph. For

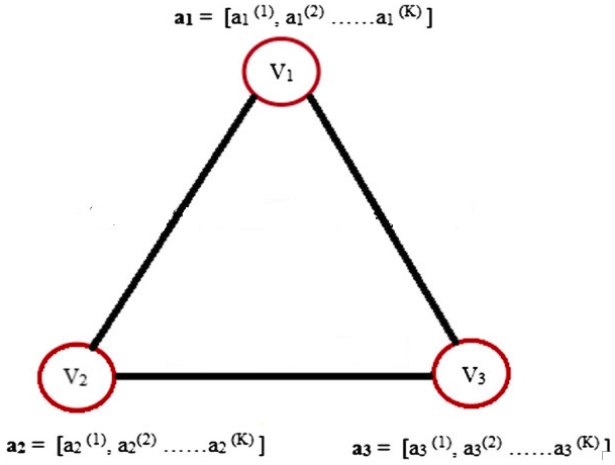

Figure 2.8: Graph with attributes

The depicted graph is containing three vertexes and three edges. For each of the the nodes it is also present a vector a_i containing the properties of the node. In this particuar case each vector is composed by a fixed number K of attributes, arranged in an ordered list, from $a_i^{(1)}$ to $a_i^{(K)}$.

achieving these tasks some typical problems need to be addressed on graphs. Usually node classification, link prediction and community detection are the main concerns, but also some other specific problem as a combination of the basic ones, such as subgraph classification or entire graph classification, can be evaluated. *Node classification* aims to correctly classify the nodes as belonging or not to a given category. For example in a molecule the nodes may represent the atoms which we want to categorize as belonging to a chemical element. A diagrammatic representation of a molecule and its corresponding graph can be seen in Figure 2.9 (b). *Link prediction* problem, instead, goes in the direction of being able to predict the existence of a connection between two nodes and, more specifically, in the case of weighted graphs, the strength of the connection which can be represented by a real number. In Figure 2.9 (a) we can see an example where having a community of members represented with nodes we may want to predict a future connection, i.e. link prediction, among members which are not already in contact. *Community detection*, instead, aims to cluster sets of nodes sharing specific properties that are intended to belong to a certain group. Figure 2.9 (a) can be an

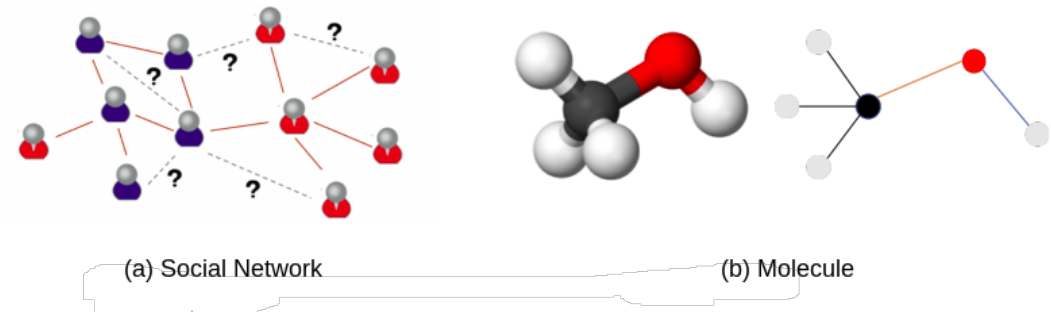

Figure 2.9: Graph models of reality
Abstract representation two real world situations suitable for graphs. (a) Social Network instance with the members depicted as nodes and their relationships as edges. (b) Molecule representation where the nodes are the atoms and the edges are the molecular bonding. Image taken from [45]

example where the two groups are formed by the members depicted in blue and red respectively. Other derived subproblems of the above described ones can be, for example, *subgraph classification* where the purpose is to classify entire portions of the original graph, comprehensive both of the nodes and the edges. *Graph classification*, instead, aims at classifying the entire graph with a category from a given domain. Still in the context of molecules, the purpose may be to classify the graph as "drug" or "non drug". Many machine learning applications seek to make predictions or discover new patterns using graph-structured data as feature information.

ML can automate functions, such as image classification, that are easy for a human to do, but hard to turn into a discrete algorithm for a computer. Deep learning allows us to transform large pools of example data into effective functions to automate that specific task. This is doubly true with graphs; they can differ in exponentially more ways than an image or vector thanks to the open-ended relationship structure. The central problem in machine learning on graphs is finding a way to incorporate information about the structure of the graph into the machine learning model. Graphs usually lack of a common structure either between two different graphs refering to the same domain, either among the nodes and the edges of the same graph. For

example in a situation where two graphs represent two different communities, with the nodes representing the individuals and the edges representing the relations among the nodes, it would be really improbable to have the same number of nodes and edges. Also among the same graph the vertexes are generally having different numbers of edges connecting to their neighbours. Traditional methods that use machine learning algorithms with graphs rely on handcrafted features for encoding the graph structural information. Also, using directly the graphs as input for machine learning algorithms has a high computational and space cost [43]. Recent approaches overcome the problem by learning *graphs embeddings*, i.e. converting graphs into low dimensional space in which the graph information is preserved. This allows to use the embeddings of the graps as input to downstream machine learning models. Methods that create embeddings from graphs are part of the field of representation learning [44]. There exist, however, also deep learning models which are able to handle directly graphs as inputs for performing tasks such as classification and regression of the entire graph, parts of it or its nodes. Standard machine learning algorithms generally rely on grid data structures (in 1, 2 and 3 dimensions). Convolutional Neural Networks (CNNs), one of the most successful examples of deep learning algorithms, exploit grid data structures and the translational equivalence/invariance with respect to this grid [46]. One of the key challenges of extending CNNs to graphs is the lack of vector-space structure and shift-invariance making the classical notion of convolution elusive [47]. In their work, the authors of [46] showed how is it possible to extend these properties for graphs. Then, even though the distincion is not sharp and the two categories may overlap, deep learning on graphs can be distinguished mostly in two groups of methods: representation learning through graph embedding and graph deep learning with direct application of the models over the graphs.

Graph Embedding

As stated in [43], the problem of graph embedding is related to two traditional research problems: graph analytics and representation learning. Particularly,

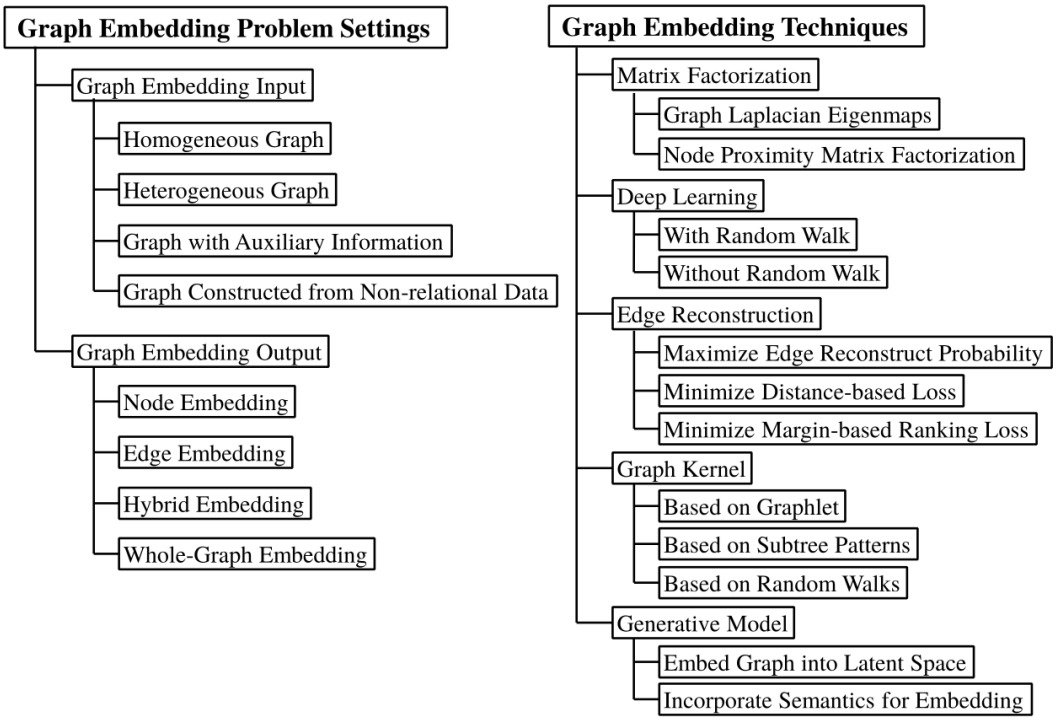

Figure 2.10: Graph Embedding
Figure taken from [43] representing the taxonomy of graph embedding.

graph embedding aims to *represent* a graph as *low dimensional* vectors while the graph structures are preserved. Previous work addressed to this problem as a pre-processing step using hand-engineered statistics, such as node degree, to extract structural information. In contrast, representation learning approaches treat this problem as a machine learning task itself, using a data-driven approach to learn embeddings that encode graph structure [44]. In their survey the authors of [43] show a graph embedding taxonomy based on the problem setting and on the used technique. This distinction is showed in Figure 2.10.

Most of the embedding methods work in an *unsupervised* manner, i.e. the algorithms have no prior knowledge about how the embeddings should be done or for which downstream machine learning task the embeddings will be used. There exist however also some graph embedding approaches which can be categorized as *supervised* which make use of regression numerical

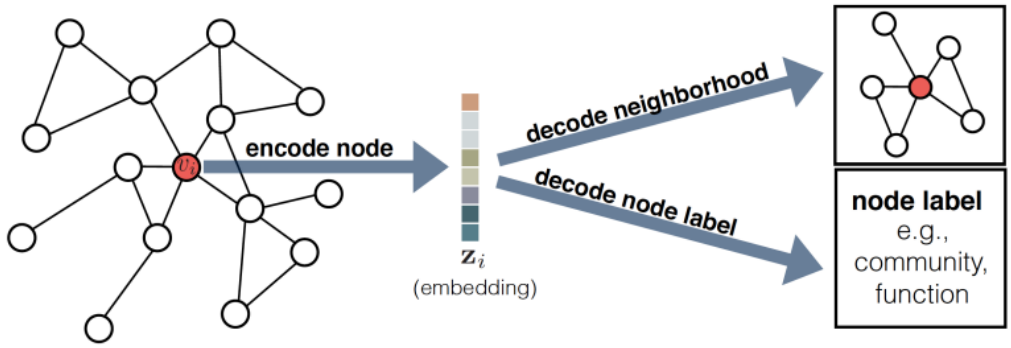

Figure 2.11: Encoder-decoder approach

Figure taken from [44] representing an overview of the encoder-decoder approach. First, the encoder maps the node, v_i, to a low-dimensional vector embedding, $\mathbf{z}_i$, based on the node's attributes and/or local neighborhood structure. Then, the decoder extracts user-specified information from the low-dimensional embedding, which can be the local neighborhood of v_i or a classification label associated to v_i. By jointly optimizing the encoder and decoder the system learns to compress information about graph structure.

attributes or classification labels in order to optimize the embeddings.

In the context of unsupervised node embedding a really common *framework* is the so called *encode-decode* one where the *encoder* maps each node to a low-dimensional vector (the embedding), while the *decoder* decodes structural information about the graph from the learned embeddings (Figure 2.11). The vast majority of the works use a *pairwise decoder* which assigns a graph proximity measure (expressed by real a value) to pairs of embeddings, i.e quantifies the proximity of the two nodes in the original graph. Applying such decoder to a pair of embeddings ($\mathbf{z}_i$,$\mathbf{z}_j$) returns a *reconstruction* of the proximity between v_i and v_j in the original graph. Finally, a loss function ℓ measures how far the decoded proximity value is from the real proximity value. According to the measure loss ℓ the encoder is then tuned in order to produce embeddings that would minimize the distance between the decoded proximities and the real ones.

Under the encoder-decoder framework we can find numerous methods

for embedding graphs; their differences vary based on their encoding function, decoding funcion, proximity measure and loss function. For example, DeepWalk [48] and node2vec [15], two really adopted algorithms in graph embedding literature, are using deep learning with random walks statistics. In graph theory, given a graph and a starting point, we select a neighbor of it at random, and move to this neighbor; then we select a neighbor of this point at random, and move to it. The (random) sequence of points selected this way is a random walk on the graph. Their key innovation is optimizing the node embeddings so that nodes have similar embedding if they tend to co-occur on short random walks over the graph. Instead of using a deterministic measure of graph proximity, the methods based on random walks use a flexible and stochastic measure of graph proximity, which has led to superior performance in a number of settings [49]. Both methods, however, are failing to leverage node attributes during encoding which can be a hard limitation considering that node attributes can be higly informative with respect to the node's position and role in the graph. Also, these methods are inherently *transductive* like said in [**transductive**], i.e. they can only generate embeddings for nodes that were present during the training phase, and they cannot generate embeddings for previously unseen nodes unless additional rounds of optimization are performed to optimize the embeddings. This is highly problematic for domains that require generalizing to new graphs after training. Other methods, like Deep Neural Graph Representations (DNGR) [50] and Structural Deep Network Embeddings (SDNE) [51] implement encoders that do not use only the graph structure in order to compress the information about a node's local neighbor but they incorporate also the information about the node. These two methods also differ from the previous ones because they use a *unary decoder* instead of a pairwise one. However, also these approaches are not using attribute information about the nodes and they are strictly transductive and cannot generalize across graps. They are also really costly because the dimension of the autoencoder is fixed and equal to the number of vertexes inside the graph which can be really a huge problem for graphs with millions of nodes. Similarly, some recent node embedding approaches designed encoders that rely on a node's local neighborhood, but not

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\{\mathbf{W}^k, \forall k \in [1, K]\}$; non-linearity σ; differentiable aggregator functions $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 2.12: Neighborhood-aggregation encoder algorithm
Figure taken from [44] representing the pseudocode of the neighborhood-aggregation encoder algorithm.

necessarily the entire graph. The idea is to generate embeddings for a node by *aggregating* information from its local *neighborhood*. The aggregation in this context relies on nodes features or attributes to generate embeddings. For example, a social network might have text data (e.g., profile information) or the nodes of a molecule, i.e. the atoms, can have features regarding their chemical properties. The neighborhood aggregation methods leverage this attribute information to inform their embeddings. In cases where attribute data is not given, these methods can use simple graph statistics as attributes such as node degrees. These methods are often called *convolutional* because they represent a node as a function of its surrounding neighborhood, in a manner similar to the receptive field of a center-surround convolutional kernel in computer vision [52]. During the encoding phase the neighborhood aggregation methods build up the representation in an iterative/recursive fashion. As showed in [44] the procedure can be represented with an algorithm whose pseoudocode can be seen in Figure 2.12. The initial node embeddings are set to be equal to the nodes attributes that are usually represented as vectors. At each iteration of the encoder algorithm, nodes aggregate the embeddings of their neighbors, using an aggregation function that operates

over sets of vectors. Then, for each node, the aggregated neighborhood vector is combined with the node's previous embedding of the last iteration generating a new embedding which will be assigned to the node. Finally, this combined embedding is fed through a neural network layer and the process repeats. As the process iterates, the node embeddings contain information aggregated from further and further reaches of the graph. The encoder is forced to compress all the neighborhood information into a low dimensional vector such that, as the process iterates, the dimensionality of the embeddings remain constrained. After K iterations the process terminates and the final embedding vectors are output as the node representations. Different recent approaches fall into the setting illustrated in the algorithm of figure 2.12. Graph Graph Convolutional Networks (GCN) [52], column networks [53] and GraphSAGE [14] all follow the neighborhood aggregation principle but differ primarly in how the *aggregation* (line 4) and vector *combination* (line 5) are performed. These algorithms exploit a set of trainable parameters, i. e. the aggregation functions and the weight matrices of the neural network layer $\mathbf{W}$, which specify how to aggregate information from a node's local neighborhood. Differently from the encoder-decoder approaches listed before, the neighborhood aggregation encoder algorithm share the trainable parameters across the nodes. The parameter sharing increases efficiency (i.e. the parameter dimensions are independent of the size of the graph), provides regularization, and allows this approach to be used to generate embeddings for nodes that were not observed during training [14]. Also, differently from the algorithms of the encoder-decoder framework which are by default unsupervised, the neighborhood aggregation approaches can also incorporate task-specific supervision from node classification tasks in order to learn the embeddings. The task-specific supervision can be seen as a different way of computing the loss between the embedded vector and the desired one represented by the supervision. This allows for more fine tuned embeddings depending on the task we want to achieve.

Based on the convolutional node embedding algorithms it is possible also to define subgraphs embeddings where the goal is to encode a set of nodes and edges into a low-dimensional vector embedding. The basic intuition behind

these approaches is that they equate subgraphs with sets of node embeddings. They use the convolutional neighborhood aggregation idea (i.e., Algorithm in Figure 2.12) to generate embeddings for nodes and then use additional modules to aggregate sets of node embeddings corresponding to subgraphs. An example is the work in [54] where Duvenaud et al. introduced the so called "convolutional molecular fingerprints", a *sum-based* approach, where they create representations for subgraphs of molecular graphs by summing all the individual node embeddings in the subgraph. The node embeddings are generated using a variant of the Algorithm in Figure 2.12. Differently from the sum-based techniques where they sum the node embeddings for the whole graph, the *graph-coarsening* approaches, such as the ones of Deferrard et al. [55] and Bruna et al. [46], stack convolutional and "graph coarsening" layers. In the graph coarsening layers nodes are clustered together and the clustered node embeddings are combined using element-wise max-pooling. After clustering, the new coarser graph is again fed through a convolutional encoder and the process repeats. These approaches place considerable emphasis on designing convolutional encoders based upon the graph Fourier transform. Since naive versions of these encoders have complexity $O(|V|^3)$, with |V| the number of vertexes, the authors of [55] introduce an approximation of the encoders by using the Chebyshev polynomials. However, as stated in [44], the introduced approximations make the graph coarsening methods conceptually similar to the algorithm in Figure 2.12.

Regarding the taxonomy showed in the survey [43] for the graph embedding input in our work we focused on "graph with auxiliary information" with vector features representing properties of the nodes. For what concerns the output, instead, for our case was suitable to consider as objective the creation of "node embeddings" where for each node the embedding is a vector in a low dimensional space. We focused mainly on the "deep learning" embedding techniques because, as stated also in the survey, they are quite robust and effective and have been widely used in the field of graph embedding.

Graph Deep Learning

In Section 2.2.4 we said that we wanted to distinguish between deep learning techniques aiming to create embeddings and the ones that learn directly on graphs. We refer as Graph Deep Learning techniques to the latter ones. However, this distinction is not sharp and the graph embedding techniques can be seen just as an intermediate step towards the global task of learning from graphs. Indeed, the neighborhood aggregation approaches that can also incorporate task-specific supervision from node classification tasks can be seen as more general algorithms which apply directly deep learning over the graphs. They first learn how to encode the graph and then solve the specific ML task, such as classification, regression, etc., by applying more classical algorithms from ML literature. Indeed, we presented the GraphSAGE [14] model as a graph embedding technique which acts in an unsupervised manner. Nonetheless it can also be seen as a deep learning model that learns directly from graphs when incorporating supervised information such as labels for nodes. Also Deferrard et al. in their work [55] present their model not as an embedding algorithm but rather as a direct deep learning model on graphs which extends the concept of convolution from euclidean domains to irregular ones such as the graphs. The embedding step, however, as stated by [44], constitute an important part for these algorithms too, such that enables them to learn the proper representations for solving specific ML tasks. The models that rely on the Fourier transform, such as the one of Deferrard [55], are called spectral models. As stated in [11] a key criticism of spectral approaches is the fact that the spectral definition of convolution is dependent on the Fourier basis (Laplacian eigenbasis), which, in turn is domain-dependent. It implies that a spectral CNN model learned on one graph cannot be trivially transferred to another graph with a different Fourier basis. In their work the authors of [11] solved partially the problem by being able to generalize over graphs with different number of edges. However, when dealing with tasks such as graph classification, their model still requires to keep fixed the number of nodes, i.e. the different graphs which are classified can have different number of edges but require to have the same number of

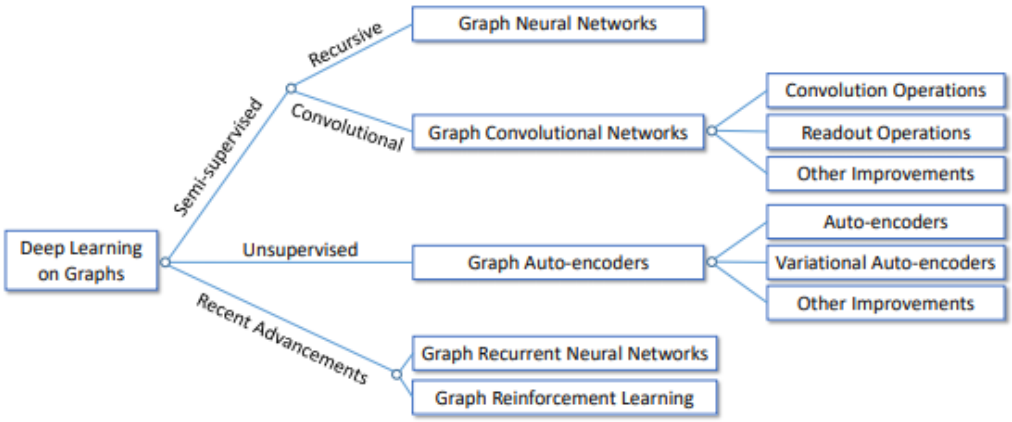

Figure 2.13: Categorization of deep learning methods on graphs
Figure taken from [57]. The authors divide the existing methods into three categories: semi-supervised, unsupervised and recent advancements. The semi-supervised methods can be further divided into Graph Neural Networks and Graph Convolutional Networks based on their architectures. Recent advancements include Graph Recurrent Neural Networks and Graph Reinforcement Learning methods.

nodes. Still in the context of spectral methods the authors of [56] attempted to advance deep learning for graph-structured data by incorporating another component: transfer learning. By transferring the intrinsic geometric information learned in the source domain, their approach can construct a model for a new but related task in the target domain without collecting new data and without training a new model from scratch.

In their work the authors of [57] proposed a categorization of the deep learning methods on graphs that can be seen in Figure 2.13. We worked mainly with the unsupervised deep learning methods,specifically with graph auto-encoders such as *node2vec* [15], and with the semi-supervised graph convolutional networks such as *graphSAGE* [14] and the convolutional neural networks in the spectral domain of [55]. It is important to highlight that in our work the algorithms which are categorized as semi-supervised in [57] can be and have been used in a supervised context. In semi-supervised approaches

only a few nodes have additional supervised inoformation such as node labels, i.e it is also used an amount of unlabeled data during training, while in the supervised cases all the nodes have such information. As we will see in Chapter 3 the graphs used in this work are the so called *Surface Networks* where the nodes have attributes given by terrain conformation while the labels for the classification task is given by their category (peak / non peak).

Chapter 3

Overview of the relevant techniques and graph learning architectures

3.1 Surface networks

In math by *surface* is intended a function f of one or more variables. In geography one is usually interested in those functions for which two of the independent variables denote location in a geographic domain; that is, functions f(u,v) where (u,v) denotes a point within a geographic coordinate system. A topographic surface in which altitude is a function of position is a convenient prototype. *Surface Networks* are an abstraction of the 2-dimensional surfaces by storing only the most important (also called fundamental, critical or surface-specific) points and lines in the surfaces.

Surface networks allow to abstract the Earth's surface and store its fundamental properties. These networks can be represented as graphs whose nodes and edges are extracted from the Earth's shape by finding its critical points. In math, the critical points can be classified as maxima (or peaks), minima (or pits) and saddles (or passes). If we define a surface's function $y = f(\mathbf{x})$, then we can think at the maxima as those points whose value of the function $f(\mathbf{x})$ is the highest compared to their neighbours. We can distinguish be-

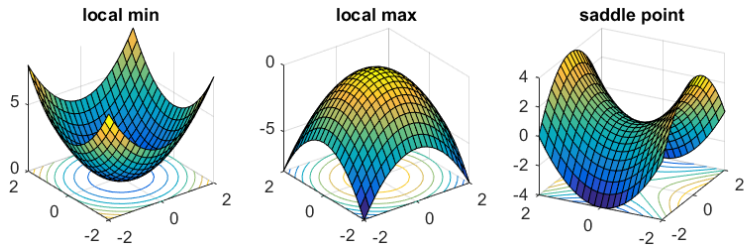

Figure 3.1: Critical Points

Figure illustrating the 3 types of critical points of a continuous surface.

tween local maxima and global maxima; local maxima are the points whose function value is the largest within a given range, while a global maximum is the point that has the highest value for the entire domain of the function. Accordingly, local minima are those points whose value of the function is the smallest within a given range, while the global minimum is the smallest for the entire function. Finally saddles are those points that are connected to two local (global) maxima and to two local (global) minima. The connection to the local maxima is given by following the two most steepest ascent paths starting from the saddle and reaching the maxima, while the local minima are reached by following the two most steepest descend paths. Examples of critical points for a 3-dimension surface defined by a function $z = f(x, y)$, such as the Earth, can be seen in Figure 3.1.

Surface networks capture the topological relations between the critical points of a continuous surface. In a surface network, every saddle is connected, at least, to two maxima and to two minima. The paths with the steepest ascents starting from a saddle connect it to the maxima, while the paths with the steepest descents connect it to the minima. The resulting graph of critical points and critical lines connecting them is termed *surface network* (Pfaltz 1976) [58]. An example of surface network for a portion of the Latschur Mountains in Western Carinthia, Austria can be seen in Figure 3.2. Surface networks represent special types of graphs with the vertexes set consisting of the critical points and the edges set consisting of the critical lines.

