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

Property Booking

A Web Platform Application

Supervisor: Prof. Sara Comai

Master Thesis of:
Mohsen Faghfourmaghrebi, 894263

Academic Year 2020-2021

Here you can put your dedication, like:

To time, that do not go backwards

— A & D & E

Acknowledgments

Here you can put acknowledgements to people that helped you during the thesis. Remember that helping students to write thesis is part of the job of some of them, and they're also paid for that. Please make sure to thank them for what they weren't supposed to do.

Remember also that this page is part of your thesis. I know that your boyfriend/girlfriend is very important to you and you cannot live without her/him, as it is for me. But there's no need to put her/his name here unless she/he gave a proper contribution to this work. Same goes for friends, parents, drinking buddies and so on.

Contents

List of Figures

List of Tables

Sommario

Abstract

Chapter 1

Introduction

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 **??** we will see more classical and heuristic methods of computer science which directly interact with DEMs for the identification of peaks. Section **??**, 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 [**peackerAndDouglas**]. 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 [**lee1992modeling**], 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 moutains peaks identification. The pioneer work [**graff1993automated**] introduces an automated method for classi-

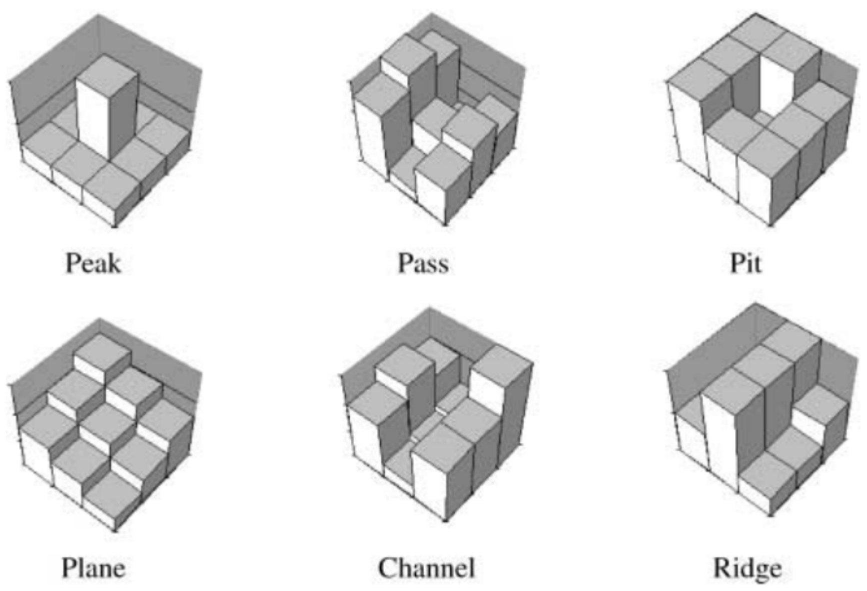


Figure 2.1: Morphometric classes
Figure illustrating the six morphometric classes that can be extracted from a raster DEM. Image courtesy Peter Fisher, Jo Wood and T. Cheng [**fisher2004helvellyn**].

fying generic terrain features extracted from DEMs. Their two-class 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 [**lee1992modeling**] 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 **??**. 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 [**fisherWhatIsAMountain**] 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 [**fisher2004helvellyn**] 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 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 [**wood2009geomorphometry**]. 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 **??** (A) where red denotes a higher value of peakness. These techniques are further explored with a qualitative work in [**fisher2005fuzziness**] 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 [**wood2009geomorphometry**]. These quantities are two tunable parameters. An example can be seen in figure **??** (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 [**schneiderWood**], 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