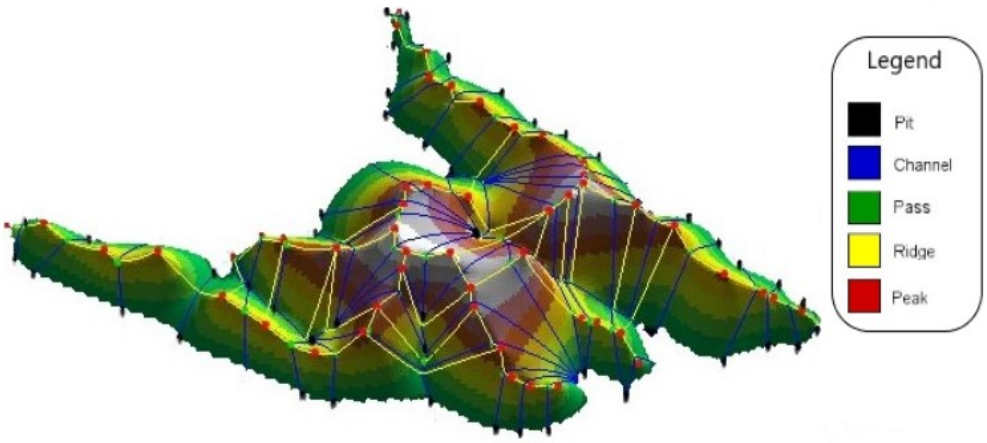

Figure 3.2: Surface Network

Figure illustrating a surface network. The critical points are evidenced with red for peaks (maxima), black for pits (minima) and green for passes (saddles). In morphology the connections between saddles and maxima points are called ridges and here represented with a yellow line, while the connections between saddles and minima are called channels, here showed with a blue line. Image taken from the work of Sanjay Rana and Jeremy Morley in [58].

500	520	510	500	520
540	620	590	580	530
500	610	500	570	520
490	590	550	540	490
470	500	520	510	500

Figure 3.3: DEM raster
Figure illustrating a portion of DEM raster containing elevations expressed in meters.

3.1.1 Building Surface Networks from Raster Data

Contemporary Geographical Information Systems (GISs) for representing Earth's surface are using as a main data structure the so called *digital elevation models (DEMs)*. A DEM can be defined as a raster (a grid of squares, also known as a heightmap when representing elevation) or as a vector-based triangular irregular network (TIN). We used as terrain representations mainly the rasters, i.e. matrices containing the elevations for the areas they were representing. Different sources for the DEMs can be cited, such as Laser Imaging Detection and Ranging (LiDAR) or Shuttle Radar Topography Mission (SRTM) missions [59]. By using DEMs, the critical points and their connections cannot be evaluated like in classical differential topology where we analyze the derivatives of the surface. Instead, DEMs are an approximation of surfaces and they are not continuous; indeed each cell of the matrix represent a different elevation, and moving from a cell to another leads to discontinuous changes. An example of a small portion of a DEM containing elevations can be seen in Figure 3.3.

Traditional methods for finding the critical points and their connections involve considering for each cell its eight direct neighbours, and based on the comparison among this neighbourhood it can be classified as : peak, pit, pass, ridge, channel, plane. A cell is considered a maximum (peak) if it is the highest one among its 8 neighbours, while it is a minimum (pit) if it is the lowest one. A cell is a saddle (or pass) if it is the highest point considering

the direction given by two of its neighbours that are not adjacent among them and it is the lowest point considering another direction given by other two non adjacent cells of its neighbours. An area composed by nine (or more) cells where all have the same elevation is a plane. Finally, the channels are composed by a sequence of cells surrounded by higher ones, while the ridges are a sequence of higher cells compared to the neighbours. These six basic morphometric classes can be identified with the eight-neighbours method, and an example can be seen in Figure 2.1.

The method of Shigeo Takahashi [12] for the creation of surface networks suggests that features like critical points and their connections come from the theory of differential topology and they should satisfy some topological formulas. The most important one is the *Euler-Poincarè formula* which states that the total number of critical points for a surface satisfies this property: #maxima - #saddles + #pits = 2. Here # means "number of". As said in [12] the eight direct neighbours method finds a set of critical points which does not satisfy this rule. Shigeo Takahashi proposed an algorithm for extracting critical points that preserves the validity of the Euler-Poincarè formula. The proposed method is based on the *Delaunay triangulation* for defining the neighborhood of a cell without incurring in unwanted critical points which happens with standard methods like the one proposed by Peucker and Douglas in [16]. The Delaunay triangulation is a subdivision of a set of points into a non-overlapping set of triangles, such that no point is inside the circumcircle of any triangle. In practice, such triangulations tend to avoid triangles with small angles. In the case study of DEMs the triangulation is defining for each cell of the matrix which of the adjacent cells is a neighbour, i.e. not all the adjacent cells are considered anymore neighbours. As we can see in Figure 3.4 each cell is surrounded by other eight cells but there is not a connection with all of them. Instead of extracting critical points directly from DEMs, Wood suggested in [60] a method for specifying bi-variate quadratic surface patches at raster points. These surface patches make possible to identify the directions of steepest descent and ascent and allowing then to identify the critical points and critical lines. The great benefit of the bi-variate quadratic surfaces is that they can

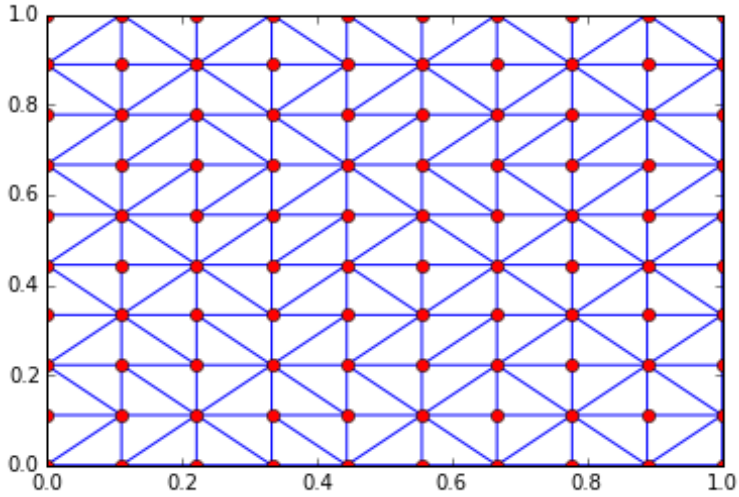

Figure 3.4: Delaunay Triangulation
Figure illustrating a portion of DEM and the Delaunay triangulation. The red dots represents the cells of the raster while the blu lines how the triangulation is setting the neighbours of each cell.

be fitted to windows of any size enabling to perform the analysis on a desired level of scale. However, this approach does not guarantee consistency of the extracted network. Still based on the idea of interpolating surfaces B. Schneider proposes in [61] a simpler bilinear interpolation scheme allowing a less flexible but more rigorous method in terms of continuity. These researches sometimes refer to the surface networks as *metric* or *weighted* surface networks. These versions assign a weight to the edges of the graph; the most common kind of weight is the difference of elevation between the two nodes considered.

3.2 Machine Learning Techniques

In this section we will see an overview of the main Machine Learning techniques we used for extracting peaks from DEMs.

3.2.1 Logistic Regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes. In ML logistic regression is the go-to method for binary classification problems (problems with two class values).

Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail, win/lose, true/false, etc.; these are represented by an indicator variable, where the two values are labeled "0" and "1". The dependent variable has a relationship with a set of independent variables called regressors or predictors. The goal of logistic regression is to find the best fitting model to describe the relationship between the dependent variable (also called dichotomous characteristic of interest or response or outcome variable) and the set of independent variables. The *probability* or *odds* of the response taking a particular value is then modeled based on the combination of values taken by the predictors and the parameters of the model. Then, we define *log-odds* l, i.e. the logarithm of the odds, as a linear combination of the predictor x_1 with the parameters β_i, including the constant term β_0. For example, for a model with one predictor x_1 the log-odds is computed as:

$$l = ln(o) = \beta_0 + \beta_1 x_1 \tag{3.1}$$

where the coefficients β_i are the parameters of the model and o the odds. The corresponding odds, i.e. the outcomes of the response, are then the exponent:

$$o = b^{\beta_0 + \beta_1 x_1} \tag{3.2}$$

where b is the base of the logarithm and exponent.

In classification we are determining the probability of an observation to

be part of a certain class or not. Therefore, we wish to express the probability with a value between 0 and 1. A probability close to 1 means the observation is very likely to be part of that category. In order to rescale the odds values to be between 0 and 1, the probability is expressed using the following equation:

$$y = \frac{e^{\beta_0+\beta_1 x_1}}{1 + e^{\beta_0+\beta_1 x_1}} \tag{3.3}$$

where the equation above is defined as the *sigmoid function* with b set as e (Euler's number) in the original odds and y is the predicted output.

The coefficients β_i of the logistic regression algorithm are estimated from the training data by means of the maximum-likelihood estimation. In statistics, maximum likelihood estimation (MLE) is a method of estimating the parameters of a statistical model, given observations. Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms. The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients β_i that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class). The sum of errors for all the predictions constitutes the so called cost function which represents how far the model coefficients β_i are from the optimal ones, i.e. the values of the coefficients for which the model would be able to perfectly classify the training samples. When the predictions are totally wrong, the cost function outputs a higher number; while, if the predictions are closer to the ones in the training data the cost function value reduces. When tuning the coefficients β_i the loss function is useful to understand if the change can lead to better results. A minimization algorithm is then used to optimize the best values for the coefficients such that to reduce at the minimum the cost function for the training data. This is often implemented in practice using efficient numerical optimization algorithms such as Stochastic Gradient Descent (SGD), i.e. a gradient-based optimization method for

minimizing the cost function. Various improvements of the classic Stochastic Gradient Descent (SGD) algorithm emerged during the years: an example is Adam [62] which uses adaptive learning rates and second-order curvature informations.

3.2.2 Node2vec

As we saw in Section 2.2.4 *node2vec* [15] is an algorithmic framework for representational learning on graphs. Given any graph, it can learn continuous feature representations for the nodes, which can be then used for various downstream machine learning tasks, such as node classification. Learning representations from highly structured objects such as graphs is useful for a variety of machine learning applications. Besides reducing the engineering effort, these representations can lead to greater predictive power.

This fundamental idea of the algorithm is closely related with some recent advancements in representational learning for natural language processing which opened new ways for feature learning of discrete objects such as words. Among these, the Skip-gram model [63] aims to learn continuous feature representations for words by optimizing a neighborhood preserving likelihood objective. The Skip-gram technique scans over the words of a document, and for every word it aims to embed it such that the word's features can hold properties about nearby words (i.e., words inside some context window). The word feature representations are learned by optmizing the likelihood objective using Stochastic Gradient Descent as optimization function [63]. The Skip-gram objective is based on the distributional hypothesis which states that words in similar contexts tend to have similar meanings [64]. That is, similar words tend to appear in similar word neighborhoods.

With a similar approach node2vec aims to learn features representations for the nodes of the graph such that they represent the structure of the network. By choosing an appropriate notion of neighborhood, node2vec can learn representations that organize nodes based on their network roles and/or communities they belong to. This framework learns to generate for each node of a graph low-dimensional representations, often called *embeddings* or

features, which can then be used for downstream machine learning tasks. The authors formulate the feature learning in networks as a maximum likelihood optimization problem, i.e. to maximize the likelihood of preserving network neighborhoods of nodes in a d-dimensional feature space, where d is the number of features we want to use for representing the node.

Feature Learning

Given a network $G = (V, E)$ the authors define as $f : V \rightarrow \mathbb{R}^d$the mapping function from nodes to the feature representations we aim to learn for downstream prediction task. Here d is a parameter specifying the number of dimensions of our feature representation. Equivalently, f is a matrix of size $|\mathrm{V}| \times d$ parameters. For every source node u $\in$ V , they define $N_S(u) \subset$ V as a network neighborhood of node u generated through a neighborhood sampling strategy S. The objective function is defined as:

$$\max_f \sum_{u \in V} logPr(N_S(u)|f(u)) \tag{3.4}$$

By varying f, the embedding function, we want to maximize the log-probability of observing a network neighborhood $N_S(u)$ for a node u conditioned on its feature representation, i.e., find the best f such that the learnt features represent as much as possible the most representative neighborhood of the source node u.

This is method based on the Skip-gram architecture which have been originally developed in the context of natural language. Given the linear nature of text, the notion of a neighborhood can be naturally defined using a sliding window over consecutive words. Networks, however, are not linear, and thus a different notion of a neighborhood is needed. To resolve this issue, the authors propose a randomized procedure that samples many different neighborhoods of a given source node u. The neighborhoods $N_S(u)$ are not restricted to just immediate neighbors but can have vastly different structures depending on the sampling strategy S.

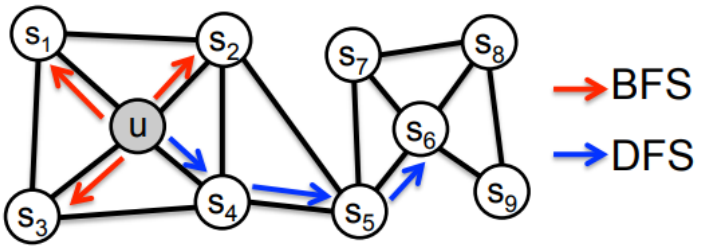

Figure 3.5: BFS and DFS Strategies
Figure taken from [15] illustrating the difference between BFS and DFS strategies.

Sampling strategy

In classic search strategies there are two extreme, opposite, approaches for generating neighborhood set(s) N_S of k nodes: Breadth First and Depth First. With Breadth-first Sampling (BFS) the neighborhood N_S is restricted to nodes which are immediate neighbors of the source; with Depth-first sampling (DFS), instead, the neighborhood consists of nodes sequentially sampled at increasing distances from the source node.

The breadth-first and depth-first sampling represent extreme scenarios in terms of the search space they explore leading to interesting implications on the learned representations. As said by the authors, there are two kinds of similarities when dealing with graphs: homophily and structural equivalence. Under the homophily hypothesis nodes that are highly interconnected and belong to similar network clusters or communities should be embedded closely together (e.g., nodes s1 and u in Figure 3.5 belong to the same network community). In contrast, under the structural equivalence hypothesis nodes that have similar structural roles in networks should be embedded closely together (e.g., nodes u and s6 in Figure 3.5 act as hubs of their corresponding communities). The authors highlight how BFS and DFS strategies play a key role in producing representations that reflect either of the above equivalences. In particular, the neighborhoods sampled by BFS lead to embeddings that correspond closely to structural equivalence. In DFS, the sampled nodes more accurately reflect a macro-view of the neighborhood which is essential in

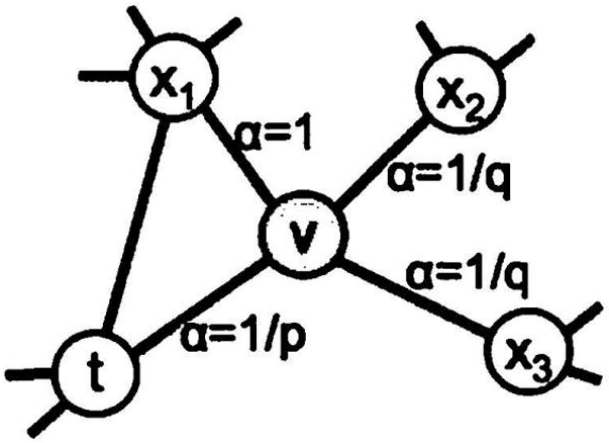

Figure 3.6: Random Walk
Figure taken from [15] illustrating a random walk procedure. α is controlling the transition probability.

inferring communities based on homophily. Based on the above sampling differencies the authors design a flexible neighborhood sampling strategy which allows to smoothly interpolate between BFS and DFS. They achieved this by developing a flexible biased **random walk** procedure that can explore neighborhoods in a BFS as well as DFS fashion.

The neighborhood sampling is controlled by two parameters, p and q, defining the probabilities of of revisiting the same nodes or moving farther from the source node t. The authors define the transition probability on edges *(v,x)* as $\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$, where

$$\alpha_{pq}(t,x) = \begin{cases} 1/p & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ 1/q & \text{if } d_{tx} = 2 \end{cases} \tag{3.5}$$

and d_{tx} denotes the shortest path distance between nodes t and x. Note that d_{tx} must be one of 0, 1, 2, and hence, the two parameters are necessary and sufficient to guide the walk.

Parameter p, also called *return parameter*, controls the likelihood of im-

mediately revisiting a node in the walk. Setting it to a high value ensures that we are less likely to sample an already-visited node in the following two steps. Parameter q, instead, allows the search to differentiate between "inward" and "outward" nodes. As we can see in Fig. 3.6, if q $>$ 1, the random walk is biased towards nodes close to node t.

The overall process of *node2vec* can be resumed in three phases: 1) preprocessing to compute transition probabilities, 2) random walk simulations and 3) optimization of the embedding function f using SGD. More details can be seen in the paper of the authors [15], especially for the random walks.

3.2.3 GraphSAGE

As we saw in Section 3.2.2, the basic idea behind node embedding approaches is to use dimensionality reduction techniques to distill the high-dimensional information about a node's graph neighborhood into a dense vector embedding [14] to be then used for downstream machine learning tasks such as classification or regression. We saw that node2vec learns to create embeddings for the nodes of a graph based on their neighborhood. A different approach, named *GraphSAGE* [14] (SAmple and aggreGatE), apart from the position of the node in the graph, leverages the signals, i.e. properties, of the nodes.

Most of the existing approaches require that all nodes in the graph are present during training of the embeddings; these previous approaches are inherently *transductive* and do not naturally generalize to unseen nodes. However many applications require generating embeddings for unseen nodes, or entirely new (sub)graphs. The authors present *GraphSAGE* as a general *inductive* framework that exploits node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data. Instead of training individual embeddings for each node, the model learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. This inductive capability is essential when dealing with graphs or nodes which are evolving (as may be the users of social media) or are unseen during training (for example new posts of the users).

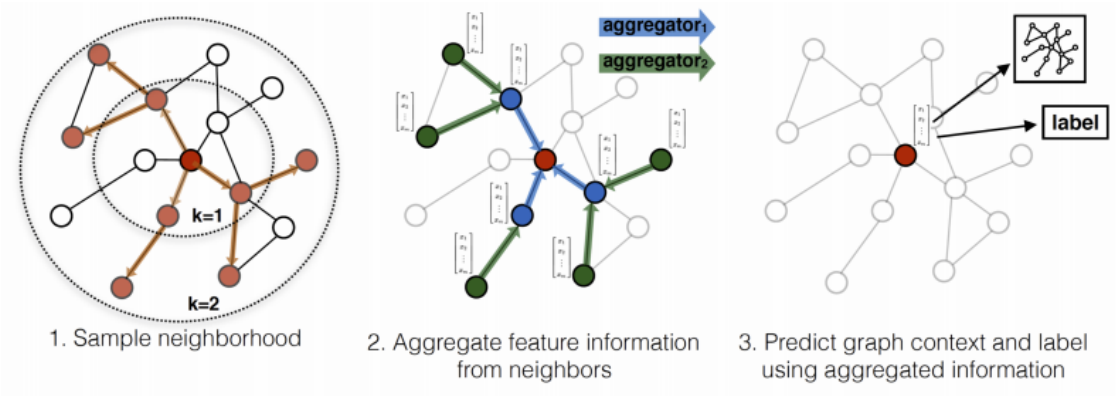

Figure 3.7: GraphSAGE sample and aggregate approach
Visual Illustration of the GraphSAGE sample and aggregate approach.
Image taken from [14]

As highlighted by the authors, by leveraging node features (e.g., text attributes, node profile information, node degrees) GraphSAGE can behalf as embedding function which generalizes to unseen nodes. By incorporating node features in the learning algorithm, it can simultaneously learn the topological structure of each node's neighborhood as well as the distribution of node features in the neighborhood [14]. The key idea behind this approach is that it can learn how to aggregate feature information from a node's local neighborhood. An example of the process of sampling and aggregation can be seen in Figure 3.7.

Embedding Generation

Before describing how the parameters are learned, suppose the model is already trained and we can then generate the embeddings for the nodes. In particular we assume that we have learned the parameters of K aggregator functions (denoted as $AGGREGATE_k, \forall k \in \{1, ..., K\}$ in Figure 3.8) which are aggregating information from node neighbors. Also we have to assume that a set of weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$ have been already learned; they are used to propagate information between different layers of the model,

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 3.8: GraphSAGE Algorithm - Embedding Generation
GraphSAGE algorithm illustrating the steps for creating the embeddings

also called "search depth" by the authors.

As shown in Figure 3.8 we assume that in input are provided the entire graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and the features for all the nodes $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$. We will denote with k the current step in the outer loop (which the authors also refer to as the depth of the search) and with $\mathbf{h}^k$ the node's representation at this step. The algorithm proceeds as follows: First, each node $v \in \mathcal{V}$ aggregates the representations of the nodes in its immediate neighborhood, $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(u)\}$, into a single vector $\mathbf{h}_{\mathcal{N}(u)}^{k-1}$. This aggregation step depends on the representations generated at the previous iteration of the outer loop (i.e. $k-1$), where for $k = 0$ the representations coincide with the initialization vectors, i.e. the nodes initial features. The aggregation of the neighbor representations can be done by a variety of aggregator architectures (denoted by the $AGGREGATE$ placeholder in the algorithm), which we will see more in detail in the following sections. After aggregation the algorithm proceeds by concatenating the node's current representation $\mathbf{h}_u^{k-1}$ with the aggregated neighborhood vector, $\mathbf{h}_{\mathcal{N}(u)}^{k-1}$; this concatenated vector is then fed through a fully connected layer with nonlinear activation function σ, which transforms the representations to be used at the next step of the algorithm, i.e. it generates $\mathbf{h}_v^k, \forall v \in V$, the vector representing the embedding at step k. The

final embedding at step K is denoted by the authors as $z_v \equiv \mathbf{h}_v^K$.

Neighborhood definition

Given the set $\{u \in V : (u, v) \in \mathcal{E}\}$ representing all the nodes to which v is connected, in Figure 3.8 the authors use the expression $\mathcal{N}(v)$ to denote the set of sampled nodes representing the neighborhood of node v. They define $\mathcal{N}(v)$ as a fixed-size uniform draw from the set of all connected nodes. Also, at each iteration k, is drawn a different uniform number of samples which allows to control the memory usage and the expected time of a single batch.

Learning the parameters of GraphSAGE

In the work where the authors present GraphSAGE [14] there is not an explicit definition of the loss function for supervised tasks, such as is the purpose of our work. Indeed they present the work mostly for unsupervised tasks (i.e. without having labels on the nodes) by letting the reader know that the model is adaptable for supervised tasks by modifying the loss function with a cross-entropy one. However, the same authors present in another work a review of different Graph Deep Learning models [44] where they also show how is it possible to define the loss function for GraphSAGE in a supervised manner.

Assume to have a binary classification label (such as True or False, Female or Male, etc..) $y_i \in \mathbb{Z}$ associated with each node. To learn to map nodes to their labels, the embedding vectors, $\mathbf{z}_i$, are fed through a logistic (or sigmoid) function $\hat{y} = \sigma(\mathbf{z}_i^\intercal \theta)$ where θ represents the vector of trainable parameters. With the vector of parameters θ we include both the $\mathbf{W}^k$, $\forall k \in \{1, ..., K\}$ and the parameters for the aggregator functons. We can then compute the cross-entropy loss between these predicted class probabilities $\hat{y}$ and the true labels y as follows:

$$\mathcal{L} = \sum_{v_i \in V} y_i log(\sigma(\mathbf{z}_i^\intercal \theta)) + (1 - y_i) log(1 - \sigma(\mathbf{z}_i^\intercal \theta)). \tag{3.6}$$

The gradient is computed according to equation 3.6 and can be then back-propagated through the encoder to optimize its parameters.

Aggregator Architectures

When dealing with networks, the nodes' neighbors have no natural ordering; thus the aggregator functions we introduced above must operate over an unordered set of vectors. An aggregator function is said to be symmetric when it is invariant to the permutations of the input nodes; this ensures that the neural network model can be trained and applied to arbitrarily ordered node neighborhood feature sets. The authors examined three different types of aggregator functions:

1. **Mean Aggregator**: the first candidate aggregator function is the mean operator which is simply taking the elementwise mean of the vectors in $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(u)\}$.

2. **LSTM Aggregator**: the authors introduce also a more complex aggregator based on LSTM (Long Short Term Memory) Neural Networks [65], essentially neural networks which are recurrent and are having loops, i.e. the output of the network at the previous step is used to feed again the network itself at the current time step. Moreover, apart from the loops, LSTM are also able to keep some memory of the past iterations, adapting between shorter and longer windows of steps back in time (the part of the name long-short refers to this ability). It is important to highlight that LSTM are not symmetric, so they are not permutation invariant. The authors adapt LSTMs to operate on an unordered set by simply applying the LSTMs to a random permutation of the node's neighbors.

3. **Pooling Aggregator**: finally, the authors examine the pooling aggregator which is both symmetric and trainable. In this approach, each neighbor's vector is independently fed through a fully-connected neural network; following this transformation, an elementwise max-pooling

operation is applied to aggregate information across the neighbor set:

$$AGGREGATE_k^{pool} = max(\{\sigma(\mathbf{W}_{pool}\mathbf{h}_{u_i}^k + \mathbf{b}), \forall u_i \in \mathcal{N}(u)\}), \quad (3.7)$$

where max denotes the element-wise max operator and is a nonlinear activation function. In principle, the function applied before the max pooling can be an arbitrarily deep multi-layer perceptron, but the authors focused on simple single-layer architectures. As we saw in Section 2.2.1, the multi-layer perceptron can be thought of as a set of functions that compute features for each of the node representations in the neighbor set. By applying the max-pooling operator to each of the computed features, the model effectively captures different aspects of the neighborhood set. Apart from the max operation in equation 3.7 other symmetric vector function, such as mean-pooling, can be applied.

Chapter 4

Learning to find mountains

4.1 Methods under evaluation

In this Section, we explain in more detail the heuristic methods used in this thesis to extract peaks from DEM data, describe their parameters and their input and output format.

4.1.1 Landserf Peak Classification

This algorithm is part of the Landserf Tool [66] and it focuses only on peak extraction and does not take into account scale during the analysis. It defines a peak as a point with at least a certain elevation and surrounded by points with an elevation lower at most by a given amount. The user has to set the following parameters:

- **Minimum Height of a Peak**. This value represents the minimum elevation (in meters) a point must have to be considered as a peak; all points that do not satisfy this criterion are discarded. This is useful because it allows to filter out some points that can resemble a peak but are too low to really be a peak. In the equations below, the parameter is represented as min_{height}.

- **Minimum Drop Surrounding Peak**. This value contributes to determine whether a given point is to be considered as a peak summit,

as part of the extent of a peak or none of the two classes. It represents the maximum elevation difference a point can have with respect to a candidate peak, to be considered as part of its extent. In the equations below, it is represented as min_{drop} .

The algorithm performs the search by iterating over each cell of the DEM and tests if the following rules are complied:

$$E_{point} \geq min_{height} \text{ and } E_{point} \geq E_{min} + min_{drop} \tag{4.1}$$

with E_{min} being the minimum elevation in the DEM. If the rule is respected then the cell is considered a valid candidate and the algorithm starts analyzing its neighbors sorted by decreasing elevation and proceeds as follows:

- if a neighbor has a higher elevation it stops the analysis and goes on with the next point. This cell can be part of the extent of another peak but cannot be considered as a peak itself
- all neighbors whose elevation E_{point} respect the Equation 4.2 are added to the candidate peak extent and their neighbors are checked recursively too. The extent will be later used to calculate the peak summit location.
- the analysis continues until there are no remaining points or the current relative drop is greater than the user-defined Minimum Drop parameter.

At the end, a list of cells is formed that are part of peak extent and, only if the last relative drop is greater than Minimum Drop, the summit position is calculated: the center of the extent is determined by averaging the position of all the components; then the point with the highest elevation and nearest to the center is considered as the peak summit.

$$0 \leq E_{candidatePeak} - E_{point} \leq min_{drop} \tag{4.2}$$

4.1.2 Landserf Surface Classification

This method is still part of the Landserf Tool and it aims at classifying a surface into 6 categories: pits, channels, passes, ridges peaks and planar regions. For identifying these categories in a surface it first builds a Metric Surface Network [60], which is a Surface Network similar the one we built but achieved through a different method relying on linear interpolation of heights. The method works with 3 parameters:

- **windows size** defines number of cells along one side of the square window centered on the currently analyzed point,with values ranging from 7 to 75, sampled with a step of 2;
- **curvature tolerance** value determines, instead, how convex/concave ('sharp') a feature must be before it can be considered part of any class, with values corresponding to 0.1, 0.5, 2, 4, 6. Curvature is recorded as a dimensionless ratio, with typical tolerance values ranging from 0.1 to 0.5. Larger values tend to increase the proportion of the surface classified as planar, leaving only the sharpest features identified.
- **distance decay**, to determine the importance of the cells near the center of the window with respect to those at the edges, with values ranging from 0 to 4 with a step of 1.

4.2 Input Data

In Chapter 3 we saw how surface networks can be considered a useful representation of the topological properties of the shape that build the graphs are the Digital Elevation Models. The source we used for our files is the Shuttle Radar Topography Mission (SRTM) DEM provided by NASA [59]. SRTM DEM data are organized into a regular grid containing for each cell the elevation for the given coordinate, and they can be distinguished among STRM1 DEM and SRTM3 DEM. The distinction comes from the different resolutions they are subject to, i.e. how dense are the measurements available for a given area. DEM1 grids are relative to resolutions of 1 arc-second,

while DEM3's resolution is in the order of 3 arc-seconds. Figure 3.3 showed an intuitive example how the elevation data can be organized in grids.

For our work we focused mainly on the Switzerland territory and we used both the SRTM1 DEMs with resolution of 1 arch-second and the STRM3 with 3 arch-second, that in areas realtively far from the poles correspond approximately to 27m-30m for DEM1, i.e. we have a measurement of elevation every 27m-30m, and 70m-90m for DEM3. The data collected in SRTM1 DEM is then divided into a series of tiles of 3601x3601 cells for each tile and every one is spanning 1 degree in latitude and 1 degree in longitude. Accordingly, for STRM3, we have 1201x1201 cells per tile. Tile N47E008, for example, contains the elevations arranged in the grid for the area between longitude from 8°E to 9°E and latitude from 47°N to 48°N.

4.3 Algorithm for building the graph

Among the methods illustrated in Chapter 3 for building surface networks from DEMs we opted for the algorithm of Shigeo Takahashi [12]. However, we introduced some modifications to tackle a problem of misclassified critical points. As Shigeo Takahashi said, classical methods like eight neighbours fail in extracting a number of critical points that respects the *Euler-Poincarè formula*. The main problem in the eight-neighbours method is the appearance of many unwanted saddles due to lack of smoothness of DEMs, making the formula invalid. The proposed algorithm in [12] tackles this problem by determining a unique surface interpolation from the given samples. For this purpose *triangulation* is used since it offers the most commonly adopted linear interpolation and does not incur unwanted critical points. A suggested triangulation in the paper is the *Delaunay* because it avoids thin triangles that are undesirable for sound linear interpolations. Essentially, by introducing a triangulation the set of neighbours for each cell may reduce to a smaller number, avoiding the appearance of the unwanted passes. In fact, with triangulation we can define the neighbours of each sample and then introduce the criteria for critical points. If we consider a point P its neighbours are

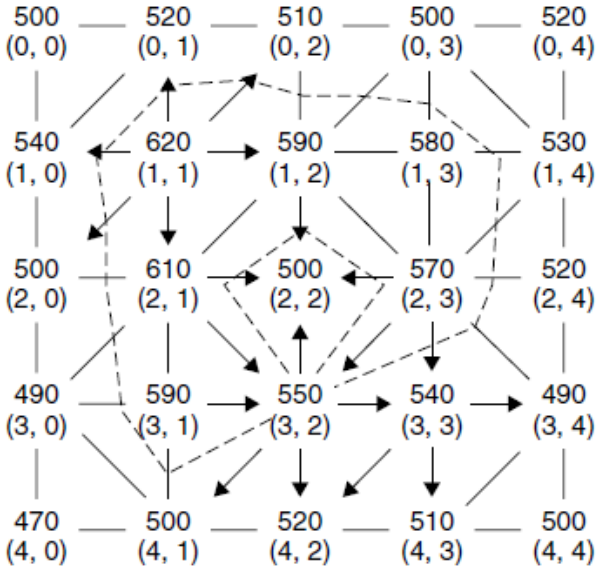

Figure 4.1: Triangulated DEM
Figure illustrating how the neighborhood is established between the cells. Image taken from Shigeo Takahashi's work in [12].

those points that are adjacent to P in the triangulation. An example referred to the DEM raster in Figure 3.3 can be seen in Figure 4.1. The method we decide to use is based on the work of Takahashi [12] but it will not implement the proposed triangulation due to the misclassified saddles as local maxima.

4.3.1 Criteria for finding Critical Points

Shigeo Takahashi's algorithm considers a circular list of neighbours for each point P in a counter-clockwise (CCW) order with respect to xy-coordinates (or latitude/longitude if based on the DEM's coordinates). The criteria for critical points is as follows:

peak $|\Delta_+| = 0, \quad |\Delta_-| > 0, \quad N_c = 0$
pit $|\Delta_-| = 0, \quad |\Delta_+| > 0, \quad N_c = 0$
pass $|\Delta_+| + |\Delta_-| > 0, \quad N_c = 4$

where the following notation are used:

n the number of neighbors of P

Δ_i the height difference between $P_i(i = 1, 2, ..., n)$ and P

Δ_+ the sum of all positive $\Delta_i(i = 1, 2, ..., n)$

Δ_- the sum of all negative $\Delta_i(i = 1, 2, ..., n)$

N_c the number of sign changes in the sequence Δ_1, Δ_2,..., Δ_n, Δ_1

By using these criteria in combination with Delaunay triangulation it is possible to maintain the Euler-Poincarè formula. However, for our graph we decided to not use the triangulation, instead we kept all the eight neighbours and then applied the criteria. The reason for this choice is due to the fact that with triangulation more than 20% of the saddles for the different areas we analyzed became local maxima. If our purpose would be a study of the semantics of these graphs the correctness of the topological rules would be fundamental. Instead, we are trying to understand if it is possible to categorize the nodes of the graph as being or not a mountain peak. Using the information about the category of the nodes in the features that carachterize them, i.e. knowing in advance what kind of critical point is a node (minimum, maximum or pass), increases the semantic knowledge we have about the graph. Instead, including many saddles in the set of local maxima would make the feature of the category for the nodes less important and less discriminative for the learning purpose. We decided, then, to build the graphs by using the above criteria for distinguishing among the different types of critical points but considering always the eight direct neighbours instead of making a selection based on triangulation. When considering a cell and its eight direct neighbours we may refer to them as *patches*. An example of how triangulation would incorrectly assign a maximum to a patch that instead would not be a critical point can be seen in Figure 4.2.

Finally, since we said that the data collected by STRM is organized in different tiles, particular attention should be given to the cells on the borders of the different grids. Indeed, each cell in the corner has only three neighbours, while each cell on the border (except the corner) has five neighbours. To tackle this problem we padded each tile's border with the cells from the adjacent tiles. For example, considering the grid of 3601x3601 cells for the tile N47E008, the cell at position (0,0) has as neighbours inside its tile the cells (0,1), (1,1) and (1,0). Then, to these internal neighbours are

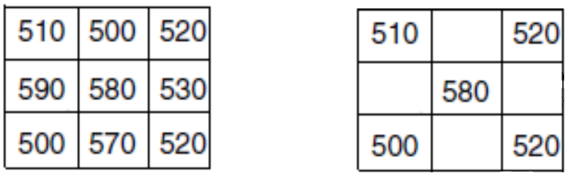

Figure 4.2: Incorrect classification of cell with triangulation
Figure illustrating how the cell with elevation 580, which is not belonging to any morphological class, would be added to the set of maxima in case of the illustrated type of triangulation.

added the ones coming from the adjacent tiles, in this case the cell at position (3600,3600) from grid referring to N48E007, the cells (0,3600) and (1,3600) from N47E007 and the cells (3600,0) and (3600,1) from N48E008. Specular approaches are adopted for the other corners and borders of the tiles.

4.3.2 Handling degenerate critical points and paths

There are two kinds of degenerate critical points that can arise when extracting them from DEMs: *level regions* and *duplicate passes*.

The level regions, or as we said in Chapter 2.1 the planes, are those patches that all have the same altitude. These kind of regions can extend beyond the standard nine cells patch. They are the result of the discrete quantization that leads to limited precision of the height values. A simple solution may be to consider the entire group of points having the same elevation as a single point. However, as suggested by S. Takahashi, this may not be a good idea if the flat area is surrounding a critical point in its interior, like in Figure 4.3(a). Instead, a second ordering is introduced, in the sense that when there is a tie between the elevations we consider a second factor for deciding which cell should be considered as higher. Similarly to the proposed method, in our implementation we used lexicographical ordering with respect to the xy-coordinates, where for x and y we can think at the row and column index inside the raster. Suppose, for example that cells (34,121) and (35,122) from the same grid have the same altitude 765m. In this case the second cell is considered as higher. Since the DEM is a set of samples

Figure 4.3: Level region

A level region: (a) a level region surrounding a pit and (b) the effect of introducing a second ordering. Image taken from S. Takahashi's work [12]

on a single-valued function $z = f(x, y)$, there are no two samples that have identical xy-coordinates. Indeed, if there is a tie for the x coordinate if the cells are on the same row of the grid, they cannot be on the same column, hence y would be different. Although this solution depend on the choice of x and y ordering, it enables uniform data manipulation by converting the degenerate critical points to non-degenerate ones.

The second type of degenerate case are the duplicate passes from which is possible to extract two or more passes. With the criteria for the critical points of Takahashi [12] that we modified, each saddle is supposed to be connected to two maxima and to two minima points. Starting from the pass and following the two steepest ascents we would reach the two maxima while following the steepest descents we would reach the two minima. Degenerate saddles, instead, can be connected to three or four maxima and three or four minima respectively. In the case of three minima and maxima it means that there are six direct neighbours of the saddle whose path lead to a critical point; the three steepest descents lead to the minima, while the three steepest ascents lead to the maxima. The same consideration is valid for four minima and four maxima. Obviously, due to the arrangement of the raster as a matrix is not possible to have more than eight neighbours for a cell. The method of Shigeo Takahashi tackles this problem by splitting the degenerate saddles in two or three passes, depending on the number of changes in the sign. It modifies then also the criterion for the passes as:

pass $\quad |\Delta_+| + |\Delta_-| > 0, \qquad N_c = 2 + 2m (m = 1, 2, ...)$

This problem, however, is not an issue because respecting the Euler-Poincarè formula is not our main concern. We kept then using saddles with

more than four critical points connected.

4.3.3 Connecting critical points

We said that a surface network is a graph constituted by a set of vertexes, the critical points, and a set of edges, the critical lines, connecting the critical points. The ridges and channels constitute the physical path connecting the critical points. The ridges connect saddles to peaks (maxima) while the channels connect them to the pits (minima). The last type of morphological patch, the plane, is instead resolved by the second ordering factor and its never present in our graphs. Saddles are always connected to at least two maxima and two minima, with a maximum of four maxima and four minima. Instead there is never a connection between maxima and minima or between two saddles.

The graph is intended to be undirected, i.e. all the edges are bidirectional. First, we find all the locations for the critical points by applying the Takahashi's modified criteria. Considering a DEM we scan all the grid and consider always a cell and its eight neighbours and apply the crtieria to decide if it is a critical point and which type it is. Then, for finding the connections between the saddles and the minima and maxima we analyzed the adjacent cells to the ones in which we evaluated the presence of a saddle. Suppose P_0 is one of these locations containing a saddle. Then, in the neighborhood of P_0 there are other eight cells, $P_1, P_2, ..., P_8$. We ordered these cells such that they would constitute a circular sequence around P_0; consider, as an example, the following clockwise ordered sequence based on the coordinates of the grid starting form the upper-left one $\{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$. Then, in this sequence we look for the subsequences constituted by the higher neighbours and the ones constituted by the lower elevations. Consider, as an example, P_0's elevation as 700m, while the ones of the adjacent cells, in order from P_1 to P_8 as the sequence $\{710, 715, 690, 700, 720, 680, 675, 720\}$. Then among this sequence we can find the following subsequences of higher points: $SH_1 = \{P_8, P_1, P_2\}$ with elevations $\{720, 710, 715\}$ and $SH_2 = \{P_4, P_5\}$ with elevations $\{700, 720\}$. The tie about considering P_4 higher or lower than P_0

720	710	705	700	680
730	710	715	690	650
750	720	700	700	660
740	675	680	720	700
700	670	665	690	750

Figure 4.4: Higher and lower subsequences
Circled with blue we have the cell where the saddle is supposed to be located. Then around it there are four subsequences composed by the adjacent cells, two with higher elevations and are circled in red, two with lower elevations, circled in green.

has been resolved with the second ordering. If the matrix coordinates start with (0,0) in the upper-left corner and we increase the values of the coordinates by moving down for the rows and moving right for the columns, then P_4 results being a higher cell. The lower sequences are, instead, the following: $SL_1 = \{P_3\}$ with elevations $\{690\}$ and $SL_2 = \{P_6, P_7\}$ with elevations $\{680, 675\}$. An illustration can be seen in Figure 4.4.

Among these subsequences we look for the highest value when dealing with higher sequences and for the lowest when dealing with lower sequences. These chosen cells will be the starting point for, respectively, steepest ascend path and steepest descend path. By applying this procedure in our example the subsequences reduce to $SH_1 = \{P_8\}$, $SH_2 = \{P_5\}$, $SL_1 = \{P_3\}$ and $SL_2 = \{P_6\}$. For finding the paths to the critical points we look for the steepest ascend path for the cells in the higher sequences and for the steepest descend path for the lower sequences. Consider, for example, the cell of P_8 which constitute the starting point for the path to a maxima. For finding the steepest ascend we check the elevation of all its neighbours and choose as part of the path the one with the highest altitude. In this case would be the cell with 750 m. Then again we look at the adjacent cells to the last one added to the path and opt for the highest one. We continue applying this procedure and add cells to the path until we reach the location of a

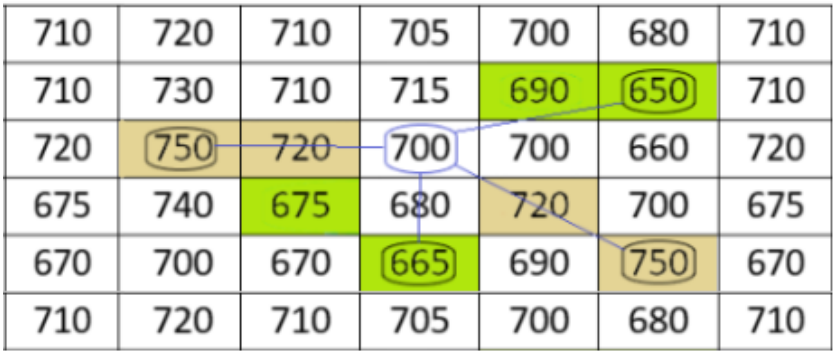

Figure 4.5: Paths and connections to the critical points
This figure shows in brown the ridges, i.e. the paths from the saddle (circled in blue) to the maxima and in green the channels, i.e. the paths to the minima. The blue lines represent the edges in the graph.

maxima found previously. For simplicity suppose that the cell is already the cell of a maxima. Similar procedure is applied for finding the connected minima. Figure 4.5 illustrates how this procedure can find the paths to the four connected critical points.

Notice that the edges connecting the critical points don not follow strictly the paths. Indeed, surface networks graphs are an abstraction and the edges cannot follow the physical lines; edges are delimited by the coordinates of the saddle and the coordinates of the critical point. We can also see that the steepest ascend paths are part of the ridges, while the steepest descends are the channels. This procedure is highly inspired by the method of Shigeo Takahashi for splitting the saddles.

4.3.4 Degenerate paths

Due to the lack of smoothness of the shape given by the DEM it may happen, as we can see in Figure 4.6, that both steepest ascend (descend) paths reach the same maxima (minima). In that case we removed the double edge and forced the connection with the closest maxima (minima) in terms of distance of cells inside the raster and which is not already present in the saddle edges. By making this reassignment of the duplicated edges we kept a slight recur-

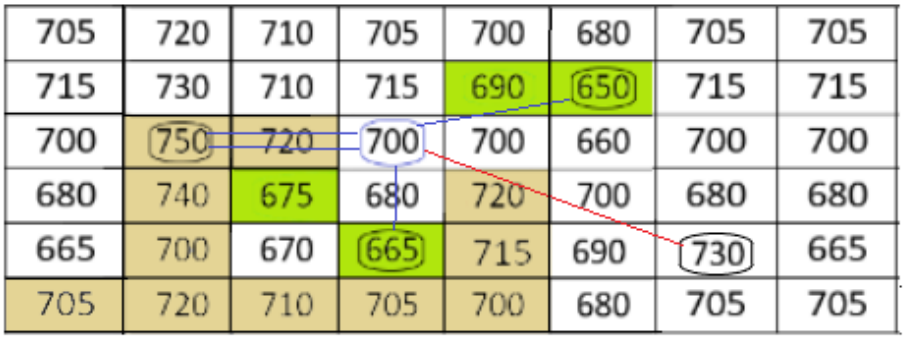

Figure 4.6: Rearrangement of edges
The blue lines represent the edges of the graph computed following the paths. The red line represents the new connection with a local maxima that was not reachable with the steepest ascent path.

rent structure inside the graph in the sense that we know for sure that all the saddles have four connected critical points, two maxima and two minima. The case with six or eight critical points can be easily reduced to four by splitting the saddle into two saddles with coincident location but different paths for different critical points. Keeping this small recurrence in the graph revealed to be useful when dealing with Deep Learning methods for graphs that require to have a common structure among the nodes. However, except of the saddles, the peaks and the pits have a really unpredictable number of connections, starting from one edge to even more than a hundred for few of them.

4.3.5 Enrichment of the graph with features

The graph presented until now reflects a surface network (even though it is not respecting the Euler-Poincarè rule). It contains a set of vertexes, the critical points, and a set of edges, built over the critical lines. When dealing with Deep Learning for graphs some approaches require just the structure of the graph as input to learn from, but some other techniques can use some signals on the nodes or on the edges for better learning the properties of the

nodes and the graph.

Signals on the edges

For methods that can handle weights on the edges, like *node2vec* [67], we used the slope as a representation of the steepness of the path. The signal of the slope, for each edge between nodes i and j, is then computed as:

$$s_{\text{ij}} = \Delta_{\text{dist}_{\text{ij}}} / \mid \Delta_{\text{elev}_{\text{ij}}} \mid \tag{4.3}$$

where $\Delta_{\text{dist}_{\text{ij}}}$ is the distance expressed in meters between the cells where the two nodes of the graph are located, while $\mid \Delta_{\text{elev}_{\text{ij}}} \mid$ is the difference of elevation between the two cells in absolute value.

For the distance we had to consider that the Earth's surface is not flat and we cannot compute the distance like in an Euclidean plane. We used the *haversine* formula to calculate the great-circle distance between two points on a sphere given their longitudes and latitudes, i.e. the shortest distance over the Earth's surface (without considering any hill). First, we considered the location of the cells inside the grid relative to the tile's global position expressed as latitude and longitude, i.e. we calculated the specific latitude and longitude coordinate of each cell based on their position inside the grid. Then, the haversine formula is used to compute the distance as follows:

$$\Delta_{\text{dist}_{\text{ij}}} = R \cdot c \tag{4.4}$$

where $R = 6,371m$ is the Earth's radius, while c is computed as:

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)}) \tag{4.5}$$

with $atan2(x, y)$ defined as the angle in the Euclidean plane, given in radians, between the positive x-axis and the ray to the point $(x, y) \neq (0, 0)$. $atan2(x, y)$ is intended to return a correct and unambiguous value for the angle θ in converting from cartesian coordinates (x, y) to polar coordinates

(r, θ). It is defined as:

$$atan2(y, x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \wedge y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \wedge y < 0 \\ +\frac{\pi}{2} & \text{if } x = 0 \wedge y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \wedge y < 0 \\ \text{undefined} & \text{if } x = 0 \wedge y = 0 \end{cases} \tag{4.6}$$

Finally, a is the *Haversine* computed as follows:

$$a = sin^2(\Delta\phi/2) + cos(\phi_i) \cdot cos(\phi_j) \cdot sin^2(\Delta\lambda/2) \tag{4.7}$$

where ϕ_i is latitude for node i, while λ_i is the longitude.

Signals on the nodes

Other methods for Deep Learning on graphs, like *GraphSAGE* [14], during the learning process use a set of signals representing the properties of each node. Among the possible features of the nodes we considered both the topological properties of the graph, like the degree of the node or the type of critical point represented, and the terrain properties, like elevation or slope. In what follows consider that, although for the saddles we have a recurrent topology in the number of critical points to which they are connected, for the maxima and the minima it is not possible to replicate any structure, indeed they have a really large range of possible numbers of neighbours. Consider also that with *GraphSAGE* the edges represent just the existence of a relation between two nodes, i. e. that they are connected, and it is not possible to use signals on the edges. *GraphSAGE* also requires that for each node we are using a vector of signals of fixed length, meaning that we are using the same features for each node. Then, for features like the slope which involve a relation between two nodes, if we would use all the signals

that are possible to construct for that feature between the considered node and all of its neighbours we would have also really different sizes for the sets representing the signals. For being able to keep fixed the length of the vector containing the signals for the nodes, for features for which it is possible to calculate a variable number of signals depending on the number of neighbours we considered uniformly only the average, the maximum and the minimum of the set of signals for that feature. For example, for the slope, if a node has ten neighbours, first we computed all the ten possible slopes and then we calculated the average, the maximum and the minimum values as the signals representing the node's slope. The features we identified for building the nodes' signals are then the following:

- *elevation:* a real value expressed in meters representing the altitude of the cell inside the DEM where is located the node.
- *degree:* a natural number consisting in the number of neighbours to which the node is connected.
- *critical point type:* categorical attribute representing which kind of critical point there is at the location of the node. The possible values are {peak,saddle,pit}.
- *slope:* as we stated before the slope is computed based on the neighbours. We compute all the slopes with all the connected nodes by using the Formula 4.3 but without the absolute value when computing the difference of elevation:

$$s_{ij} = \Delta_{dist_{ij}} / \Delta_{elev_{ij}} \tag{4.8}$$

  between the node i and each of the connected nodes j and then the following variants:

  - *average slope:* the average value of the set of all the slopes
  - *maximum slope:* the maximum slope in the set

  - *minimum slope:* the minimum value in the set

- *absolute slope:* we compute all the slopes with all the connected nodes by using the Formula 4.3 and then its derived features:
  - *average absolute slope:* the average value of the set of all the slopes
  - *maximum absolute slope:* the maximum slope in the set
  - *minimum absolute slope:* the minimum value in the set

- *distance from neighbours:* this is another feature that depends on how many neighbours a node has. We compute all the distances by using the Formula 4.4 and then its derived features:
  - *maximum distance from neighbors:* the maximum distance in the set
  - *minimum distance from neighbors:* the minimum distance in the set
  - *average distance from neighbors:* the average value of the set of all the distances

- *elevation drop*: this feature is strictly related with the terrain around the location of evaluated node. It quantifies how much difference of elevation there is between the altitude of a considered node and the neighbor cells around it. If for example we consider a peak, we are expecting it to be the highest cell withing the grid of 3x3 cells centered in the location of the node. But it can also be important to understand the magnitude of this difference with the neighbors because mountains can have different shapes regarding the summit. There are mountains whose summit is really evident and is rising among the neighborhood while others have a more flat behaviour and can extend over different cells (more like hills). This feature is then measured as the difference of elevation between the altitude at the location of the node and the altitudes of the adjacent cells.