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

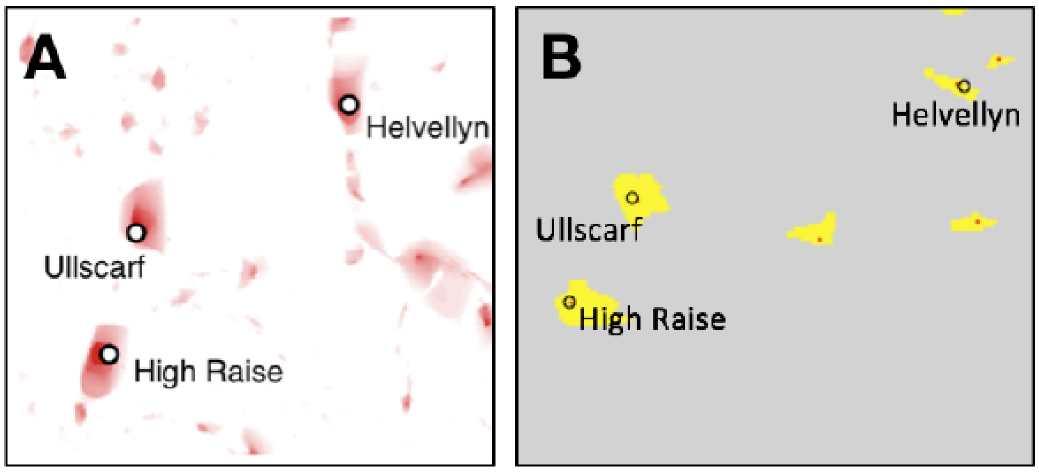


Figure 2.2: Landserf application
(A) Landserf fuzzy feature extraction for peak classification and
(B) Landserf peak classification. Examples in a small area of Lake District
using OS Terrain 50 DEM

author of [**podobnikar2009method**] and [**podobnikar2010mountains**] 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 [**podobnikar2012detecting**] 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 [**fisher2004helvellyn**], in [**deng2008multi**] 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 [**JASIEWICZ2013147**] 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 [**schillaci2**] 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 [**kirmse2017calculating**] 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

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

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.

2.2 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 [**DBLP:journals/corr/Schmidhuber14**]. 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

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

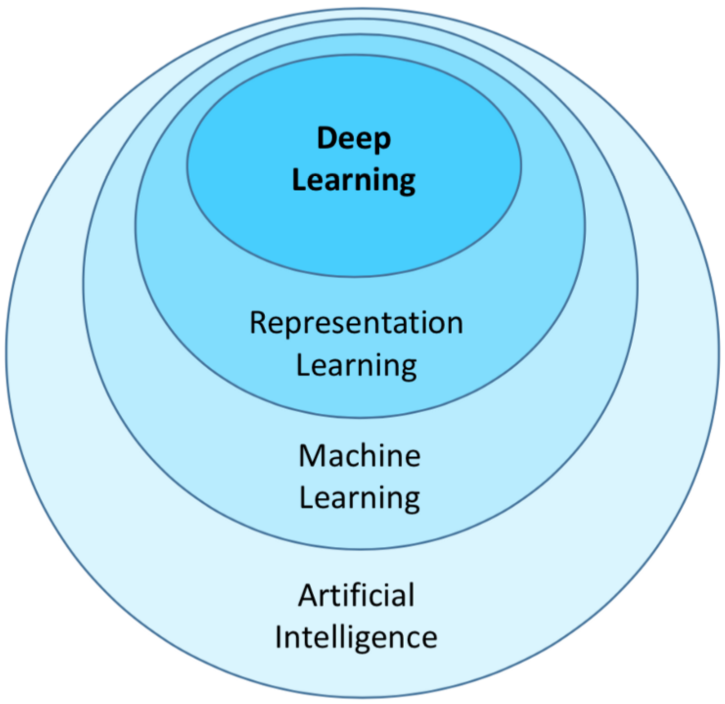


Figure 2.3: Artificial Intelligence fields
Figure showing how Artificial Intelligence can be hierarchically organized in subfields. Deep Learning can be seen as a specific field of Representation Learning, that is by itself a specific field of Machine Learning, a subset of the broader class of Artificial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

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 [**DBLP:journals/corr/Schmidhuber14**]. The organization of the AI fields can be seen in Figure **??**. In this section, we will do an overview of main concepts on these areas.

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 **??**, 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 **??**) and write the output signal y_i as:

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

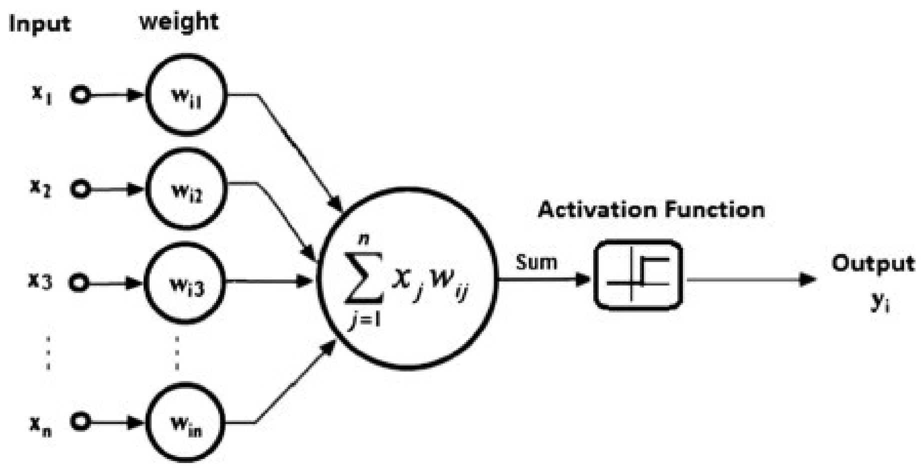


Figure 2.4: Artificial Neuron
Figure taken from [**neuron**] depicting the computational structure and the flow of information between neurons in an Artificial Neural Network.