To account to possible mountains with a bigger summit extension we considered also the elevation drop between the altitude of the cell of the considered node and farther adjacent cells. In a grid of 3x3 when we consider the drops we are considering cells at a distance of 1, i.e. we move away of 1 cell from the location of the considered node. For DEM STRM3 we considered also the difference with cells at distance 2 and 3 away from the center of the grid, i.e. the border cells respectively in the a 5x5 grid and 7x7 grid. By considering that at the latitude of Switzerland one cell of STRM3 covers a distance between 70-90 meters, this means that we included an area of 200m-250m. This value (200m) is important also in the graph labeling step as we will see in Section 4.4.2 and accounts to the fact that the real location of a mountain can differ from the ones available in the public databases.

For DEM STRM1 to mantain the same distances and to keep fixed the number of features involved we had to use different distances from the center of the grid. Indeed, for STRM1 1 cell covers around 27m-30m, which means that if we would consider only cells at distance 1,2 and 3 we wouldn't cover an area more than 90m. We decided then to use hops of 2, 5 and 8 cell distances from the cell where the involved node is located, covering approximately the same area of 200m-250m of distance from the center cell. An example of how these areas overlap and which areas are taken into account for the calculation can be seen in figure 4.7.

Finally, we have to consider that if we use as features for our models all the elevation drops, first of all the number is really high, second, and most important, this feature will be subjective to the relative location of the node's neighbor with which have been computed the difference. This would be a bias if, for example, during the training would happen that for the peaks all the neighbors located in the top-left cell have a similar elevation with the node's cell. Then, if in the test set there would be some peak which is not having a neighbour in the top-left cell with a similar elevation it would have less probabilities to be correctly

identified. We need to have *order invariant* features to be able to generalize as much as possible. To achieve this important property and to not have a huge number of features, as happened for the other features, we are keeping the *average, minimum and maximum* values for each level. This means that we compute the elevation drop for all the cells ad distance 1 and then calculate the average, minimum and maximum values as final signals to be used on the graph. We did this for *3 levels* both for STRM1 and STRM3 as it can be seen in figure 4.7.

4.4 Graph Labeling

Deep Learning and in general Machine Learning models need trusted example data to learn from. Learning mountains from a graph can be achieved in different ways. In this work we tried to learn which nodes of the graph correspond to the locations of the peaks of mountains. Note that here "peak of a mountain" is not the same of a "peak" of a surface network, even though they may coincide. Indeed in surface networks peaks are synonyms of maxima (always local, except of the case of global maxima that is the highest summit of the Earth, mount Everest). Instead, "peak of a mountain" or "summit of a mountain" are referred to the mountains that are registered in cartography or that are traditionally recognized by society. This means that not all the maxima of the Earth's surface have been recognized as mountains. A local maxima that is almost flat or whose elevation is really low, for example a hill with really low slope, is often not registered neither in maps or public databases because society did not recognize them as mountains and didn't give a name to those locations. Also it is not even guaranteed that public databases and maps have the precise location for all the mountains. It depends often how historically they have been built and the social value that had those locations. Often these public data sets are mostly built by volunteers and cannot be assumed to be 100% complete. Considering these aspects that influence where mountains are traditionally located we can state that

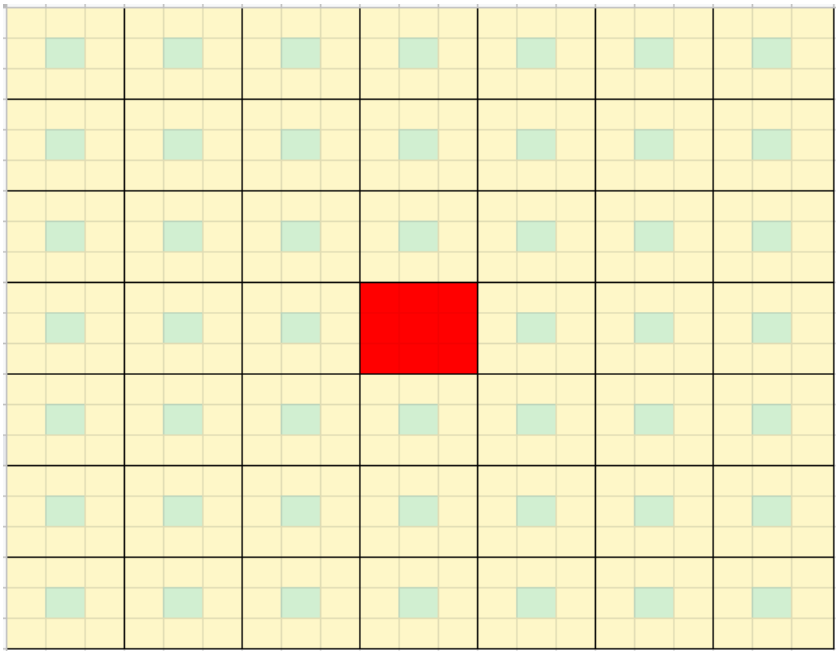

Figure 4.7: Cells involved in elevation drop
Illustration showing how STRM1 and STRM3 DEMs can be overlapped when computing the elevation drop. In red we have the cell representing the location where a critical point is supposed to have been found with the techniques we have seen in Section 4.3.1. Then, for STRM3, we consider the cells in yellow with bold border. In the figure we can see 3 levels of distance from the center, leading to 3x3, 5x5 and 7x7 grids. The drop is computed between the red cell and the adjacent ones when dealing with 1 level of distance; instead, with distance 2, for example, we consider the border cells of the 5x5 grid and we calculate the difference of elevation with the red cell. For STRM1 we can see that we can cover the same areas by considering that the data is more resoluted and that we can approximate the STRM1 location with the center of the STRM3 location shown in green, i.e. one cell in STRM3 corresponds to 9 cells (3x3) in STRM1. By doing this we keep the number of computed features fixed and we are also covering roughly the same area, leading to a more consistent set of features when switching between the graphs built over different DEMs.

not all the locations where there is a maxima of the graph are "real peaks". Often, the locations of the "real mountains" are tens of meters far from some node of the graph that corresponds to a maxima. More rarely, the location of a node is few meters apart of an area registered as "peak" in one of our datasources, and really few times they were coincident. We have then that the nodes locations do not match to the areas of the peaks registered in the data in our posses and we need to find a way to create a correspondence between the locations given by the different sources.

4.4.1 Ground truth

We said initially that we need trusted examples from which to learn. For our specific task we need then a set of "real peaks" or "trusted peaks" having a position that is known with acceptable certainty. These examples are said to constitute the *Ground Truth*. They are not used only for training the machine learning models but also for making a comparison between which peaks the models we trained are able to extract and the ones extracted by other algorithms from literature. However, the reasons which makes impossible to have an exact overlap between the locations of the "real peaks" and the peaks of the surface network also makes impossible to have an ideal gold standard. Then we should account the fact that there is noise in the ground truth when we will evaluate the model output. It may happen for example that the model predicts as being a peak the node for a given location where it is not registered any "real summit", a so called false positive. However we may consider the fact that the peak was missing from the ground truth. Implementation of techniques for coping with label noise like [68] are not part of this work but it may be for future developments.

The data sets we used for our ground truth refer to the Switzerland territory. We used two different publicly available databases: OpenStreetMap[1] (OSM) and SwissNames3D[2]. We merged the peaks of the two databases by considering that their locations were overlapping when their distance was

[1]https://www.openstreetmap.org

[2]https://shop.swisstopo.admin.ch/en/products/landscape/names3D

Figure 4.8: Mountain peaks distribution
(a) contains the peaks from SwissNames3D, (b) peaks from OpenStreetMap and (c) their combination in the Switzerland territory. Image courtesy [41]

lower than 80m. We considered them potentially the same mountain summit and kept only one of the two after a manual inspection of the correctness of the choice. The resulting ground truth data set contains 12,788 peaks. Their distribution can be seen in Figure 4.8.

4.4.2 Label assignement to nodes

After deciding which areas of the world to use as the ground truth we had to decide how to match the nodes of the surface network with the locations of the "real peaks" from the ground truth. Considering the results achieved in [41] we decided to keep the same maximum distance of 200m for the matching between the locations of the elements of the ground truth and the nodes of the surface network. Indeed, for each "real peak" in the ground truth, if there is a node whose category is a maximum and whose distance from the location in the ground truth is less than 200m then to that node is assigned a label "1" meaning that in our data that example will correspond to a peak. Here we are talking specifically about maxima nodes because it may happen that for an area in the ground truth the closest node found may be a pit or a saddle. In that case we ignored those points and continued increasing the distance since a maxima would be found or the maximum distance of 200m reached.

4.5 Development of Graph learning-based methods

The goal of this work is to explore the application of Graph Deep Learning techniques for identifying the location of mountain summits. The previous work [7] developed a Deep Learning method which leverages Convolutional Neural Networks fed with small grids (patches) extracted from DEMs. CNNs are really powerful models which achieved great results for pattern recognition and image classification. As the authors showed, the patches extracted from DEMs can be considered as images where the cells containing elevations can be treated as pixels whose values represent the intensity on a given channel, allowing then to exploit the CNN and their capabilities. This work, instead, focused more on treating DEMs as digital (non-continuous) surfaces over which locate specific points, called *critical points*, and their connections. It is therefore possible to build a graph representing these particular spots and their links where the nodes are the critical points and the edges represent the relationship existing between them.

At the time we started this work, numerous works arose exploiting Deep Learning and Artificial Neural Networks on graphs (ref. 2.2.4). We tried to cover the following approaches alone or combined for learning to classify the nodes of the surface networks extracted from the DEMs.

4.5.1 Logistic Regression

Logistic Regression (ref. Section 3.2.1) is a classical Machine Learning model which works on euclidean data, i.e. a setting where for every observation of a phenomenon there exists a record which can be represented as a vector with numerical (but also categorical) values. Considering our case, we used the Logistic Regression model with the vectors of signals representing the nodes properties of the surface network built over the DEMs, i.e. each node of the graph represents a record constituted by its features. These features are referred as "handcrafted" in the sense that have been built following the process in section 4.3.5. Furthermore, from the handcrafted features there

have been selected the most meaningful ones according to the procedure in section 4.7. Finally, after building the graph and computing the signals for the critical points we have a set of nodes with their relative features (elevation, slope, degree, drop, node type) and binary labels (peak/non peak). While this is a suitable scenario for a supervised binary classification task, the graph topology is not used directly due to it's relational structure which is hard to fit into an euclidean space. It can be, however, interpreted as a useful mean for finding the locations of the candidate nodes and for calculating their signals. We can consider these experiments also as a strong baseline to be used as comparison for the more advanced Deep Learning models.

As we will see in the next section, another kind of features which can be used as input are the ones learned with Node2vec and GraphSAGE. This setting is supervised where for each vector of signals (or learned embeddings) there is also a label given by matching the nodes with the ground truth peaks, as described in section 4.4.2. The graph labels are then assigned to the embeddings of the nodes and the dataset splitted into training, validation and test accordingly to the division we will see in Section 4.6. In the setting of a binary classification the output of the model is then a probability value for each node of belonging or not to a given class, which in our case can be thought as a "peakness" value.

Hyper-parameters

For logistic regression we used the *sklearn python* library [69] which allows to run a Logistic Regression model with a series of parameters to be customized, such as, for example, the used loss or the weights of the classes. Below we describe each one of the parameters:

- *Class Weight*: a numeric value used during training phase to give more importance to the error made in misclassifying a class instead of another. Giving weight 2 to class 1, for example, means multipying by 2 the loss value given by wrongly classying an element of class 1 to belong to class 0. This means that the total loss would be higher if many records belonging to class 1 are wrongly predicted as belonging to class

0. This parameter con be really useful when considering imbalanced datasets as ours. The values we tried were [0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.25, 1.5, 1.75, 2, 5].

- *Regularization* is a very important technique in machine learning to prevent overfitting. Mathematically speaking, it adds a regularization term in order to prevent the coefficients to fit too much the data such that to overfit. There are two techniques of *regularization* we adopted, and they are refered as L1-norm and L2-norm (L1 and L2 abbreviated). As we will see in Section 4.7, the key distinction between these techniques is that L1 shrinks the less important feature's coefficient to zero thus, removing some features. With L2, instead, the coefficients can be really close to 0, but never equal, thus all the features are always part of the model. We tested then these two configuration parameters in combination with different sets of features. L1 altogether with the set of all the handcrafted features for letting the model to chose and L2 with a selected set of features previously with the method of Greedy Forward Search reported in Section 4.7.

4.5.2 Node2Vec

As we saw in Section 3.2.2, *Node2vec* is an unsupervised deep learning model which learns to generate embeddings for the nodes of a graph. Its input is a representation of the graph given by a set of tuples expression of the edges, each connecting two nodes. Furthermore, Node2vec can be fed also with a set of weighted edges, with a real positive value representing the weight of that edge. In case of both negative and positive weights the edges are considered to be directed. The input is then only the structure of the graph, i.e. only the edges describing the existence of a relationship between two nodes. It is an unsupervised setting because there is no classified data used for training the model.

As we saw in section 3.2.2 the main concern with Node2vec is creating representations for the nodes in the form of embeddings such that they are representative of the neighborhood of each node in the graph. This means

that based on the *structure* given by the graph, it is possible to create embeddings which are describing the role and/or the position of the node in the graph. These embeddings do not represent yet a prediction for the nodes of being or not a peak; indeed, as we can see in Figure 4.9, Node2vec constitute a step of *representation learning* as part of a bigger pipeline of models. For being able to achieve our main goal, i.e. to classify correctly the nodes of the graph as peak or not-peak, a classifier, such as *Logistic Regression*, needs to be fed and trained with the embeddings created with *node2vec*. We will see more specifically in the next section how logistic regression can be used in sequence with node2vec.

Hyper-parameters

As most of the Deep Learning models also Node2vec is subject to a flexible architecture which needs to be tuned through the choice of its hyper parameters. They can be described as following:

- p, also called *return parameter*, controls the likelihood of immediately revisiting a node in the walk;
- q allows the search to differentiate between "inward" and "outward" nodes, i.e. it controls the probability of moving farther from the source node;
- d dimension of the embedding, i.e. number of feature representations to learn;
- k context size, i.e. number of nodes considered when the emebdding is built;
- l it is length of of the random walk based on which the exploration is stopped after a number of l nodes are visited;
- r it sets the number of random walks to be done for each node.

In the Appendix we can see the grid of tested values. The optimal combination of the found parameters is:

The selection of the features is reported Appendix B. The best choice resulted in using the selected features with Greedy Forward Search and L2 regularization with 0.7 as class weight for the positive one.

4.5.3 GraphSAGE

This is a Deep Learning model which leverages both the graph topology and the node features to learn to classify the nodes of a graph. Its input are graphs, such as the surface networks generated from the DEMs; they are represented by means of a set of edges connecting the nodes and a set of vectors of signals generated from the surface properties. The features are selected according to the procedures in sections 4.3.5 and 4.7.

GraphSAGE combines, both the representation learning techniques, like *Node2vec*, and the classification ability, such as logistic regression. Indeed, for making the final prediction, GraphSAGE creates an embedding per each node of the graph by including the information from their neighborhood. The intuition behind this method is that at each step, nodes aggregate information of their local neighbourhood (topology and node features if available) and as the process moves forward, nodes acquire information of farther away nodes. At each iteration of the encoder algorithm, nodes aggregate the embeddings of their neighbors (using a learnable aggregation function that operates over sets of vector) and the aggregated neighborhood vector is combined with the node's embedding from the previous iteration. The final outcome is reached similarly as with *logistic regression*: the nodes are labeled by matching the peaks of the Ground Truth (refer to chapter 4.4.2) and the model learns to predict the grade of "peakness" of the nodes by giving as output a numerical value between 0 and 1. *GraphSAGE* uses these labels to optimize the model tuning such that it will learn to classify unseen nodes also over unseen graphs which will have similar features and similar topology. The model learns to create representative embeddings, regarding both the features and the topology of the edges, which are then used as input for a fully connected network which will output the probability of the node of being classified as peak or not-peak. The model is then tuned to reduce at the maximum the predicition

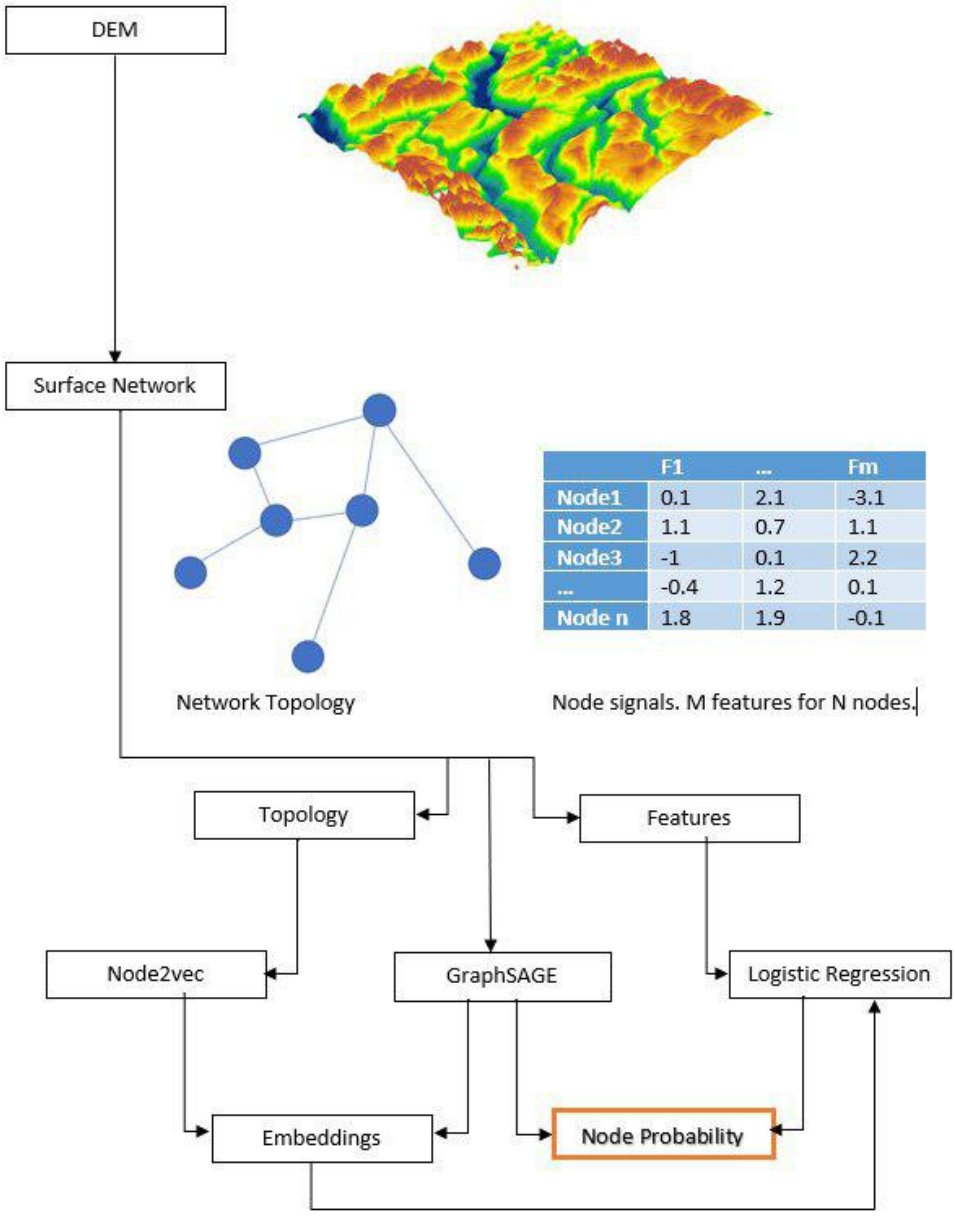

Figure 4.9: Pipeline of the models

error.

Considering that *graphSAGE* is able to generate embeddings in its intermediate steps, we also extracted the learned ones from the final stages of the trained model and used as input for the *Logistic Regression* model as we did with *node2vec*. Again the output is a set of numerical values between 0 and 1 representing the probability of the node of being or not classified as peak. The pipeline from the input data to the final outcome can be seen in Figure 4.9.

Hyperparameters

Graphsage has a complex architecture and various parameters. For finding the optima configuration we performed a Grid Search for the best combination. The choice of the values to be tested in the Grid Search have been explored through a *Parameter Sensitivity Analysis* which values are reported in the Appendix A.1. It essentially consists in keeping the architecture fixed with its base hyperparameters setting except for one which is evaluated with values different from the standard configuration. Depending on how much its variation affects the final performance of the model its values are explored more exhaustively (if it has a big impact) or kept close to the base one (if its change do not affect much). The *core parameters* for the model are reported as follows.

- *Aggregator*: it decides how to create the embedding for the nodes by aggregating in different ways the current node features with the ones of its sampled neighbors. It can be a mean pool (element-wise), max pool, mean (of the vector) and LSTM.

- *Embeddings dimension*: parameter which influences the complexity of the network and sets the number of real values which we want the final embedding to be composed of.

- *Sampling*: the number neighbours to be sampled at each iteration of the learning process.

- K: number of layers to explore. K=1 means only the immediate neighbors, i.e. the connected nodes, K=2 also the neighbors of the neighbors, and so on for increasing K. We left if K=2 as it was for the original model.

Other typical parameters of Deep Learning models consist in the choice of:

- *learning rate* (how fast the model learns);
- *dropout* (leave out randomly some parts of the model during the training phase to avoid overfitting);
- *batch* (number of samples to consider during each iteration of the training phase);
- *weight decay* (parameter for L2 regularization);

We modified the architecture and introduced also a *class weight* to cope with the high imbalance. In table A.1 we can see values used for the sensitivity analysis while in table A.2 the final grid search, where in blank we have the found optimal configuration which can be resumed with the following values: class weight = 0.7, aggregator = LSTM, number of layers = 2 (it was the default), embedding layer 2 = 32, embedding layer 1 = 512, sampling layer 2 = 5, sampling layer 1 = 5, learning rate = 0.001, dropout = 0, weight decay = 0, batch = 512. Also, the best result is achieved with the entire set of features.

4.6 Dataset

We partitioned the Switzerland territory in three distinct regions: training, validation and testing. We can see in Figure 4.10 how these areas are distributed. The choice of the validation and testing cells is not random; it has been made based on the Ground Truth described in Section 4.4.2 to have the GT peaks distributed so that 80% of the peaks belong to the training and validation areas and 20% to the testing area. The same dataset is used for all

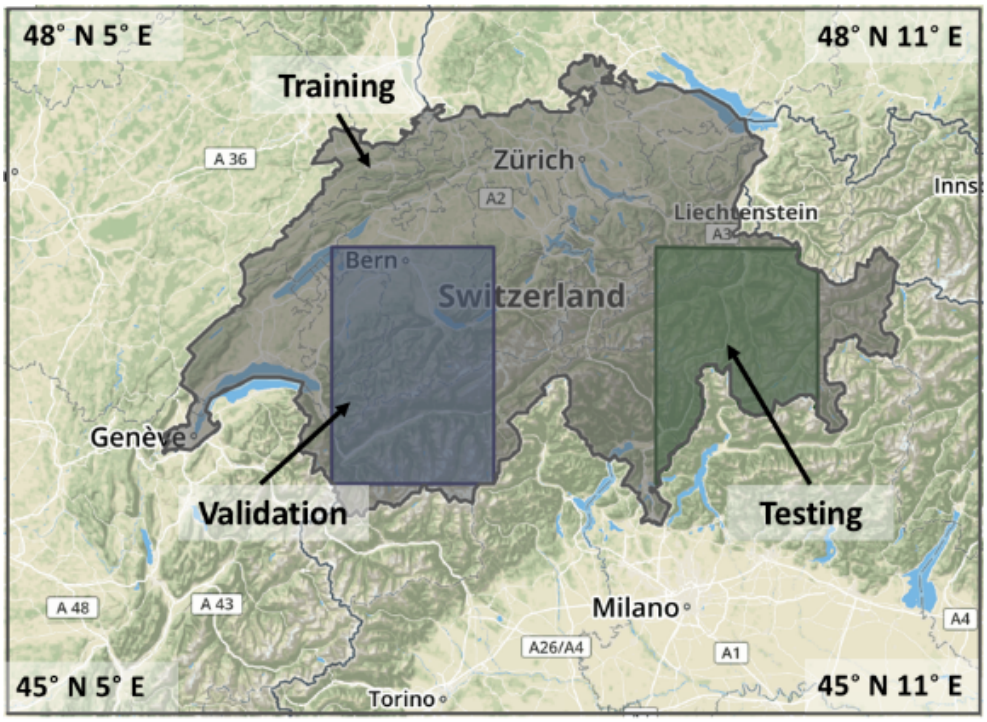

Figure 4.10: Territory distribution of the datasets
Image courtesy [41]

DEM	**#maxima**	**#minima**	**#saddles**	**#edges**	**#matched GT**
DEM3	56986	46313	260704	1075200	5367
DEM1	157943	145456	999311	4176641	6462

Table 4.1: Switzerland graph components

the presented architectures; for each DEM cell, both STRM3 and STRM1, first the critical points are extracted, then the connections between them are indivudated. When the graph is ready the signals (elevation, slope, degree, drop, node type) for the nodes are computed and then labels (peak/not-peak) assigned. The graph is then splitted in three parts, depending on the coordinates of the nodes, matching the three areas highlighted in Figure 4.10. Considering the technique we saw in Section 4.4.2 we could partially match the ground truth with the graph structure. Indeed, in the Switzerland territory we could identify the existence of 7223 summits. Regarding STRM3, it was possible match 5367 peaks of the ground truth to nodes of the graph build over the DEM3, while the graph over STRM1 has 6462 of these peaks matched to the nodes. This means that there are some regions were it wasn't possible to find any critical point, specifically local maxima, that could match the location of the peaks in the ground truth. The overall number of nodes, their type distribution, number of edges and number of matched peaks for the Switzerland area can be seen in Table 4.1.

We also studied the ability of the methods to generalize over unseen areas. Apart the test set which is still part of Switzerland where the mountains are expected to have similar features since they belong to Alps, we wanted to understand if summits belonging to mountain ranges can be identified with the model trained over Switzerland. We used as ground truth the peaks provided by OSM. Given that OSM is a VGIS based on the spontaneous collaboration of its users, not all the areas have mountain data as complete as the Switzerland territory, which may lead to a biased generalization test. We decided then to analyze only territories which have OSM mountain data with a coverage similar to that of Switzerland.

4.7 Feature Characterization

We described in Section 4.3.5 how to enrich the graphs with signals describing their properties. *Node2vec* is using only the topology so there are no handcrafted features that need to be selected. Instead, with *GraphSAGE* and *Logistic Regression* we tried to understand which of these features are having more impact on the final result and if there is the need of reducing their number.

Since the handcrafted features are almost all derived from the elevation and the distance between nodes it doesn't seem strange that they are related. More specifically the *Pearson correlation coefficient* [70] is a statistical test for measuring relationship, or association, between two continuous variables. It is one of the most known methods for measuring the association between the variables of interest, giving information about the magnitude of the association, or correlation, as well as the direction of the relationship. As we can see in figure 4.11, the correlation matrix shows the existence of a relevant correlation among the group of the *drop* features, and also a discrete correlation among the ones of the slope. Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. It seems normal for example that the "drop" at level 1 is related with the "drop" at level 2, as a continuation of the surface. Also, some Machine Learning methods may be affected on the final performance from redundant features, hence is common practice to remove the unnecessary ones.

There exist various methods in Machine Learning for selecting the best features to be used in a model. A classical approach is based on the Pearson correlation matrix we saw in Figure 4.11 and consists in keeping only one of two features that are correlated. Another approach is using a regularizer inside the model itself which assisgns a coefficient of importance to the features. The model then shrinks these coefficients for the unnecessary features close or equal to 0. Two really common regularizers in Machine Learning are L1-norm and L2-norm; the first one can lead the coefficients for the less important features to be 0, i.e. to not be considered, while with the second

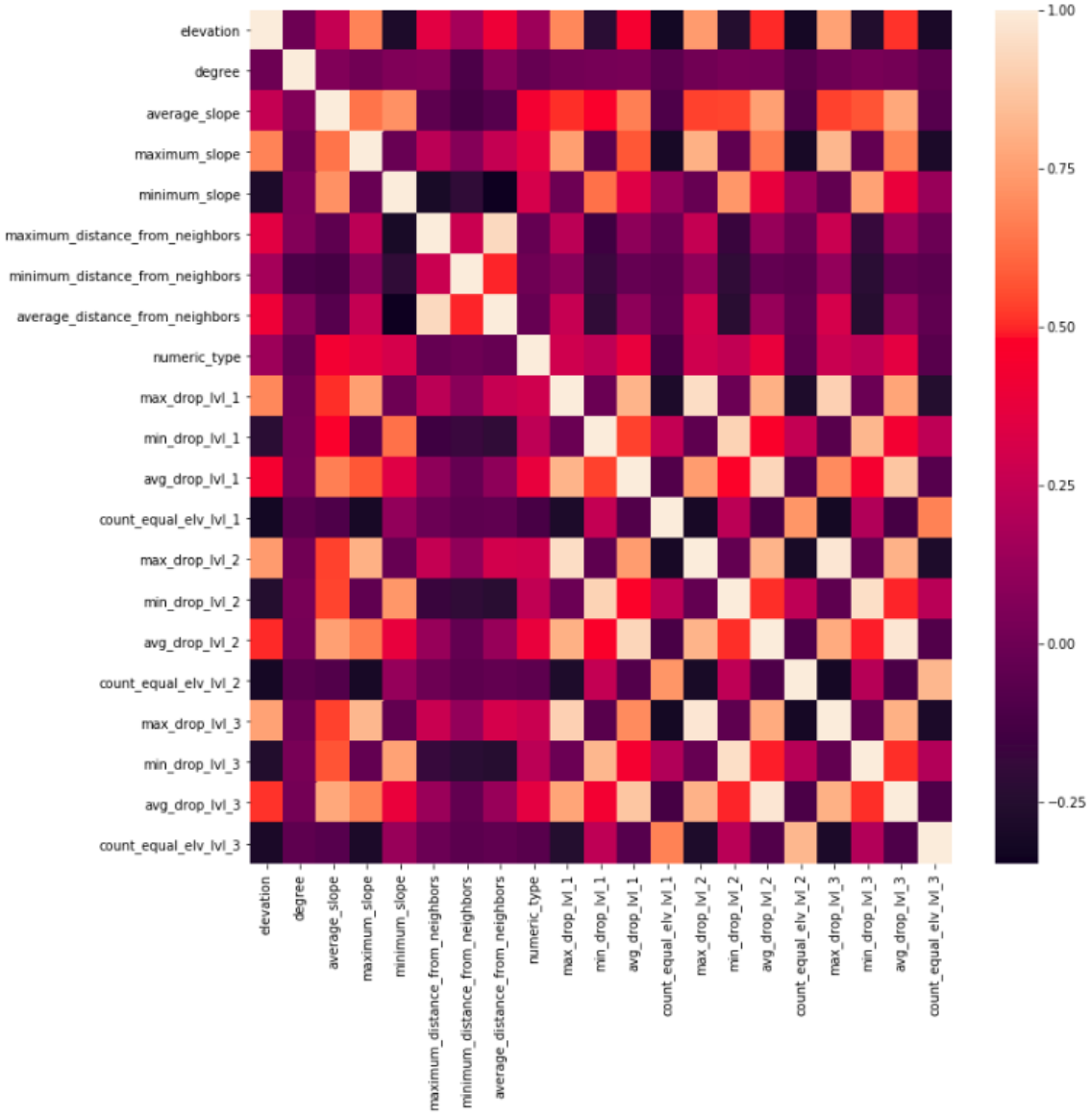

Figure 4.11: Correlation matrix of the handcrafted features

they can only get close to 0 but are never totally excluded. These techniques have been proved to work well with data laying on an Euclidean space, i.e. a setting where for each record we have a given number of variables, often called *regressors* or *attributes*, and a label representing the outcome. In our setting, despite the fact that we can consider the nodes as records where the signal represent the values of the regressors and the node label is the outcome, we have other several *dimensions* given by the topology of the graph which cannot be stored as a numerical or categorical feature due to the multiple relationships between the nodes. For the baseline experiments with Logistic Regression we could then adopt the L1-norm or L2-norm regularizers as part of the model; while for coping with the topology dimensions exploited in GraphSAGE, we adopted another method, called *Greedy Forward Search*, which essentially allows to incrementally add features to the model until a decay of performance is observed (Algorithm 1).

The measure used for understanding the goodness of the model has been the *F1 Score* which is widely used for models where the data is highly imbalanced towards a class. Indeed, as we saw in Table 4.1, for the graph built with DEMs from STRM3 we have 5367 positive labels over 364003 nodes and for STRM1 this imbalance is even higher: 6462 over 1302710 nodes. For computing the *F1 Score* we first have to calculate the so called *True Positives - TP*, representing the correctly predicted nodes as being peaks, the *False Positives - FP*, counting for the wrongly classified nodes as peaks while they were labeled as non peaks and the *False Negatives - FN*, the locations where there exists a peak but it is classified as non peak. The *F1-measure* can be then expressed as:

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}, \tag{4.9}$$

where *precision* and *recall* are defined as follows:

$$precision = \frac{TP}{TP + FP} \tag{4.10}$$

$$recall = \frac{TP}{TP + FN}. \tag{4.11}$$

Algorithm 1 Features Greedy Forward Search

$\mathcal{F} \leftarrow \emptyset$, where $\mathcal{F}$ is the set of chosed features.
$\mathcal{A} \leftarrow \{f_1, f_2, .., f_n\}$, with $\mathcal{A}$ representing the set of all the features.
$F_1 \leftarrow 0$, measure representing the goodness of the set $\mathcal{F}$
$best\_F_1 \leftarrow 0$
while $F_1 >= best\_F_1$ **do**
    $best\_F_1 \leftarrow F_1$
    $F_1 \leftarrow 0$
    $best_index \leftarrow 0$
    for $i \leftarrow 1$ to $length(\mathcal{A})$ **do**
        $f_i \leftarrow getElementAtPosition(\mathcal{A}, i)$
        $current\_F_1 \leftarrow computeF1(\mathcal{F} \cup f_i)$
        if $current\_F_1 > F_1$ **then**
            $F_1 \leftarrow current\_F_1$
            $best\_index \leftarrow i$
    if $F_1 > best\_F_1$ **then**
        $\mathcal{F} \leftarrow \mathcal{F} \cup getElementAtPosition(\mathcal{A}, best\_index)$
        $\mathcal{A} \leftarrow removeElementAtPosition(\mathcal{A}, best\_index)$

Following the Algorithm 1 and based on the curves in the Appendix B we found that the best subsets of features for GraphSAGE and Logistic regression are:

- **GraphSAGE** best features (refer to Figure B.1): maximum drop level 3, elevation, degree, average drop level 3, average drop level 1, minimum drop level 1, average slope, average distance from neighbors, maximum slope, count equal elevation level 2, numeric type, minimum drop level 2, minimum drop level 2, average drop level 2, maximum drop level 2, minimum slope, minimum drop level 3, count equal elevation level 1 and maximum distance from neighbors.

- **Logistic Regression** best features (refer to Figure B.2): minimum drop level 3, average drop level 2, degree, elevation, average distance from neighbors, numeri type, average drop level 3, average slope, maximum slope, minimum slope, minimum distance from neighbors, max drop level 2 and average drop level 1.

4.8 Evaluation procedure

The common output for the experimented methods is a probability for the candidate nodes for being a peak or not. Their location corresponds to a geographic position (latitude, longitude), which may correspond or not to mountain peak; these coordinate have been also used to filter out all the extracted peaks that are out of the area under evaluation. To determine whether an extracted peak from a node corresponds to a ground truth peak, we use a distance threshold (200m, in the evaluation described in Chapter 5). The steps for the comparison, whose pseudocode is given in Algorithm 2, are as follows:

- Calculate the distance between each *extracted peak* and every *ground truth peak*.
- For each pair, save the tuple *(extracted peak, ground truth peak, distance)*, only if the distance is lower than the established *threshold*.
- Order all tuples by increasing distance.
- Loop through the ordered list of tuples. Consider the extracted peak of the current tuple as a *True Positive* only if both the extracted and ground truth peaks have not already been used before to define other *True positive* peaks; otherwise, discard the current tuple.
- The *extracted peaks* that have a distance to all the ground truth peaks greater than the threshold are considered as *False Positives*. Also, the extracted peaks that appear in a saved tuple, but have not been selected

as true positive ones, because they were dominated by the extracted peaks in some other tuple, are classified as *False Positives*.

- The *ground truth peaks* for which no matching extracted peak has been identified are considered as *False Negatives*.

Algorithm 2 Post Processing

```
NEPeaks ← length(ExtractedPeaks)
for i ← 0 to NGT − 1 do
    GTP ← GTPeaks_i
    for j ← 0 to NEPeaks − 1 do
        EP ← ExtractedPeaks_j
        Distance ← calculateDistance(GTP, EP)
        if Distance < 200 then
            Distances_end ← (EP, GTP, Distance)
TruePositives ← ∅
Distances ← sorted(Distances)
NDistances ← length(Distances)
for i ← 0 to NDistances − 1 do
    PeakTuple ← Distances_i
    GTP ← PeakTuple_GTP
    EP ← PeakTuple_EP
    if GTP in GTPeaks and EP in ExtractedPeaks then
        TruePositives_end ← PeakTuple
        remove(GTPeaks, GTP)
        remove(ExtractedPeaks, EP)
FalsePositives ← ExtractedPeaks
FalseNegatives ← GTPeaks =0
```

In summary:

- *True Positives* are the extracted peaks that have a correspondence to a ground truth peak

- *False Positives* are the extracted peaks that have no correspondence with a ground truth peak
- *False Negatives* are ground truth peaks that the method was not able to match to any extracted peak

True negatives, i.e. locations that do correspond to non-peak sites, are more challenging to define because the number of potential candidates is much bigger to those of the other types: the input graph from DEM3 has around 360,000 nodes, from which only 5367 are local maxima labeled positively, meaning that only less than 1% are ground truth peaks. To cope with such an unbalance, we use the Precision-Recall curve in the assessment, rather than other methods such as the ROC curve, as suggested in [71] for scenarios with highly imbalanced classes.

To better quantify the accuracy of the tested methods, we considered the mean distance error of each algorithm. In this way, the assessment considers not only the presence of a match between a ground truth peak and an extracted peak, but also their distance (the lower, the better).

Chapter 5

Evaluation

5.1 Overview of the Evaluation

In this chapter, we perform a quantitative and qualitative analysis of the implemented DL and ML models and the studied heuristic methods. First of all, we present a comparison in terms of inputs, parameters and outputs of the involved methods, both heuristics and ML models and discuss their differences when extracting peaks and their applicability. After a post processing of he outputs, we evaluate the adopted DL models against the replicated heuristic methods and the baseline ML ones in terms of the Precision-Recall achieved with the method in Section 4.8 and analyze why one performs better than the other. The methods under evaluation are:

- Peak Classification,
- Landserf Surface Classification,
- Node2Vec,
- Logistic Regression,
- GraphSAGE.

The final qualitative analysis is useful to understand the different definitions of what is a peak of the various methods; furthermore, it underlines the

need to perform a further study on the False Negative peaks which present non-morphological features.

5.2 Parameter Selection

We performed the training process on the Switzerland territory, in particular we used the area with coordinates between latitude 46 to 47 and longitude 7 to 8 has been used as a validation set for the DL and ML models, as we demonstrate in Figure 4.10. We calculated precision and recall for Peak Classification and Landserf Surface Networks [18] using the Landserf tool [66] and the DL and ML models that we implemented. For all the methods we assessed the list of the extracted peaks, with the corresponding coordinates. We performed the evaluation as explained in Chapter 4, using a a 200 meters distance treshold to determine if the candidate peaks correspond to a peak listed in the ground truth dataset. We selected the treshold using as reference the previous works and choosing the less restrictive value; in [3] the authors applied a minimum distance between two mountains using a 150m window; in [23], the author employed a horizontal threshold to filter peaks and tested 150m and 200m as values. Furthermore, in the area under study, shown in Figure 4.10, the average distance between any two peaks in the ground truth data set results to be 44km, which is, as expected, much larger than the 200m threshold value used in the peak comparison metrics.

Each method executes with different parameters, which must be set heuristically. For each parameter, when the original specification of the method already provided an optimal or suggested value, we adopted it. Otherwise, we sampled the values from the parameter space and all the resulting parameter combinations (i.e., tuples of sampled values) are tested. For the DL and ML models, before we used them in this stage we performed a tuning based on the hyperparameters we provide in Section 4.5 in combination with the set of features we selected with the methods in Section 4.7. The set of values we selected for each algorithm is then as follows.

5.2.1 Heuristic methods

Here we present the selected heuristic methods of the Landserf Application which we use as comparison. We describe the input, the parameters and the output.

Peak Classification The method takes as input the DEM files and has two configurable parameters: the *minimum height*, in meters, that a point must possess to be considered as a candidate peak; values from 400m to 4500m with a step of 50m were employed for this parameter and the *minimum drop* in meters, that a point must have from a peak to be considered part of its extent; for this, we used values from 500m to 0m with a step of 5m. This yielded 8373 configurations. Its output is a matrix for each cell of the DEM saying if it is a peak or not and their extent.

Landserf Surface Classification As Peak Classification, also this method takes in input DEM files and works with 3 parameters: window size, the number of cells along one side of the square window centered on the currently analyzed point, with values ranging from 7 to 75, sampled with a step of 2; curvature tolerance, to choose how convex/concave a feature must be to be considered part of any class, with values corresponding to 0.1, 0.5, 2, 4, 6; and distance decay, to determine the importance of the cells near the center of the window with respect to those at the edges, with values ranging from 0 to 4 with a step of 1. The total number of combinations of parameters for this method was 875. The output is a Metric Surface Network from which are selected the peaks and their locations constitute the set of predicted summits.

The raw output for both the heuristic methods is post processed into a list of candidate peaks locations from the input area of the DEM.

5.2.2 Machine Learning

As we present in Section 4.5 the three methods we implemented work with different inputs but we can achieve a common output through the following steps.

Node2Vec is an unsupervised Graph Deep Learning model which produces a set of learned features for the nodes of a graph. It has in input the topology of the network expressed through the edges represented with tuples in the form <node1,node2>. The best architecture can be found in the Appendix A.2. The output is a set of embeddings of each of the nodes comparing in the tuples. The learned representations are then used as input for a classifier such as Logistic Regression.

Logistic Regression is a classical Machine Learning model which works on euclidean data. Its input are the vectors of features characterizing the nodes, which can be either the handcrafted ones or the learned ones. The best architecture uses 0.7 as class weight, with L2-normalization and the features selected in Section 4.7. The output is a probability for each node of belonging to a class. This allows to have a common evaluation method also for Node2Vec whose output can be used as input for Logistic Regression.

GraphSAGE is a Graph Deep Learning model which learns to classify a graph nodes. Its input are both the network structure and the handcrafted features of the nodes. The choice of the best architecture is performed using a Grid Search like presented in the Appendix A.1. The best features set was the one including the features. Its output is a set of probabilities for each vertex of the graph representing the membership to a class.

We have that for both For Logistic Regression and GraphSAGE the output is a probability for each node to belong to the "mountain peak" class. By filtering on this probability, we obtained a list of nodes which, since each node corresponds to a DEM cell, we could map to a set of latitude and longitude coordinates of candidate mountain peaks.

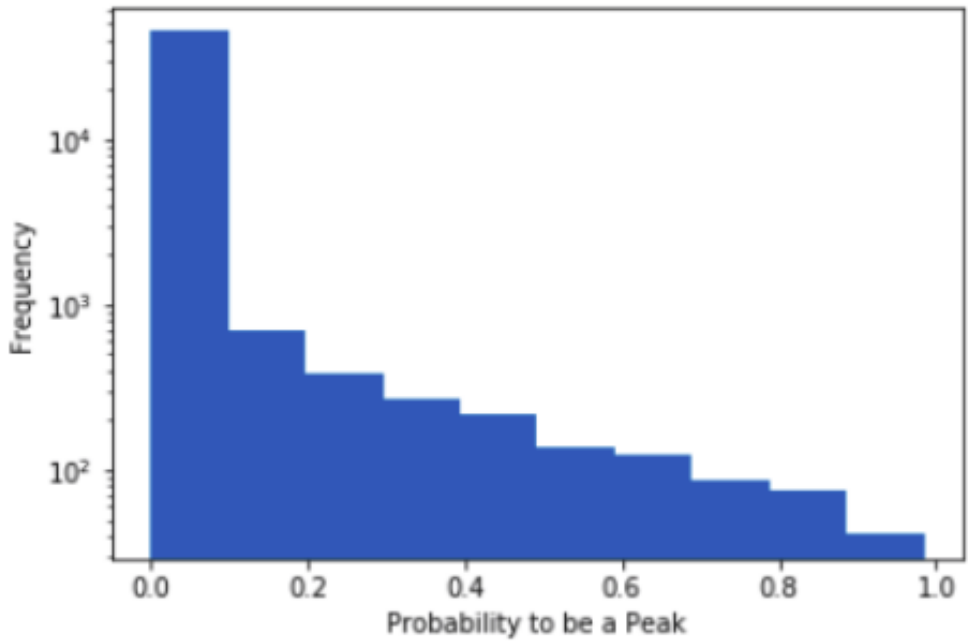

Figure 5.1: Logarithmic probability distribution assigned to each node of the graph by the Logistic Regression model with handcrafted features

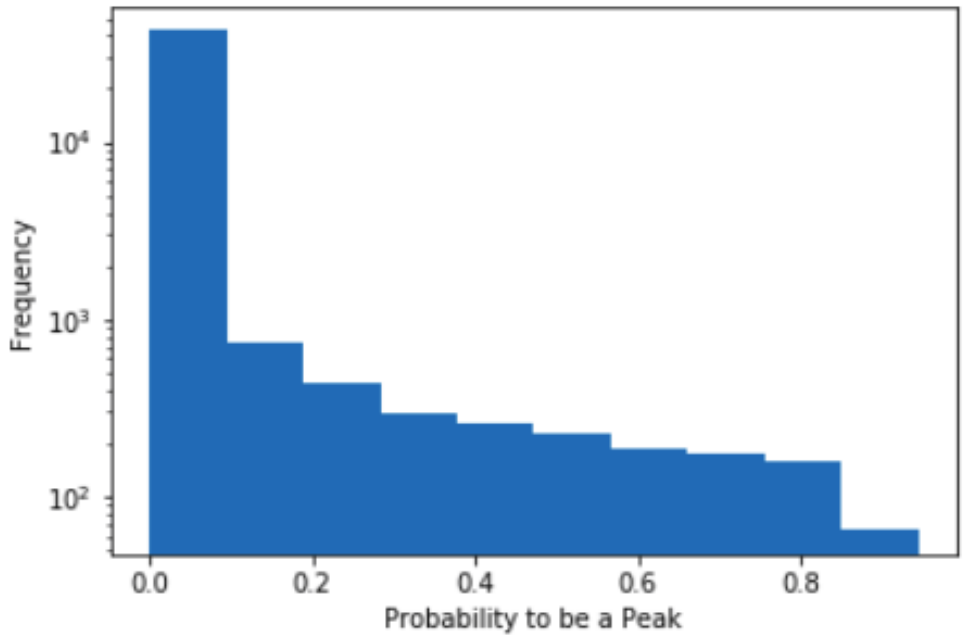

Figure 5.2: Logarithmic probability distribution assigned to each node of the graph by the GraphSAGE model

In Figures 5.1 and 5.2 we present a comparison between the distribution of the probabilities generated by the GraphSAGE and Logistic Regression with handcrafted features. We can observe that both had a high density in the range between 0 and 0.1, highlighting the fact that many nodes of the graph had low probability of being a peak. Indeed, from a graph of around 50.000 nodes for the validation area just around 1479 nodes are labeled as being peaks. As we move towards higher values, their frequency decreases showing less frequent predictions that that node was a peak. Even though the two models work with different data, their distributions are comparable and show a similar shape. Then, considering these probabilities we generated sequentially a range of 1000 tresholds of "peakness" and we applied them as a filter to the list of predictions. This yielded to 1000 configurations both for GraphSAGE and the Logistic Regression. These configurations provided a spectrum of results and enabled an analysis for understanding each algorithm behaviour of the considered territory.

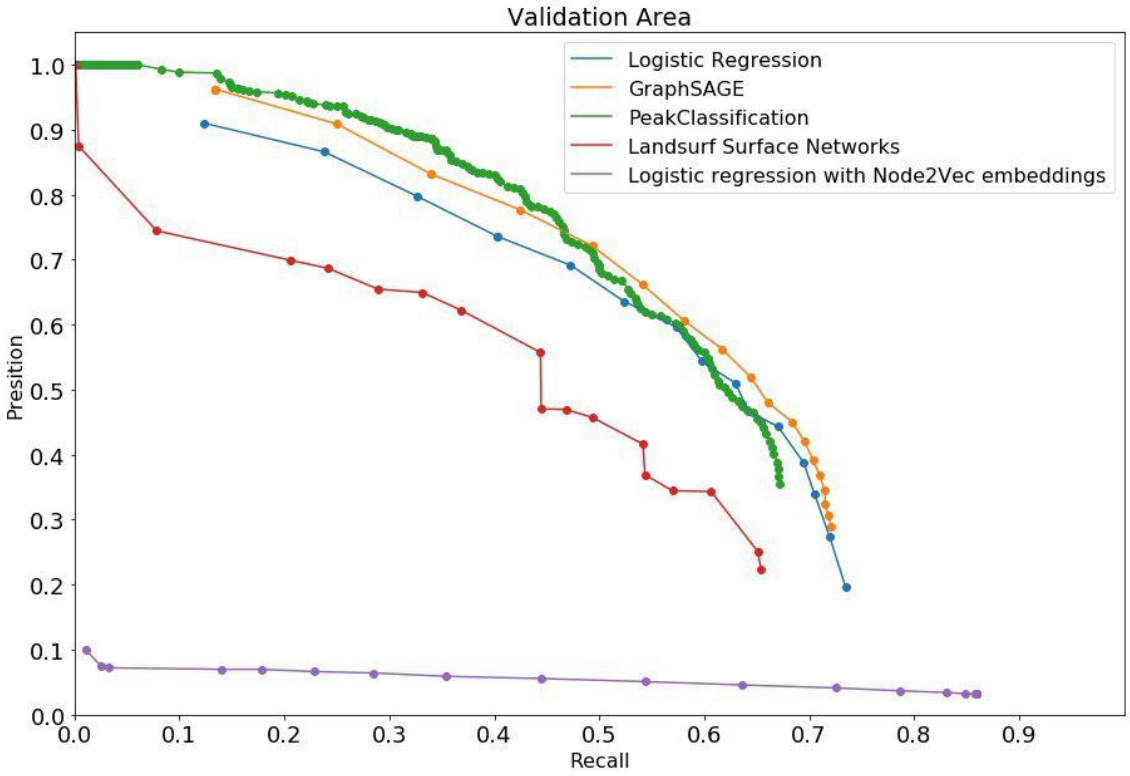

Figure 5.3: Pareto dominant precision-recall curve of the tested methods

5.3 Quantitative Analysis

In this section we present a quantitative comparison of the implemented Machine Learning models and the heuristic methods in terms of results achieved by analyzing the Precision-Recall curves in Figure 5.3 and the numerical results in Table 5.1.

Regarding the Machine Learning models we iterate over the output probabilities and calculate the Precision-Recall curve. We generate a sequence of “peakness” thresholds and filter the output set of probabilities for all the nodes of the graph. All the nodes having probability below the treshold were classified as "non-peaks" while the ones upper were classified as "peaks". For the heuristic methods, instead, for each parameters combination we generated a different list of peaks. Then, both for the machine learning models

and the heuristic methods, to determine if a peak in the provided list matches or not a real mountain peak in the GT data set we applied the procedure described in Section 4.8.