One of the most well known example of ANN are the Feedforward Neural Networks, also called Multilayer Perceptrons (MLPs). As stated in [**Goodfellow-et-al-2016**], 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 **??** 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 **??**, 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.

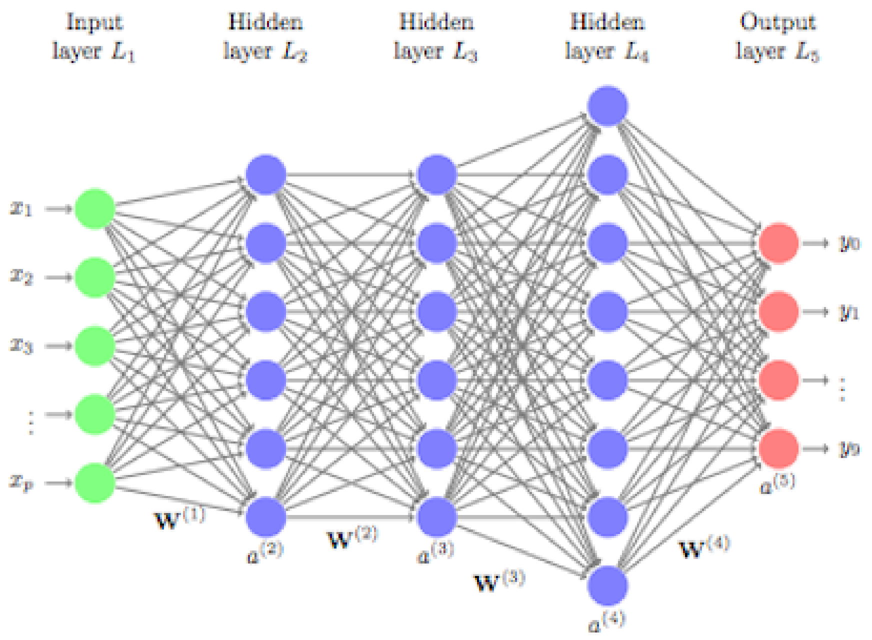


Figure 2.5: A Deep Feedforward Neural Network
Figure taken from [**deep_nn**] showing the computational structure of a Deep (Fully-Connected) Feedforward Neural Network and the Layers that can be distinguished between input layers L_1, hidden layers L_2, L_3,L_4 and output layers L_5.

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 **??**. 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, localiza-

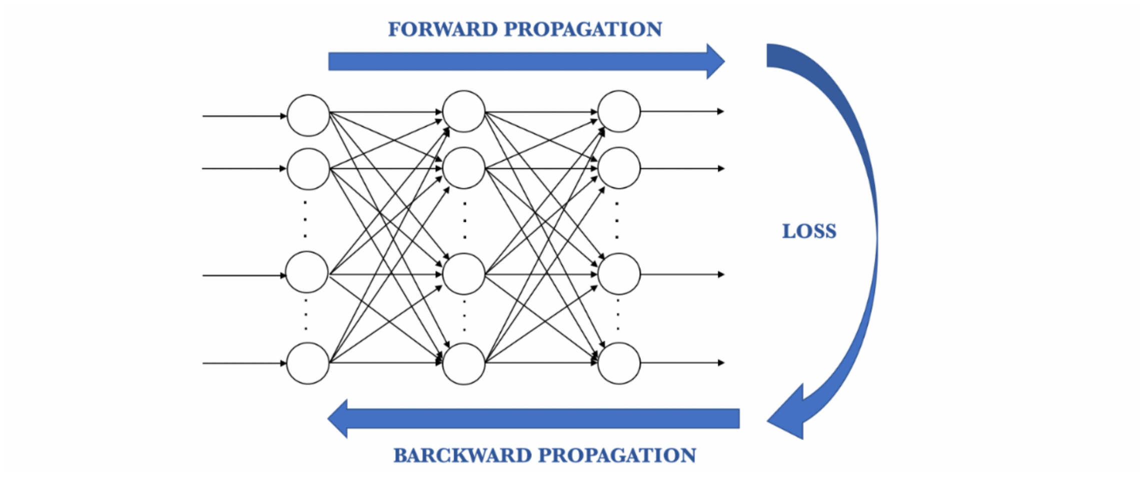


Figure 2.6: Learning phases of a Neural Network
Figure representing the learning phases of an Artificial Neural Network. (1) Forward propagation: the training data is passed across all the neural network and the outcome of the output layers represents the network's prediction. (2) Loss: represents how 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 [**backward_forward_propagation**]