In the Precision-Recall curve in Figure 5.3 we can see that the Peak Classification (green) and GraphSAGE (orange) have the highest performance, with the first slightly better in some cases. Logistic Regression with Node2Vec (purple) embeddings does not perform well showing that the topology, which is its unique input, it is not a feature representative of the "peakness". Logistic Regression with handcrafted features (blue) also shows a similar performance to the best methods, even if lower than GraphSAGE, due to the fact the the structure of the graph is not included. Landserf Surface Network (red), instead, shows having inferior performance compared to the other heuristic method, Peak Classification, which remains the primary method of comparison. Also, for Landserf Surface Network, the precision-recall curve is not smooth, showing that is affected more than the other methods from the different parameter configurations.

	Parameters	Validation		Testing	
		Precision	Recall	Precision	Recall
Peak Classification	E1450-D20	0,7	0.49	0.64	0.56
Landserf Surface Network	W7-D1-S1-C6	0.7	0.2	0.66	0.2
Logistic Regression	0.3	0.7	0.38	0.64	0.41
GraphSAGE	0.327	0.7	0.48	0.65	0.51
Node2Vec	0.1	0.06	0.3	0.05	0.3

Table 5.1: Evaluation results

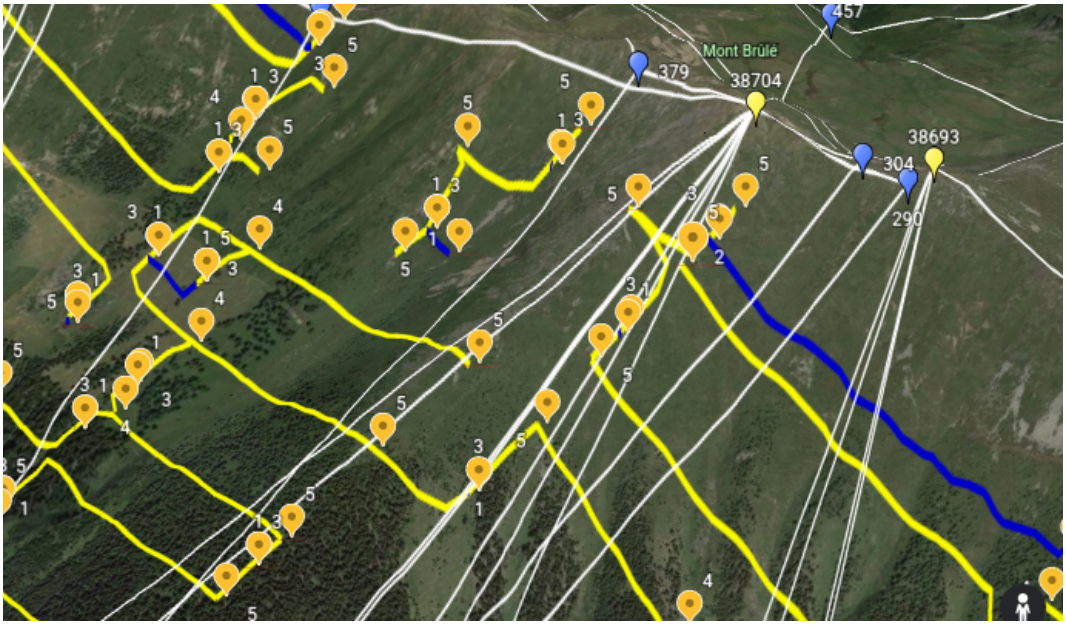

Figure 5.4: Comparison between a Metric Surface Network built with Landserf Tool and a Surface Network built with our custom method. The MSN critical paths are represented with the yellow and blue lines, respectively the ridges and channels. The orange pins represent the critical points of the MSN. The white lines represent instead the edges extracted with our method; still regarding our Surface Network, the yellow pins are the local maxima while the blue ones are the saddles. The local maxima with id 38704 in the figure is associated to a peak of the Ground Truth. It is possible to notice how the MSN critical points are disposed far from the Ground Truth peak and also an incompleteness regarding the critical paths of the MSN which do not always reach a critical point.

For each method, we select a point in the precision-recall curve with a reasonably good trade-off between precision and recall (specifically, we consider the point where all the methods get close to 70% precision). This level of precision it is suggested also by the analysis of the test predictions, showing similar results when the methods are applied in with the same configuration, i.e., it is a precision level which guarantees a good level of generalization. Indeed, in Table 5.1 Peak Classification and GraphSAGE, which are the two

best methods, in the same setting of parameters and features used to achieve 70% of precision in the validation set they reach around 65% of precision with comparable recall.

We selected Landserf Surface Network Classification as first heuristic algorithm as comparison method due to the underlying concept of the Surface Network created by searching critical points. Based on the results, it did not present a great performance in this area compared to the other methods, which we attribute not only to the method in which such surface network was created (Wolf-Pfaltz [72]), but also to the terrain features that are calculated in base to a series of parameters that act as additional filter. We can see in Figure 5.4 a Metric Surface Network [72] built with the Landserf tool, which is the base for the Surface Network Classification, compared with the Surface Network we implemented. In the MSN the critical points were not always reached by a critical path, such as a ridge or a channel, leading to an incomplete representation. Also, most of the critical points are disposed far from the ridges which contain most of the Ground Truth peaks. While other methods found in average $\approx$1100 peaks in the validation area Landserf Surface Classification only found 505.

Node2Vec, which uses only the topology of the network, is not able to create meaningful representations of the nodes sufficient for a classification task. Indeed the handcrafted features are rich in information derived from the elevations contained in the DEM that the topological structure cannot account by itself.

Regarding the other three methods Logistic Regression, GraphSAGE and Peak Classification, the Figure 5.3 shows three close curves. The effectiveness of Peak Classification is also supported by the work [7] in which the authors compared the effectiveness to Peak Classification over other state of the art methods for the same territory. The simple but effective reasoning behind this method is the definition of mountain encoded in the algorithm: a mountain is a point with a certain elevation higher than its neighbourhood by a given amount. Our grid search of the two parameters on Peak Classification aims to find the combination of elevation and drop values that lead to the best performance. In a similar way, Logistic Regression method searches

for the combination of values of the different selected features of the nodes. GraphSAGE follows a similar reasoning, however, it also takes into account the underlying graph topology, while Logistic Regression only takes as input a vector of features for each node. GraphSAGE aggregates the node features based on the graph structure, and this extra information allows the method to improve its performance with respect to Logistic Regression.

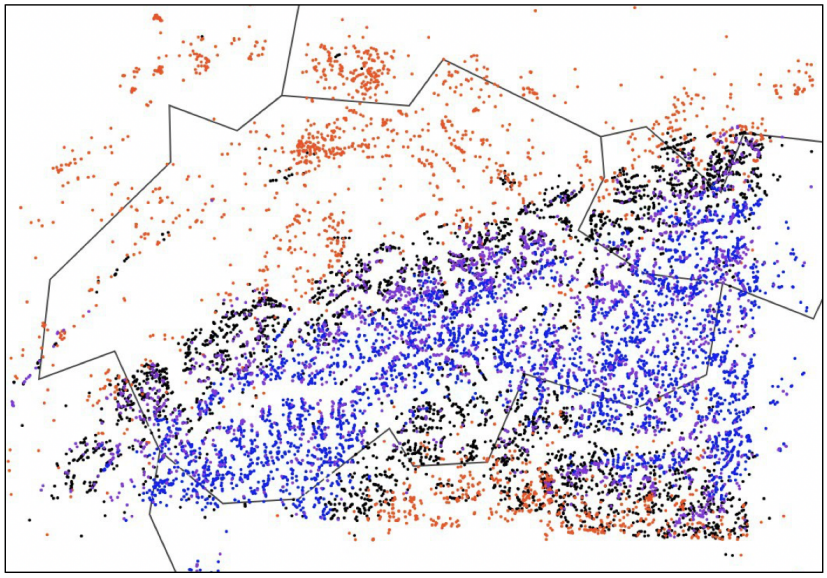

Figure 5.5: Clustering of the Ground Truth peaks based on the handcrafted features

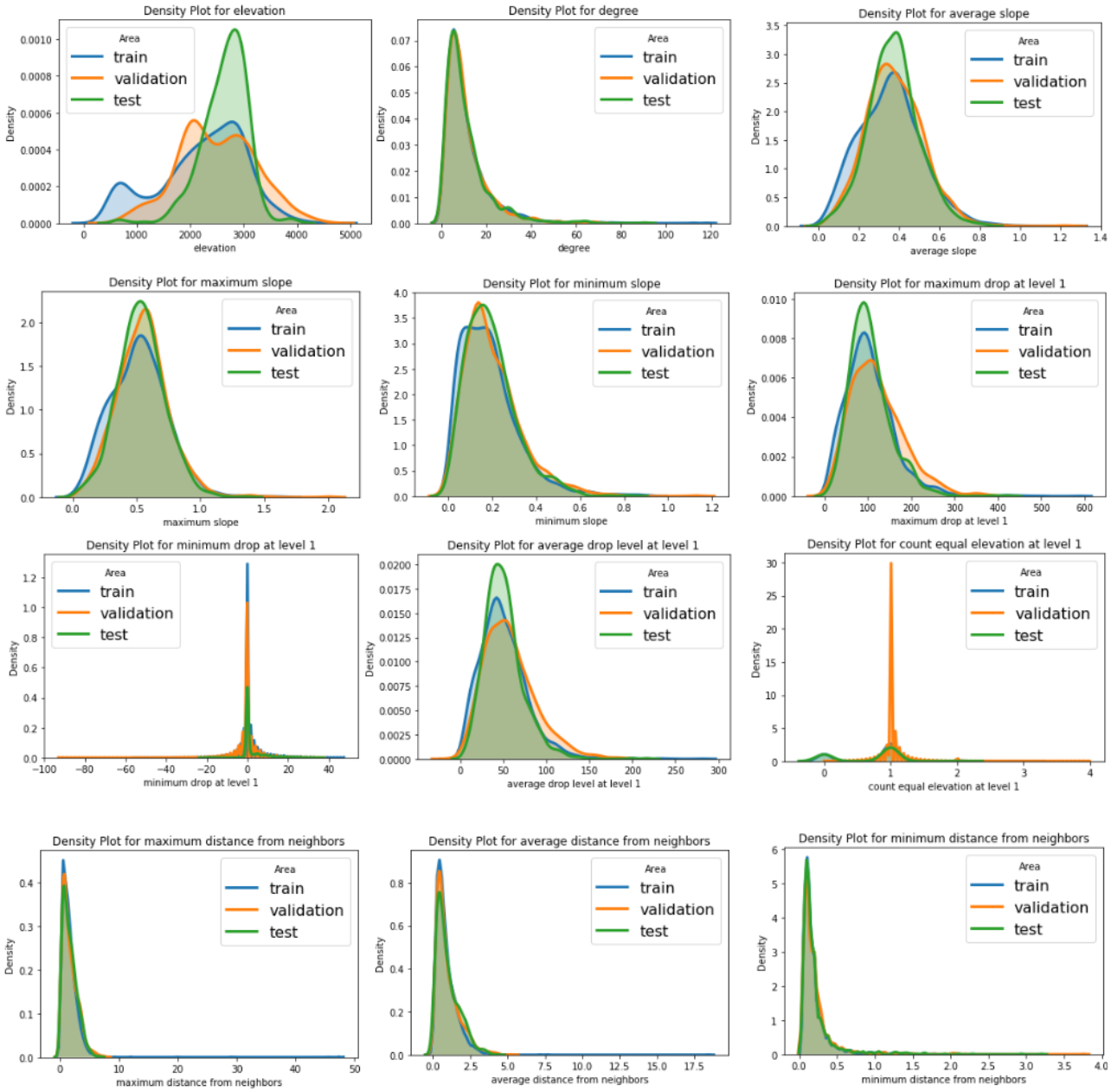

Figure 5.6: Distribution of the values for the main features for the peaks of the Ground Truth on the different areas of Train, Validation and Test

From the results in Figure 5.3 we infer that the features obtained from each node from the graph structure is good enough to carry a classification task even with such a trivial Machine Learning algorithm as Logistic Re-

gression, and yet if we add a more complex one that takes into account the topology of the graph we are able to improve the performance. This is not a contradiction of what we said about Node2Vec: the topology is not a sufficient information *alone*, but a method such as GraphSAGE can benefit from this information to aggregate knowledge from the neighborhood.

To contribute to the idea that the extracted features are useful, in Figure 5.5 we can observe the results of K-means clustering (k = 5, selected using elbow method). The figure shows the ground truth peaks where the color of the points represents the value of the cluster each point was assigned. From this figure we can appreciate three main areas: red, black and blue. In particular, based on their geographic localization it is easy to associate the black cluster with the pre-alps and the blue one with the alps. This is of particular interest, to understand that the features selected are in fact useful to capture enough geographical information to characterize the mountains. Also, the features have a quite uniform distribution over the different areas. Indeed, in Figure 5.6 we can see how most of the features of the Ground Truth peaks have really similar distribution across the different areas of train, validation and test. The few exceptions are given by the elevation which confirms the result of the clustering. Indeed in the validation and, especially, in the test areas most of the mountains are alpines having their elevations concentrated between the range 2000-3000m. For the training area, instead, we can see that the distribution has a concentration of values also around the 800-1000m, corresponding to mountains with lower elevation from the north of Switzerland.

5.4 Qualitative Analysis

Although a quantitative analysis, based on a specific ground truth data set, shows that Peak Classification, Logistic Regression and GraphSAGE discussed in Section 5.3 present a "similar" performance, a visual inspection of results was also performed.

To provide the reader an intuition of the mountain peaks that the methods

are able to identify, in Figure 5.7 we show examples of different mountain peaks found by GraphSAGE and Peak Classification for the True Positives have and agreement rate of $\approx 46\%$. One might think that given that Logistic Regression and GraphSAGE use the same data structure the agreement rate would be higher and in fact it is: $\approx 51\%$ but not for much.

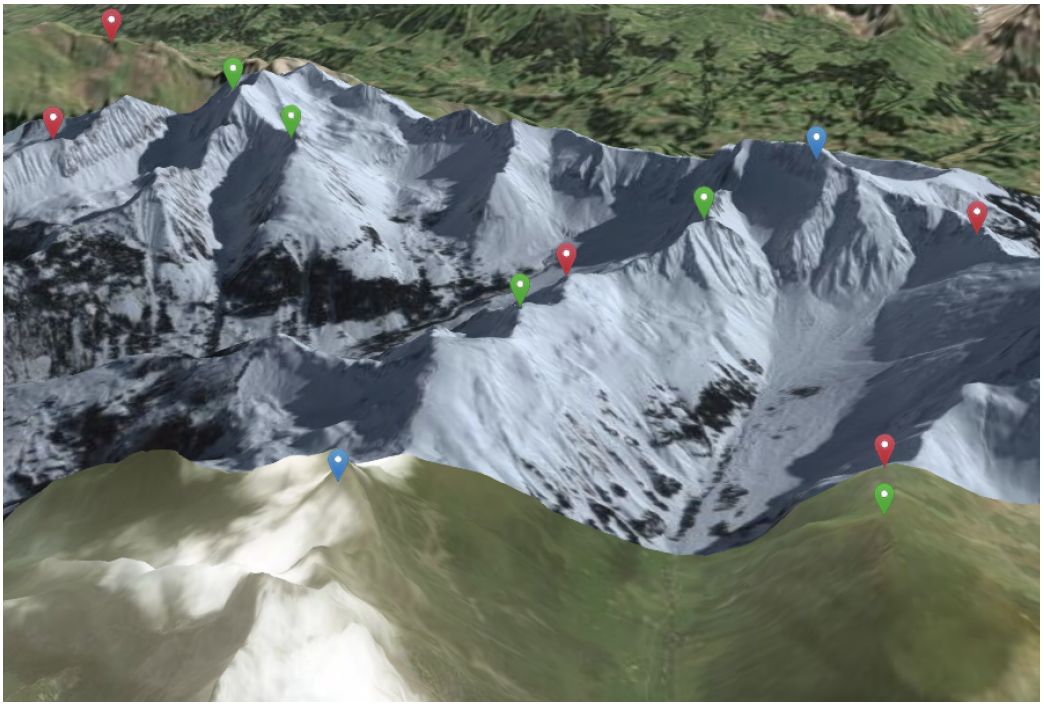

Figure 5.7: True positives examples found by: Peak Classificaion and GraphSage (green), Peak Classification (red) and GraphSAGE (blue)

Figure 5.8 presents examples of False Positives for GraphSAGE and Peak Classification. In this case, the agreement rate is of $\approx 50\%$. If we consider that we have methods that learned from a Ground Truth the characteristics (or feature values) that make a certain point in the Earth a mountain, and such methods agree that a given point not listed in the ground truth is considered a mountain, we can certainly think that such peaks might be missing from such list. In particular, from human observation, for example from Figure 5.8 of such points, we can consider that this is true for some of the cases.

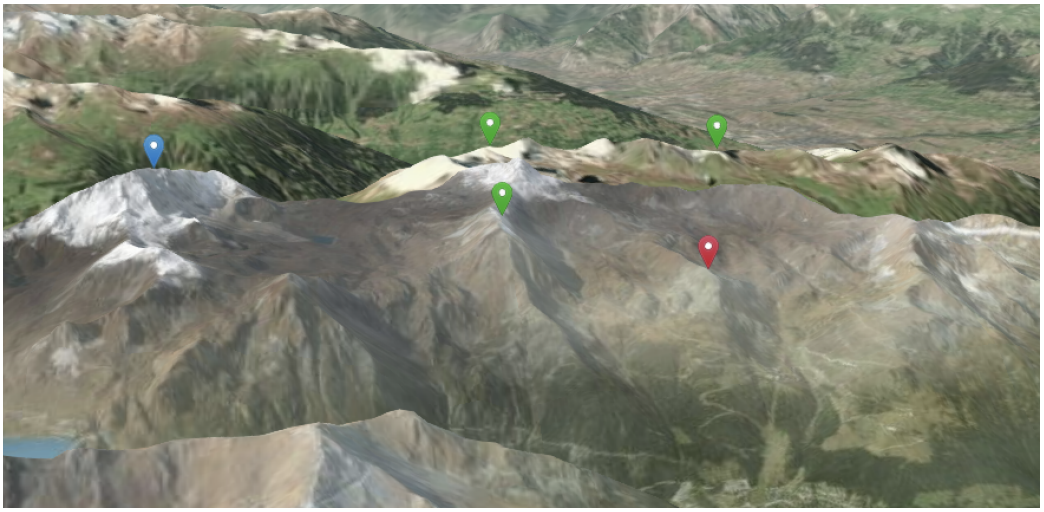

Figure 5.8: False Positives examples found by: Peak Classificaion and GraphSage (green), Peak Classification (red) and GraphSAGE (blue)

5.5 Impact of the Non-Morphological Nature of Groud Truth Peaks

In Section 5.4 we saw that due to the resolution constraints it happens than there are peaks from the Ground Truth which are not associated to any node of the graph. In fact, for DEM3 there are 259 peaks (15%) from the Ground Truth which are not matched with any node of the built Surface Network within an acceptable distance (200m).

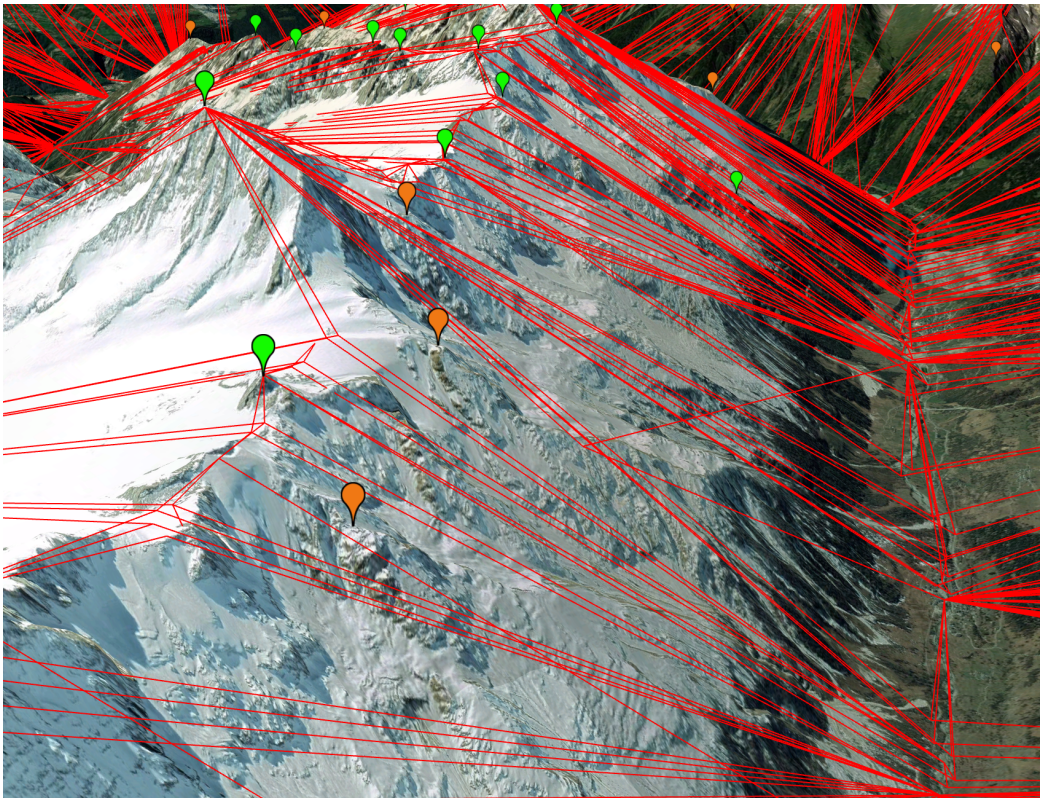

Figure 5.9: Surface Network and Ground Truth peaks: the red lines represent the edges of the surface, the green pins are local maxima of graph matching a Ground Truth peak, while orange ones are not associated to any node

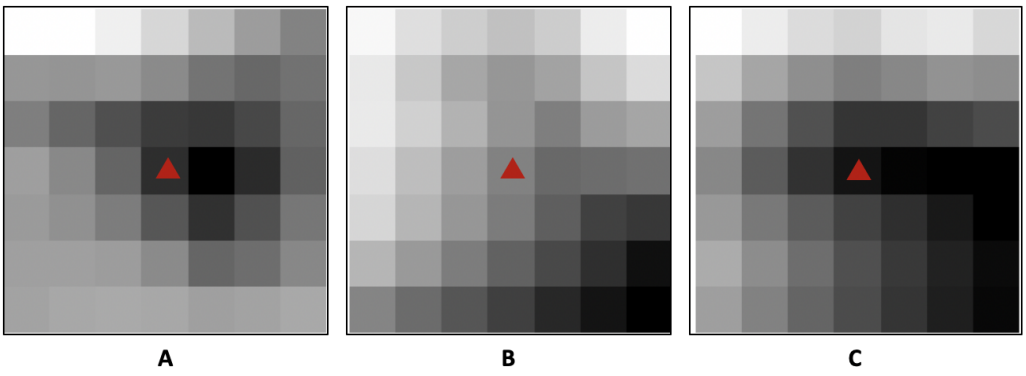

Figure 5.10: DEM representation on small areas around GT peaks, where darker colors indicate higher altitude and the red triangle indicates the positioning of the GT peak. A corresponds of a GT with a node, B and C corresponds to peaks no represented by any node.

In Figure 5.9 we show examples of peaks that were not associated to any node in the graph (orange markers) as well as the peaks that have a node associated and the edges. Given that, as mentioned before, the DEM is an approximation of the Earth, if we observe the raw values we can understand why a local maxima was not created in the graph close to those areas. In Figure 5.10 we show examples of DEM values for some of the peaks displayed in Figure 5.9. For the case of A, we can see that the higher point is the immediate neighbour cell to the one that corresponds to the GT peak ($\approx 90m$), while in the other two cases it is not clear that the point is a local maxima, and thus no node is associated.

By making a comparison with Peak Classification we find that 159 of those 259 missed peaks are shared as False Negatives, i.e., that also the heuristic method is not able to find. We inspect then the other unmatched peaks to understand if there is a pattern between them. The peaks we analyzed can be related with the cases in Figures 5.11 and 5.12.

In Figure 5.11 the image in the right shows that at the location of the Ground Truth peak (in blue) there is the beginning of an area with higher elevation, more specifically a sequence that can be associated to a ridge. Another example, in Figure 5.12, shows the same situation, where the coor-

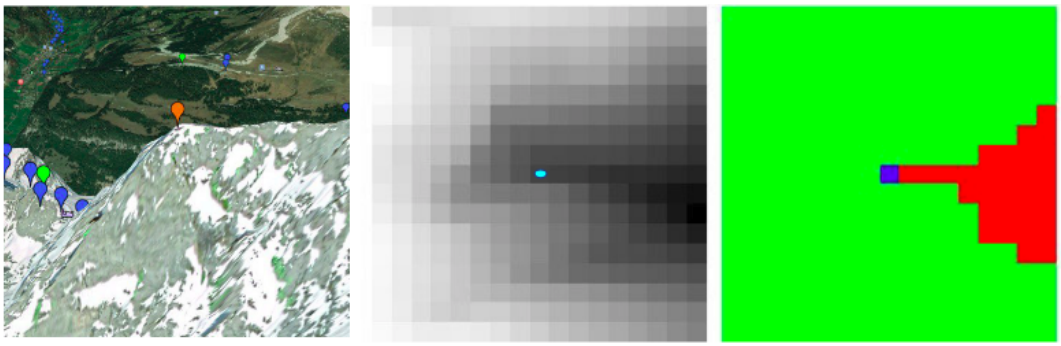

Figure 5.11: Example of missed peak of the Ground Truth. In the figure in the left we have in orange a missed peak of the Ground Truth. In the middle figure there is a gray scale of the elevations for the surrounding area, where the light blue marked pixel is the location of the GT peak. Darker gray represents higher elevation. In the right figure with the red color is represented the area having higher elevation than the GT peak (in blue) while in green delimited the area with lower elevation.

dinates of the Ground Truth peak is at the extremity of a ridge.

The remaining unmatched peaks cannot be associated to any morphological landform. This makes emerge again the issue of defining "what" is a mountain. Most of the unmatched peaks of the Ground Truth are not well defined morphologically; indeed they are present in the maps for cultural and historical reasons. Considering that the Surface Network we implemented is based only in the identification of the critical points associated to the landforms of a surface, these "non-morphological" peaks can not be associated to the graph and a measure of the goodness of the models should be performed without taking into account these spefic kind of peaks.

To account with the impact of the non-morphological nature of ground truth peaks over the Machine Learning models we replicated the experiments by not including in the set of false negatives the peaks of the Ground Truth which were not matched with any of the nodes of the graph, i.e., they were not in the range of 200m. We excluded then the 259 false negatives associated to the unmatched peaks from the computation of the precision-recall to allow a better assessment of the quality of the methods.

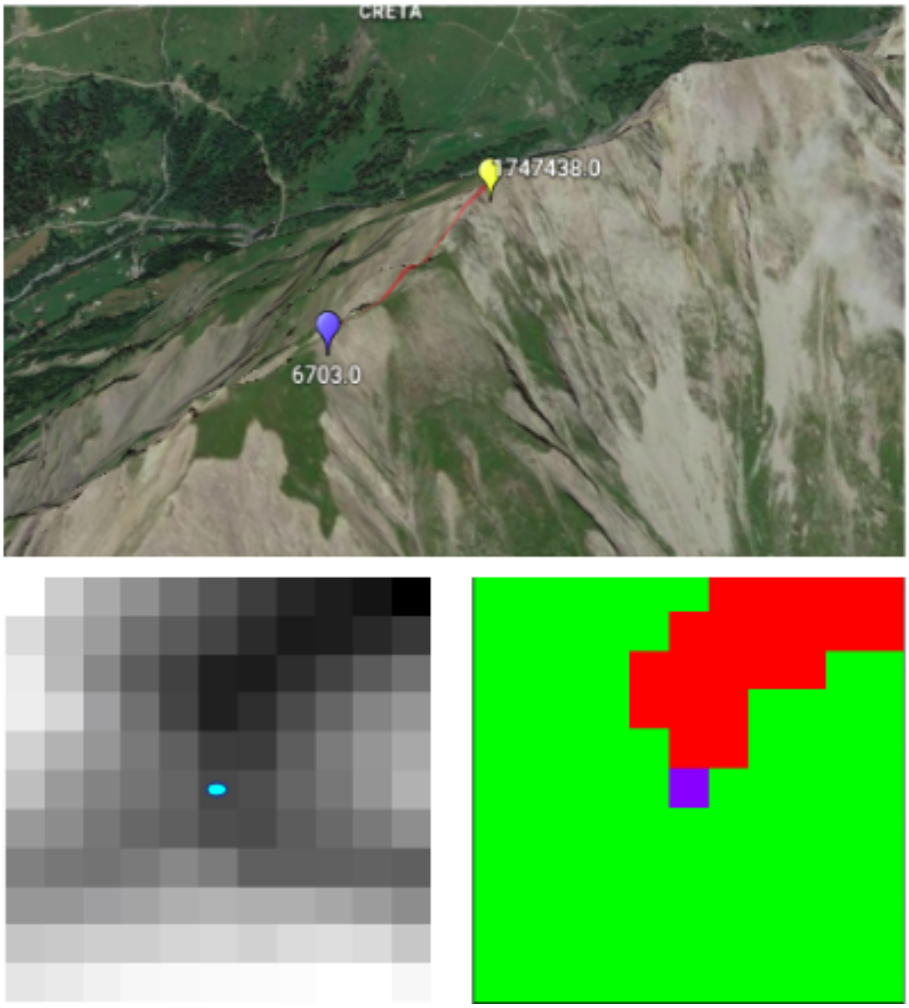

Figure 5.12: Example of missed peak of the Ground Truth. In the upper figure we have an example of a Ground Truth peak (in blue) which it wasn't possible to be assigned to any local maxima within a range of 200m. The closest local maxima is indicated in yellow at a distance of 330m. In the bottom left figure it is represented the elevation of the area on a gray scale; darker pixels mean higher elevation for the corresponding cell. The pixel at the center of the figure (with the light blue marker) is related with the elevation of the Ground Truth peak while the darkest pixel in the upper right position is the one of the local maxima. To understand better the difference of elevation, in the bottom right figure we can see in blue the pixel for the missed Ground Truth, in red all the pixels with higher elevations and in green all the pixels with lower elevation.

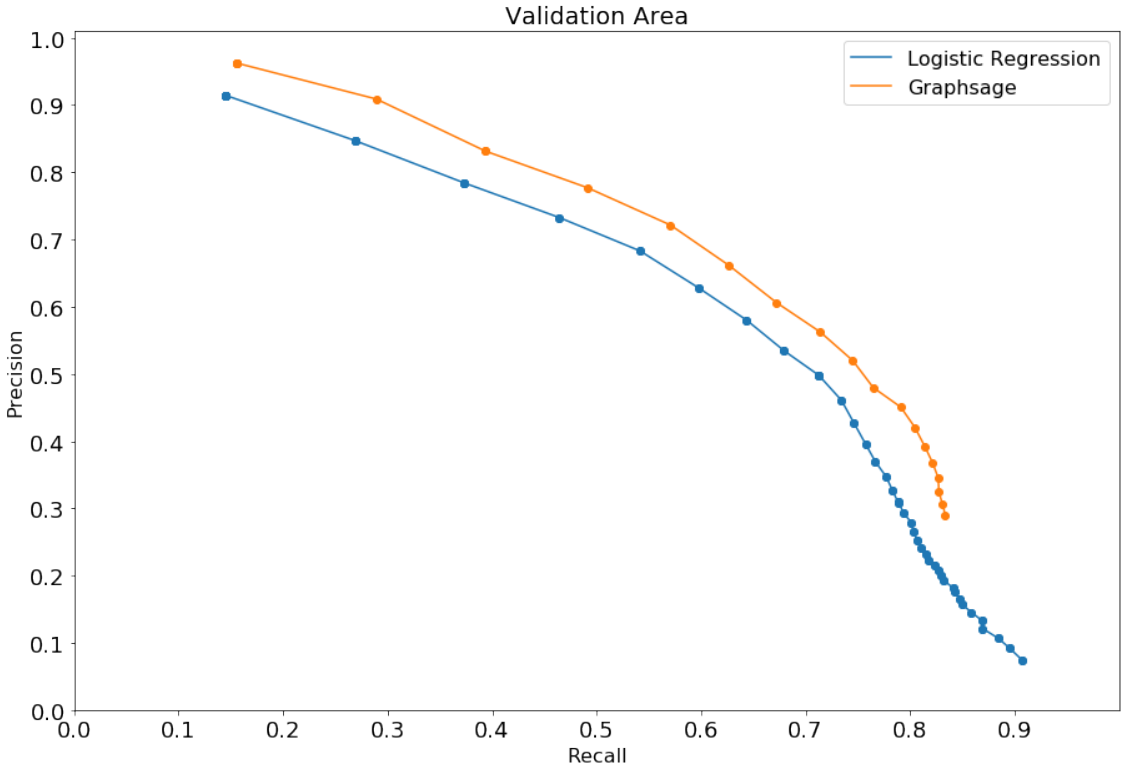

Figure 5.13: Pareto dominant precision-recall curve of the Machine Learning methods

	Parameters	Validation		Testing	
		Precision	Recall	Precision	Recall
Logistic Regression	0.3	0.7	0.46	0.63	0.49
GraphSAGE	0.327	0.7	0.55	0.62	0.50

Table 5.2: Evaluation results

Fromm the curves in 5.13 we can see that GraphSAGE still performed slightly better than Logistic Regression. They key factor for the best results of GraphSAGE is in exploiting entirely the surface network, both the signals and the structure, representing a sort of relationship between the nodes properties. Logistic Regression exploits only the handcrafted features, which have, however, a good quality for performing a classification of the nodes.

We can say that the proposed methods have a good performance comparable with the state of the art heuristic methods. When excluding the non-morphological peaks the models show a discrete improvement achieving also a good generalization capacity as showed in Table 5.2 over the test set.

Chapter 6

Conclusion and future work

In this thesis we study the use of Machine Learning techniques for node classification for learning to find peaks in surface networks created from DEM files, an approach that to the best of our knowledge has not been employed in the past for this purpose. The proposed method requires only a Ground Truth dataset, which may be chosen based on availability or suitability to the specific application scenario, with the possibility to be scaled to test areas of any size. We merged two different open source datasets of peaks and removed the duplicated ones, based on a distance rule that proved to be efficient; this may vary depending on the quality of the initial data.

We replicated state of the art heuristic methods for finding peaks and we faced with the main issues of these kind of methods: first there is an amount of parameters the user must set to be able to obtain the desired terrain features and second problem is the lack of a fair comparison between different methods. We agree with the literature on the difficulty of defining "what is a mountain".

We analyzed the problem of peak extraction and developed a process to transform the DEMs into Surface Networks, the training data for the Machine Learning models and, more specifically, for the Graph Deep Learning ones. We use the elevation to find the critical points of the Earth's Surface, among which there are represented the peaks, and we enrich the graph with meaningful signals representing the properties of the surface.

With such data, we managed to train a model that in the selected area performs equally with the state of the art methods. The main reason for which it wasn't possible to improve the results of the heuristic methods is that a set of peaks are not assigned to any node of the Surface Network. The inclusion of those Ground Truth peaks in the graph through the exploitation of other kind of landforms can be a future step.

We provide an in-depth quantitative analysis of the three Machine Learning architectures and discuss on why one architecture is better than the other. GraphSAGE, with all the handcrafted features, is able to achieve the best result because it exploits both the rich signals of the nodes and both the knowledge of the disposal of the nodes in the graph. Logistic Regression performs quite well, even if not like GraphSAGE, showing that the features of the Surface Network are slightly representative. Node2Vec shows, instead, that the topology alone is not sufficient for this kind of classification.

An evaluation in Switzerland territory, a region characterized by highly dense peak distribution, showed promising results. Still, we divided the evaluation of our method in two steps: (1) the quality of the data structure (surface network) and (2) the performance of the method, given that the methods were not able to classify nodes that are not present in the graph, since the graph could not contain local maxima that the resolution of the DEM does not allow to find.

6.1 Future work

Our future work pursues a number of directions:

- since using higher DEM resolution to create the surface network introduces too much of noise to the graph, we will investigate how to use a multi-scale approach to add information missing with lower resolution while preventing excessive noise on the data;
- given that the approaches proposed showed a promising results, we can investigate the use of more complex method such us geometric deep learning ones [11];

- use ensemble learning to improve performance, by taking the best of image- and graph-based terrain representations and DL models;
- introduce new kind of landforms in the surface network, such as at he endpoints of the ridges, which may allow to assign part of the unlabeled peaks from the Ground Truth to one of the nodes of the graph.

Bibliography

[1] Kakoli Saha, Neil A Wells, and Mandy Munro-Stasiuk. «An object-oriented approach to automated landform mapping: A case study of drumlins». In: *Computers & geosciences* 37.9 (2011), pp. 1324–1336.

[2] Bernhard Klingseisen, Graciela Metternicht, and Gernot Paulus. «Geomorphometric landscape analysis using a semi-automated GIS-approach». In: *Environmental Modelling & Software* 23.1 (2008), pp. 109–121.

[3] Linda H Graff and E Lynn Usery. «Automated classification of generic terrain features in digital elevation models». In: *Photogrammetric Engineering and Remote Sensing* 59.9 (1993), pp. 1409–1417.

[4] Mike J Smith and Chris D Clark. «Methods for the visualization of digital elevation models for landform mapping». In: *Earth Surface Processes and Landforms* 30.7 (2005), pp. 885–900.

[5] Peter Fisher and Jo Wood. «What is a Mountain? Or The Englishman who went up a Boolean Geographical Concept but Realised it was Fuzzy». In: *Geography* 83.3 (1998), pp. 247–256.

[6] Peter Fisher, Jo Wood, and Tao Cheng. «Where is Helvellyn? Fuzziness of multi-scale landscape morphometry». In: *Transactions of the Institute of British Geographers* 29.1 (2004), pp. 106–128.

[7] Rocio Nahime Torres et al. «A Deep Learning model for identifying mountain summits in Digital Elevation Model data». In: *2018*

IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE). IEEE. 2018, pp. 212–217.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. «Deep learning». In: *nature* 521.7553 (2015), p. 436.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.* 2012, pp. 1106–1114. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[10] S Rana. «Experiments on the generalisation and visualisation of surface networks». In: (2000).

[11] Federico Monti et al. «Geometric deep learning on graphs and manifolds using mixture model CNNs». In: *CoRR* abs/1611.08402 (2016). arXiv: 1611.08402. URL: http://arxiv.org/abs/1611.08402.

[12] Shigeo Takahashi. «Algorithms for Extracting Surface Topology from Digital Elevation Models». In: May 2006, pp. 31–51. ISBN: 9780470020289. DOI: 10.1002/0470020288.ch3.

[13] URL: https://en.wikipedia.org/wiki/Logistic_regression (visited on 11/24/2019).

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. «Inductive Representation Learning on Large Graphs». In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

[15] Aditya Grover and Jure Leskovec. «node2vec: Scalable Feature Learning for Networks». In: *CoRR* abs/1607.00653 (2016). arXiv: 1607.00653. URL: http://arxiv.org/abs/1607.00653.

[16] Thomas K.Peucker David H.Douglas. «Detection of Surface-Specific Points by Local Parallel Processing of Discrete Terrain Elevation Data». In: *Computer Graphics and Image Processing* 4.4 (1975), pp. 375–387. DOI: 10.1016/0146-664X(75)90005-2. URL: https://doi.org/10.1016/0146-664X(75)90005-2.

[17] Jay Lee, PF Fisher, and PK Snyder. «Modeling the effect of data errors on feature extraction from digital elevation models». In: *Photogrammetric Engineering and Remote Sensing* 58 (1992), pp. 1461–1461.

[18] J Wood. «Geomorphometry in LandSerf». In: *Developments in soil science* 33 (2009), pp. 333–349.

[19] Peter Fisher, Jo Wood, and Tao Cheng. «Fuzziness and ambiguity in multi-scale analysis of landscape morphometry». In: *Fuzzy modeling with spatial information for geographic problems*. Springer, 2005, pp. 209–232.

[20] Bernhard Schneider and Jo Wood. «Construction of Metric Surface Networks from Raster-Based DEMs». In: May 2006, pp. 53–70. ISBN: 9780470020289. DOI: 10.1002/0470020288.ch4.

[21] Tomaž Podobnikar. «Method for determination of the mountain peaks». In: *12th AGILE International Conference on Geographic Information Science*. 2009.

[22] Tomaž Podobnikar. «Moutain peaks determination supported with shapes analysis». In: *Geographia Technica* (2010), pp. 111–119.

[23] Tomaž Podobnikar. «Detecting mountain peaks and delineating their shapes using digital elevation models, remote sensing and geographic information systems using autometric methodological procedures». In: *Remote Sensing* 4.3 (2012), pp. 784–809.

[24] Y Deng and JP Wilson. «Multi-scale and multi-criteria mapping of mountain peaks as fuzzy entities». In: *International Journal of Geographical Information Science* 22.2 (2008), pp. 205–218.

[25] Jarosław Jasiewicz and Tomasz F. Stepinski. «Geomorphons — a pattern recognition approach to classification and mapping of landforms». In: *Geomorphology* 182 (2013), pp. 147–156. ISSN: 0169-555X. DOI: https://doi.org/10.1016/j.geomorph.2012.11.005. URL: http://www.sciencedirect.com/science/article/pii/S0169555X12005028.

[26] Calogero Schillaci, Andreas Braun, and Jan Kropáček. «2.4. 2. Terrain analysis and landform recognition». In: *Geomorphological Techniques (Online Edition)*. British Society for Geomorphology, 2016. ISBN: "2047-0371".

[27] Andrew Kirmse and Jonathan de Ferranti. «Calculating the prominence and isolation of every mountain in the world». In: *Progress in Physical Geography* 41.6 (2017), pp. 788–802.

[28] Jürgen Schmidhuber. «Deep Learning in Neural Networks: An Overview». In: *CoRR* abs/1404.7828 (2014). arXiv: 1404.7828. URL: http://arxiv.org/abs/1404.7828.

[29] *Artificial Neuron*. https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[31] *Deep Feedforward Neural Network*. .

[32] Jordi Torres. *Learning Process of a Neural Network*. 2018. URL: https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7 (visited on 02/26/2019).

[33] Yanming Guo et al. «Deep learning for visual understanding: A review». In: *Neurocomputing* 187 (2016), pp. 27–48.

[34] Jonathan Long, Evan Shelhamer, and Trevor Darrell. «Fully convolutional networks for semantic segmentation». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[35] Nicolas Audebert, Bertrand Le Saux, and Sébastien Lefèvre. «Semantic segmentation of earth observation data using multimodal and multi-scale deep networks». In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 180–196.

[36] Dimitrios Marmanis et al. «Semantic segmentation of aerial images with an ensemble of CNNs». In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 3 (2016), p. 473.

[37] Dimitrios Marmanis et al. «Deep neural networks for above-ground detection in very high spatial resolution digital elevation models». In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2.3 (2015), p. 103.

[38] Zixuan Chen, Xuewen Wang, Zekai Xu, et al. «Convolutional Neural Networks based DEM super resolution». In: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 41 (2016).

[39] Eric Guérin et al. «Interactive example-based terrain authoring with conditional generative adversarial networks». In: *ACM Transactions on Graphics (TOG)* 36.6 (2017), p. 228.

[40] Christopher Beckham and Christopher Pal. «A step towards procedural terrain generation with GANs». In: *arXiv preprint arXiv:1707.03383* (2017).

[41] P. Fraternali R. N. Torres F. Milani and D. Frajberg. «A Deep Learning model for identifying mountain summits in Digital Elevation Model data». In: *2018 IEEE International Conference on Artificial Intelligence and Knowledge Engineering*. Oct. 2018.

[42] R.J. Trudeau. *Introduction to Graph Theory*. Dover Books on Mathematics. Dover Pub., 1993. ISBN: 9780486678702. URL: `https://books.google.it/books?id=8nYH5OYEW24C`.

[43] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. «A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications». In: *CoRR* abs/1709.07604 (2017). arXiv: 1709.07604. URL: http://arxiv.org/abs/1709.07604.

[44] William L. Hamilton, Rex Ying, and Jure Leskovec. «Representation Learning on Graphs: Methods and Applications». In: *CoRR* abs/1709.05584 (2017). arXiv: 1709.05584. URL: http://arxiv.org/abs/1709.05584.

[45] Jie Zhou et al. «Graph Neural Networks: A Review of Methods and Applications». In: *CoRR* abs/1812.08434 (2018). arXiv: 1812.08434. URL: http://arxiv.org/abs/1812.08434.

[46] Joan Bruna et al. «Spectral Networks and Locally Connected Networks on Graphs». In: *CoRR* abs/1312.6203 (2013). arXiv: 1312.6203. URL: http://arxiv.org/abs/1312.6203.

[47] Ron Levie et al. «CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters». In: *CoRR* abs/1705.07664 (2017). arXiv: 1705.07664. URL: http://arxiv.org/abs/1705.07664.

[48] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. «DeepWalk: Online Learning of Social Representations». In: *CoRR* abs/1403.6652 (2014). arXiv: 1403.6652. URL: http://arxiv.org/abs/1403.6652.

[49] Palash Goyal and Emilio Ferrara. «Graph Embedding Techniques, Applications, and Performance: A Survey». In: *CoRR* abs/1705.02801 (2017). arXiv: 1705.02801. URL: http://arxiv.org/abs/1705.02801.

[50] Shaosheng Cao, Wei Lu, and Qiongkai Xu. *Deep Neural Networks for Learning Graph Representations*. 2016. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12423.

[51] Daixin Wang, Peng Cui, and Wenwu Zhu. «Structural Deep Network Embedding». In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 1225–1234. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939753. URL: http://doi.acm.org/10.1145/2939672.2939753.

[52] Thomas N. Kipf and Max Welling. «Semi-Supervised Classification with Graph Convolutional Networks». In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[53] Trang Pham et al. «Column Networks for Collective Classification». In: *CoRR* abs/1609.04508 (2016). arXiv: 1609.04508. URL: http://arxiv.org/abs/1609.04508.

[54] David K Duvenaud et al. «Convolutional Networks on Graphs for Learning Molecular Fingerprints». In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2224–2232. URL: http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf.

[55] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. «Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering». In: *CoRR* abs/1606.09375 (2016). arXiv: 1606.09375. URL: http://arxiv.org/abs/1606.09375.

[56] Jaekoo Lee et al. *Transfer Learning for Deep Learning on Graph-Structured Data*. 2017. URL: https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14803.

[57] Zonghan Wu et al. «A Comprehensive Survey on Graph Neural Networks». In: *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596. URL: http://arxiv.org/abs/1901.00596.

[58] Sanjay Rana and Jeremy Morley. «Surface Networks». In: *Rana, Sanjay and Morley, Jeremy (2002) Surface networks. Working paper. CASA Working Papers (43). Centre for Advanced Spatial Analysis (UCL), London, UK.* (July 2008).

[59] Tom G. Farr et al. «The Shuttle Radar Topography Mission». In: *Reviews of Geophysics* 45 (2007).

[60] Jo Wood. *Modelling the continuity of surface form using digital elevation models.* Jan. 1998.

[61] B Schneider. «Surface networks: extension of the topology and extraction from bilinear surface patches». In: (Jan. 2003).

[62] Diederik Kingma and Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *International Conference on Learning Representations* (Dec. 2014).

[63] Tomas Mikolov et al. «Distributed Representations of Words and Phrases and their Compositionality». In: *Advances in Neural Information Processing Systems 26.* Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[64] Zellig S. Harris. «Distributional Structure». In: *WORD* 10.2-3 (1954), pp. 146–162. DOI: 10.1080/00437956.1954.11659520. eprint: https://doi.org/10.1080/00437956.1954.11659520. URL: https://doi.org/10.1080/00437956.1954.11659520.

[65] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[66] Jo Wood. *Landserf.* 2009. URL: http://www.landserf.org/ (visited on 11/10/2019).

[67] Aditya Grover and Jure Leskovec. «node2vec: Scalable feature learning for networks». In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, pp. 855–864.

[68] Benoıt Frénay and Michel Verleysen. «Classification in the Presence of Label Noise: A Survey». In: *IEEE Trans. Neural Netw. Learning Syst.* 25.5 (2014), pp. 845–869. DOI: 10.1109/TNNLS.2013.2292894. URL: https://doi.org/10.1109/TNNLS.2013.2292894.

[69] URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 11/25/2019).

[70] URL: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient (visited on 11/24/2019).

[71] Takaya Saito and Marc Rehmsmeier. «The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets». In: *PloS one* 10.3 (2015), e0118432.

[72] GW Wolf. «Metric surface networks». In: *Proc. 5th Intl. Symp. Spatial Data Handling*. 1990, pp. 844–856.

Appendix A

Deep Learning Tests

In this Chapter we are showing how we performed the choice of the best architecture for the Deep Learning models, GraphSAGE and Node2Vec.

A.1 GraphSAGE

In this Section we show the explored hyper-parameters space for choosing the best architecture for the GraphSAGE model.

A.1.1 Hyper-parameters space analysis

For deciding which hyper-parameter values to test in the Grid Search for the model tuning we performed a sensitivity analysis; it is the study of how the uncertainty in the output of a mathematical model or system (numerical or otherwise) can be divided and allocated to different sources of uncertainty in its inputs. In this case the uncertainty is about the impact that the different hyperparameters have on the final model performance. In table A.1 we can see the tested values for the Sensitivity Analysis for GraphSAGE hyper parameters. We can see that for the core parameters of the model, such as the *aggregator*, *sampling dimension* and *embedding dimension* respectively in figures A.1,A.2 (level 1), A.4 (level 2) and A.3 (level 1), A.5 (level2) there is no clue in the choice for creating the Grid Search, so opted for two values, the default one plus another having good performance. For parameters such

Hyper-Parameter	Value1	Value2	Default	Value3	Value4
Aggregator	Mean Pool	Max Pool	Mean	LSTM	
Class Weight	0.5	0.6		0.7	0.8
Max Degree	8	16	128	64	64
Samples1	5	10	25	15	
Samples2	0	5	10	15	25
Samples3	5	10	0	15	20
Batch Size	64	128	512	256	1024
Validate Iteration	1250	2500	5000	10000	20000
Validate Batch Size	32	64	256	128	512
Dropout	0.2	0.5	0	0.7	
Weight Decay	0.001	0.01	0		
Dimension Embeddings 1	8	32	128	512	
Dimension Embeddings 2	8	32	128	512	
Learning Rate	0.0001	0.01	0.01	0.1	0.00001

Table A.1: Explored values with the sensitivity Analysis

as *weight decay* and *dropout* which are more inherent with Deep Learning models in general, we can see that the standard value is already the best, so there was no reason for evaluating also the others.

Although we didn't explore in deep the performance implications, this analysis helps also in reducing the parameters space considering that the time for the training is not trivial.

Based on the impact of the values we could reduce the space search to the ones in table A.2 for a total number of 3072 experiments. In bold we have the chose value after the Grid Search. We want to highlight that when we created the experiments for finding the best architecture we were not aware of the group drop features at any level. Indeed we can see that for 70% precision the recall is around 39%, 9% less compared of what we achieved by including also the drop features. For a lack of time we couldn't repeat all the experiments of the tuning but we believe that including also the drop

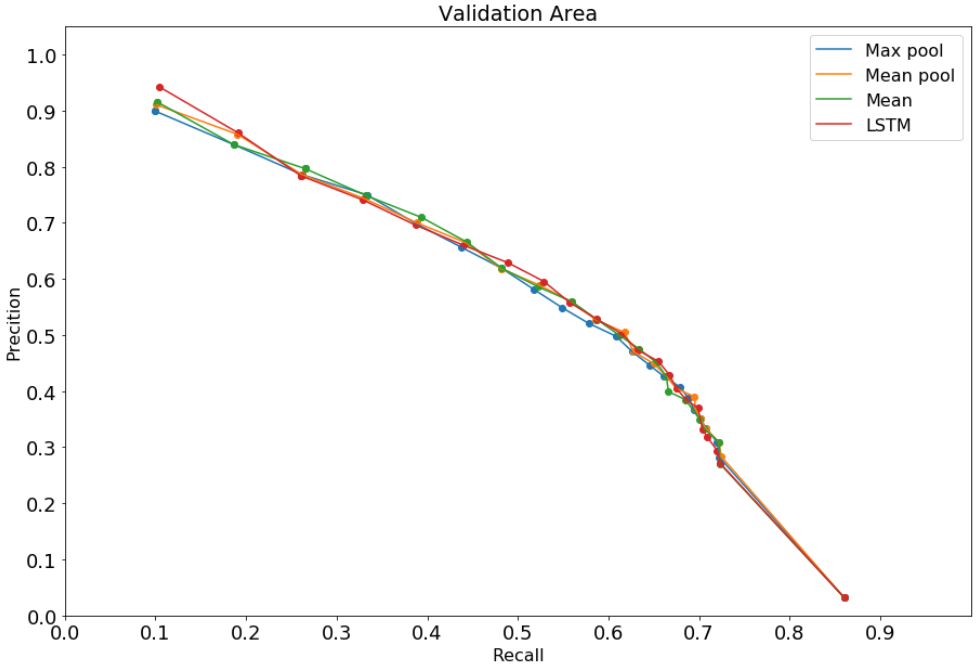

Figure A.1: Effect of the variation of the Aggregator

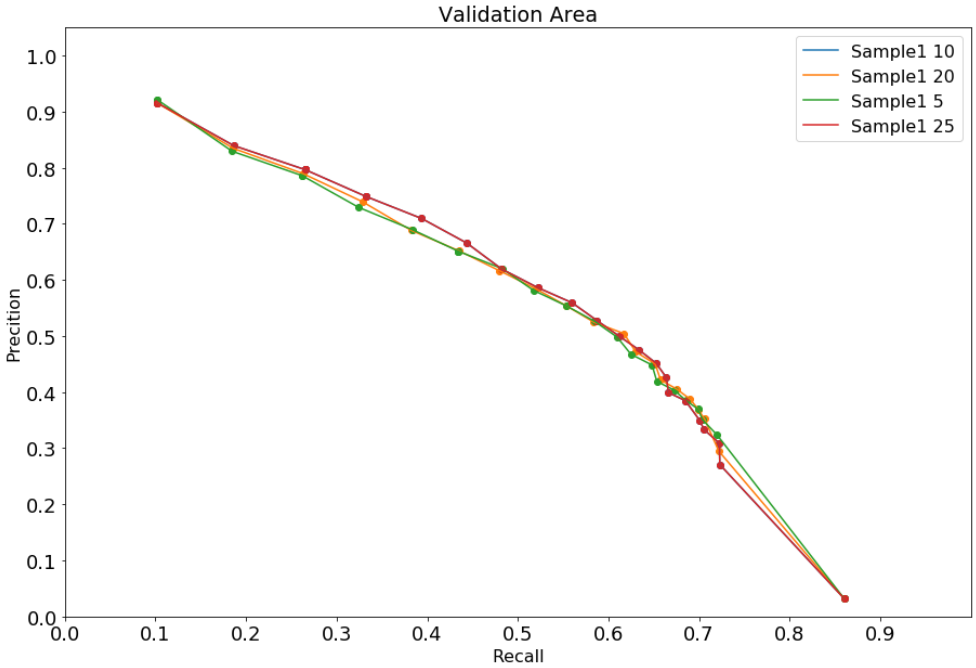

Figure A.2: Effect of the sampling at layer 1

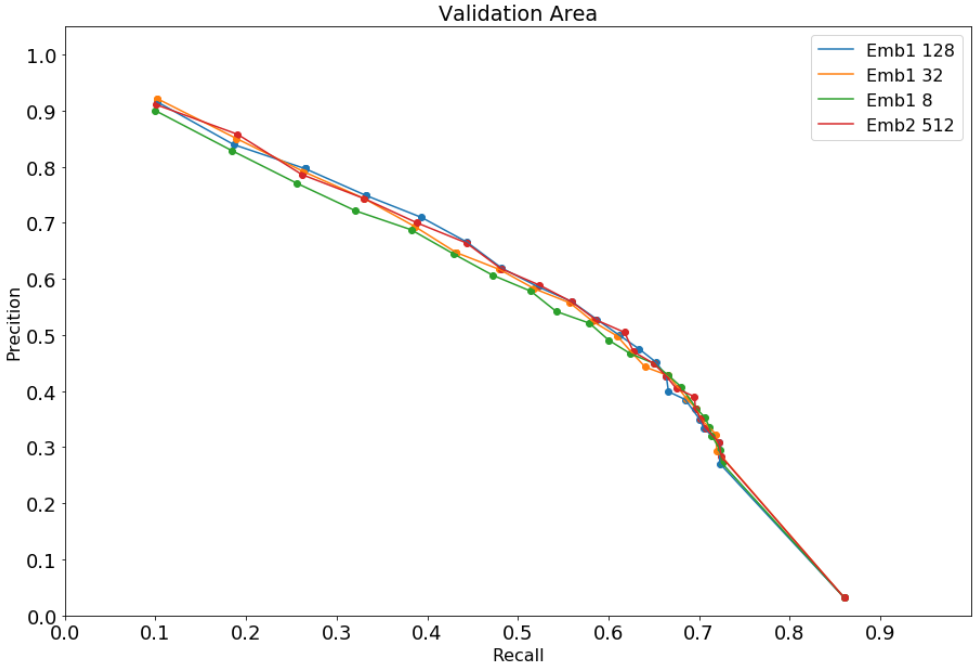

Figure A.3: Effect of the dimension of the embeddings at layer 1

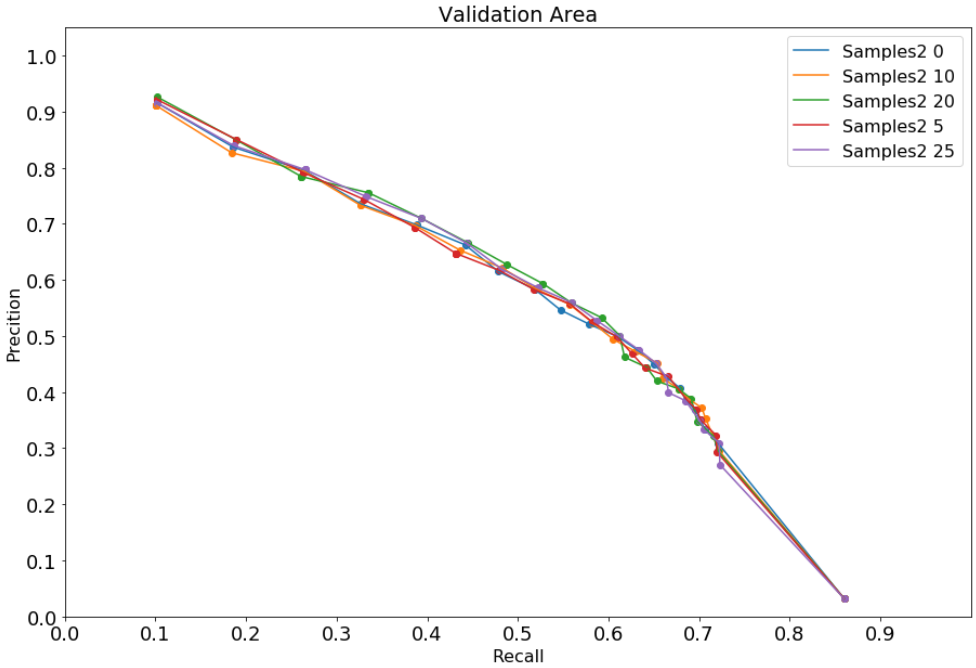

Figure A.4: Effect of the sampling at layer 2

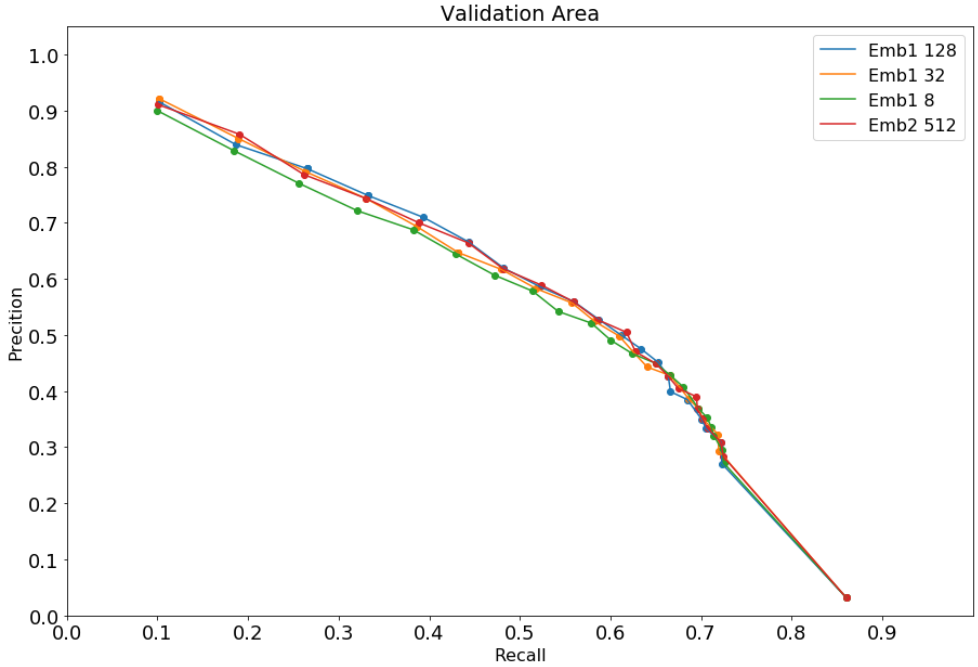

Figure A.5: Effect of the dimension of the embeddings at layer 2

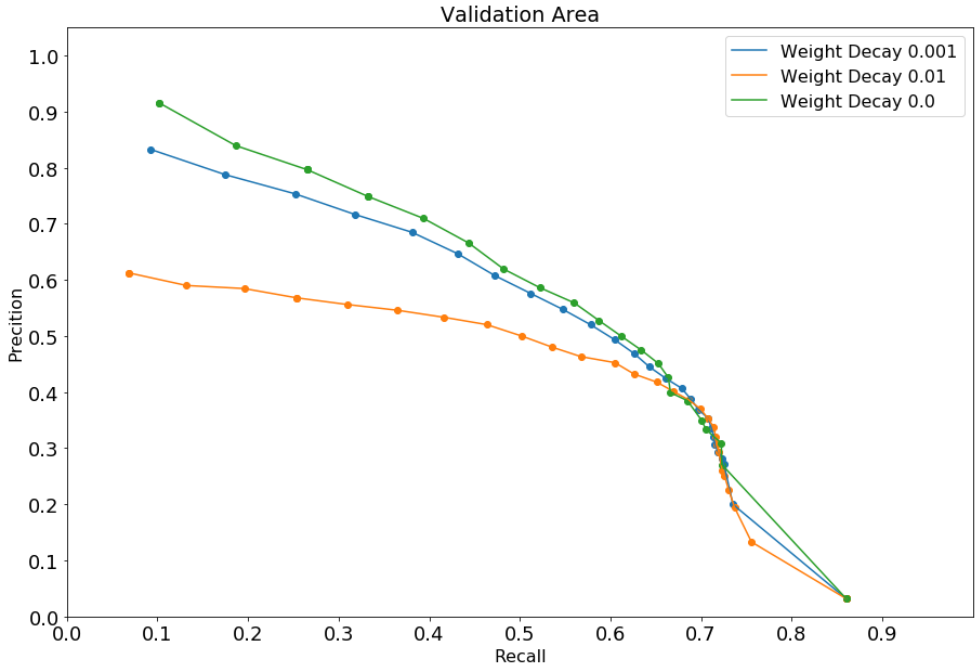

Figure A.6: Effect of the weight decay

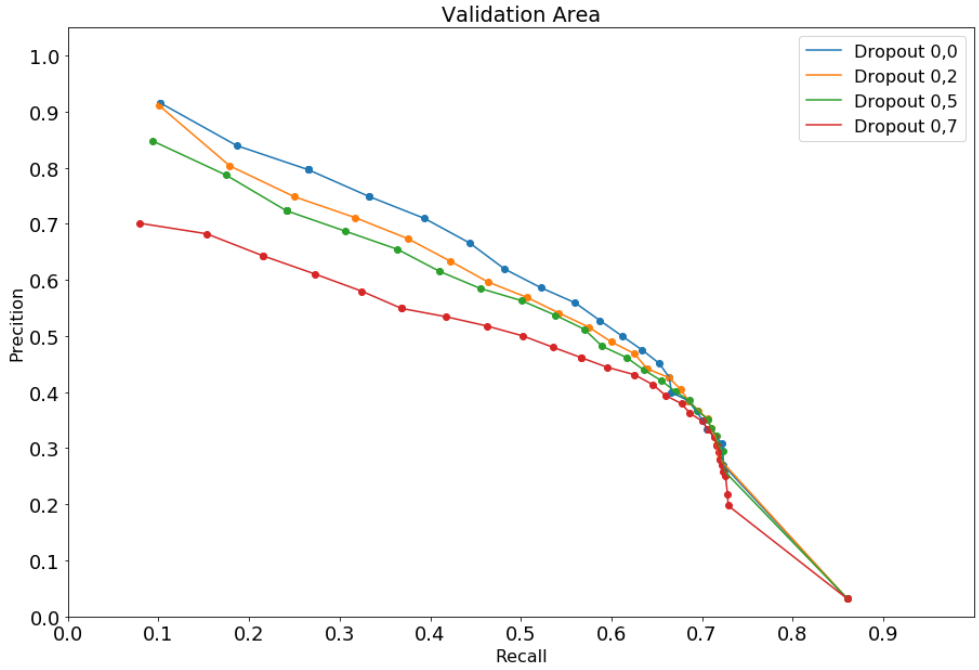

Figure A.7: Effect of the dropout

during the grid search wouldn't modify much the architecture. Indeed, the standard configuration and the tuned ones have really similar performances.

A.2 Node2Vec

For the Node2Vec choice of hyper-parameters we estimated the performances with a Grid Search by starting with the values set as default by the authors and then increased and decreased them. This led to a total number of 729 experiments whose parameters are reported in table A.3. In figure A.8 we can see in green the base architecture compared to the best ones achieved through the exploration of the grid search. The best architecture results in being p=0.01, q=1, d=128, r=20, k=10, l=20. We can see how the performance is not comparable with Logistic Regression or GraphSAGE, suggesting that the surface features are much more important the the role of the nodes inside the network.

Hyper-parameter	Value1	Value2	Value3	Value4
Aggregator	Mean	Mean Pool	Max Pool	**LSTM**
Class Weight	**0.7**			
Max Degree	2	**16**	128	
Samples 1	**5**	25		
Samples 2	**5**	25		
Samples 3	**0**	10		
Batch Size	128	**512**		
Validate Iteration	**5000**			
Validate Batch Size	128	**512**		
Dropout	**0**			
Weight Decay	**0**			
Dimension Embeddings 1	32	**512**		
Dimension Embeddings 2	**32**	512		
Learning Rate	**0.001**	0.01		

Table A.2: Grid for the Parameter Search for GraphSAGE

Parameter	Value1	Default	Value2
p	0.01	1	100
q	0.01	1	100
l	20	80	100
r	2	10	20
k	2	10	20
d	8	128	256

Table A.3: Grid for the Parameter Search for Node2Vec

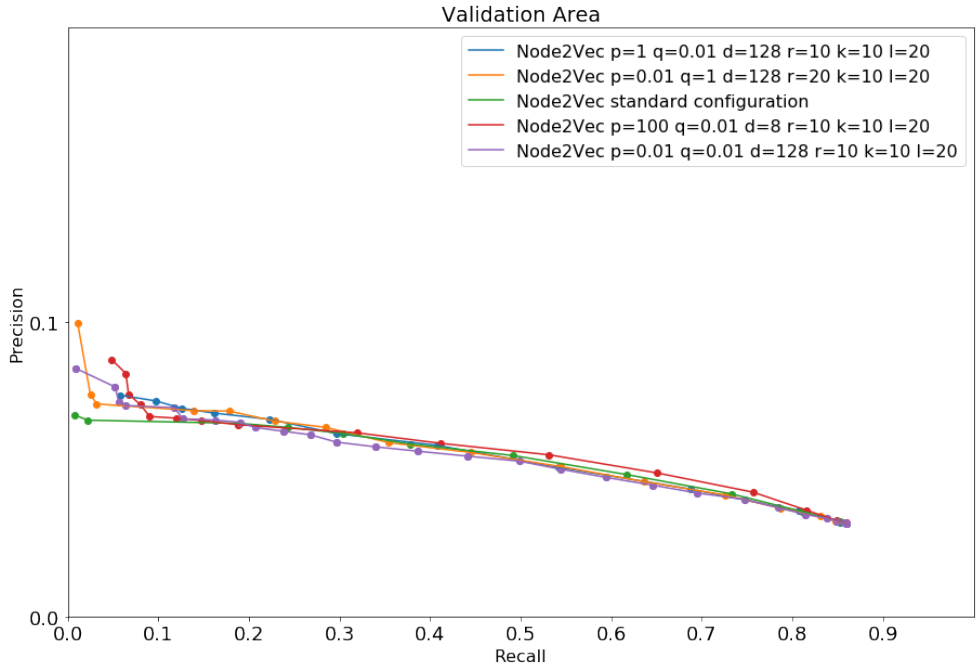

Figure A.8: Comparison of performance achieved with different configurations

Appendix B

Features Selection

In this section we are showing how we evaluated the effect of the features both for GraphSAGE and Logistic Regression.

B.1 Greedy Forward Features Selection

We can see in figures B.1 and B.2 the advancement of the features choice with greedy forward search. We can notice that the curves have a slightly different evolution: Logistic Regression reaches its best performance after adding a few features, while graphSAGE is more sensitive to new features and it exhausts its improvement almost with the entire set of features. It is important to highlight that using all the features instead of the selected subset leads to the same results for both of the models as we can see in Figure B.3.

B.1.1 Importance of the Features

In Figure B.3 we can see the effect of using only elevation with GraphSAGE. Although the other features are derived also from the elevation, it cannot be used alone, leading to worse performances.

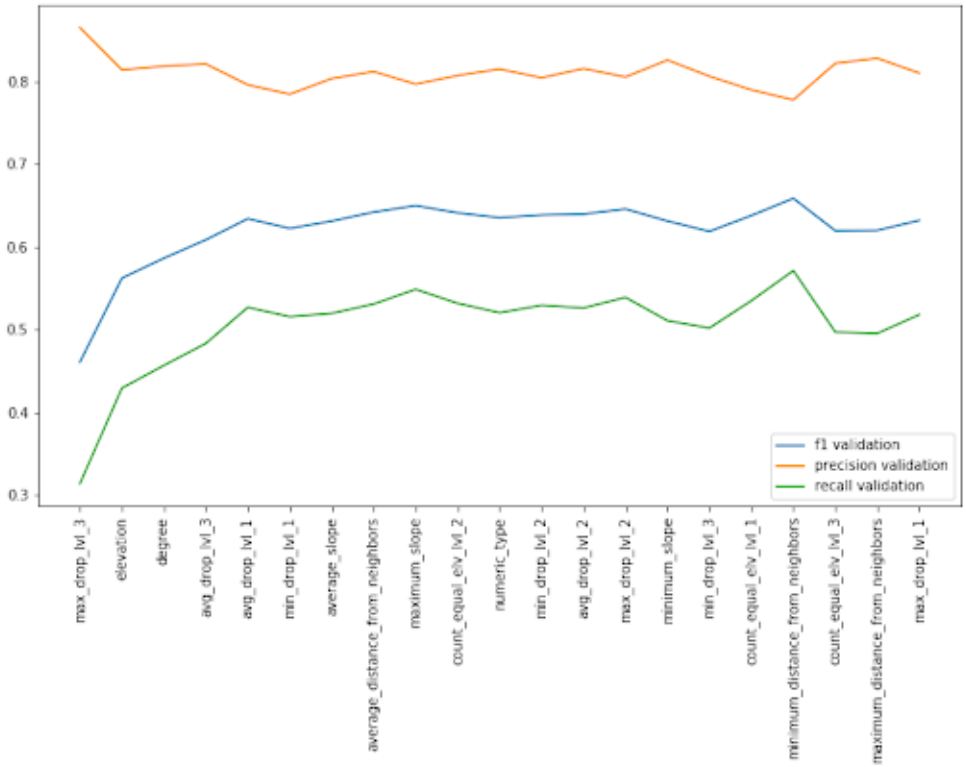

Figure B.1: Greedy Forward Feature selection for GraphSAGE

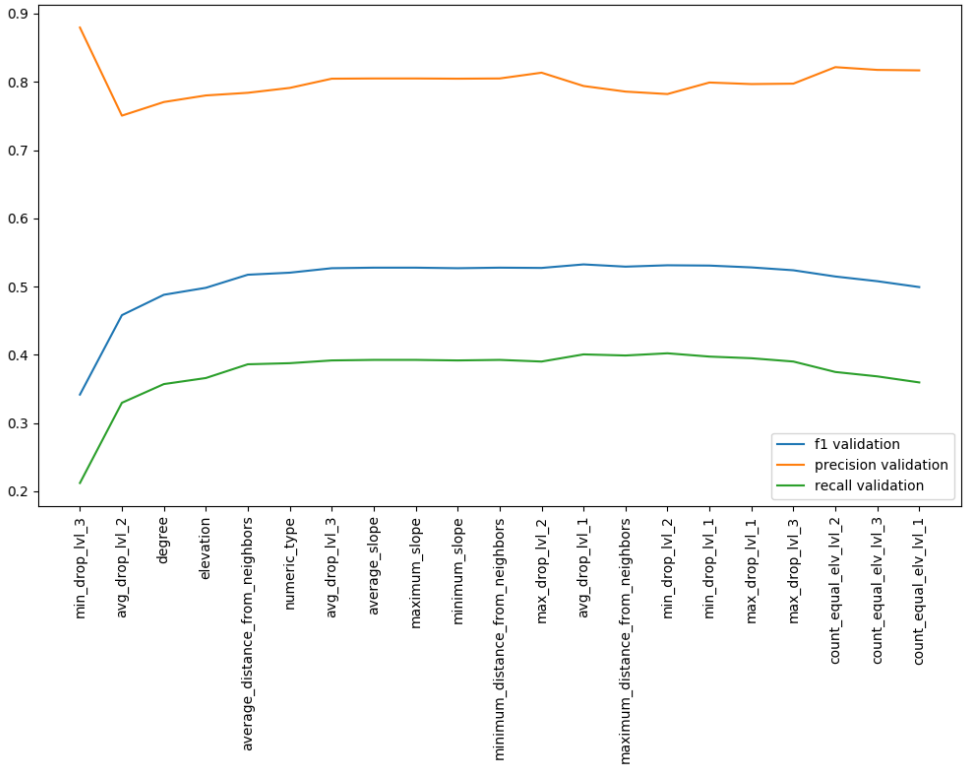

Figure B.2: Greedy Forward Feature selection for Logistic Regression

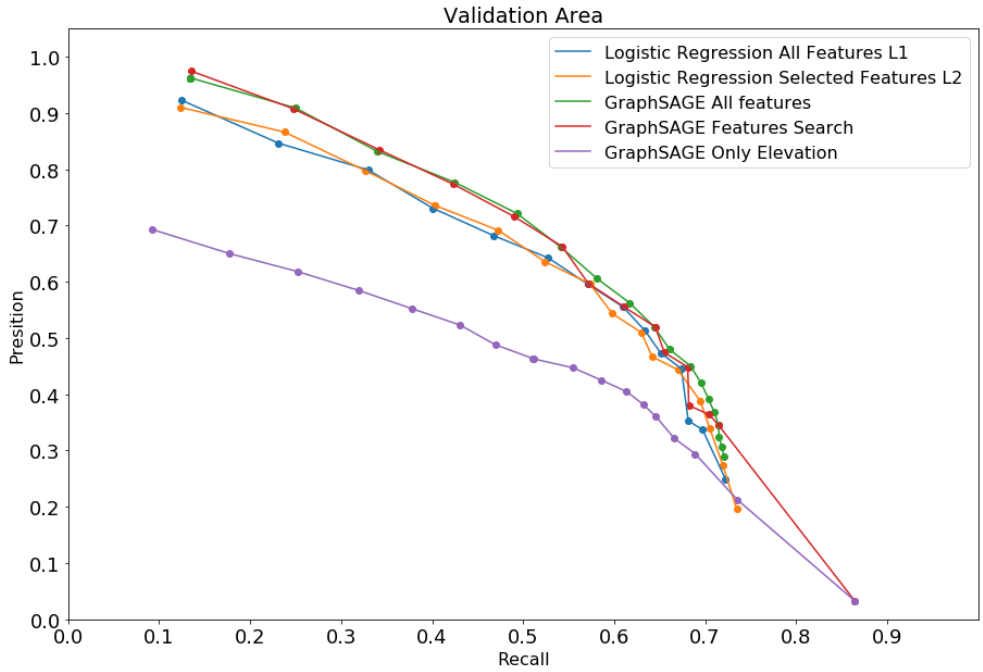

Figure B.3: Effect on the performance between using all the features or a subset