tion and segmentations [**guo2016deep**], 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 [**long2015fully**] they have been used in several works of geoscience and remote sensing as an important tool for the analysis of aerial images. Specifically, in [**audebert2016semantic**] and [**marmanis2016semantic**], 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 [**marmanis2015deep**] 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 [**chen2016convolutional**] 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 [**chen2016convolutional**] 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 [**guerin2017interactive**] and [**beckham2017step**] 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 [**AI3D-DL-PE**] 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. 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) [**trudeau1993introduction**]. 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 **??**.

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 **??** 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

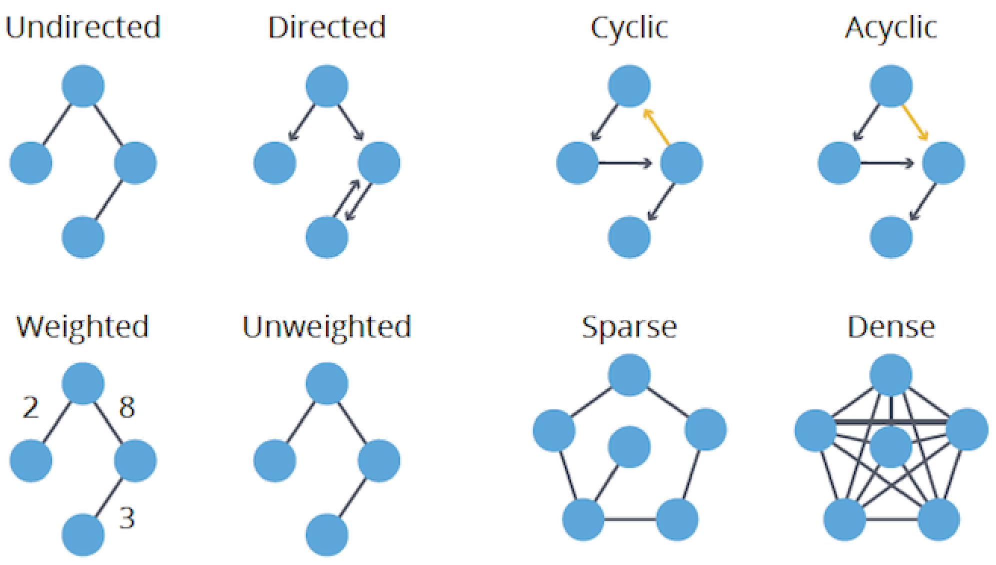


Figure 2.7: Diagrammatic representation of the types of graphs
The blue circles represent the vertexes (or the nodes), while the gray lines represent the connections among the nodes, i. e. the edges. The figure shows how can be defined 8 categories of graphs based on the type of edges.

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 **??** 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 [**surveyGraphEmbedding**]. In particular, the utilization of Machine Learning on graphs became an important and ubiquitus task with applications ranging on very heterogeneous fields [**representationLearning**].

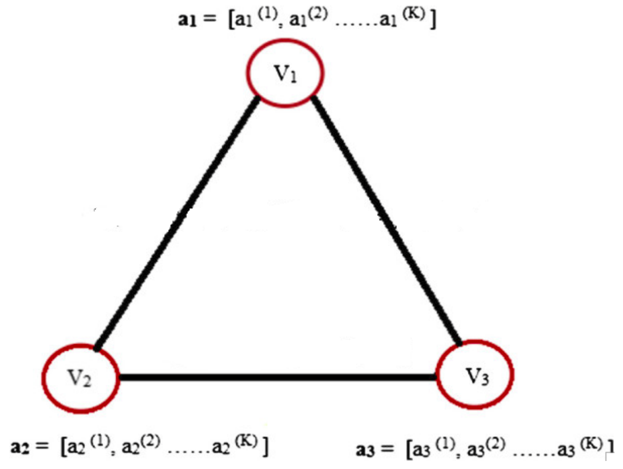

$\mathbf{a_1} = [a_1^{(1)}, a_1^{(2)} \ldots\ldots a_1^{(K)}]$
V_1
V_2
V_3
$\mathbf{a_2} = [a_2^{(1)}, a_2^{(2)} \ldots\ldots a_2^{(K)}]$
$\mathbf{a_3} = [a_3^{(1)}, a_3^{(2)} \ldots\ldots a_3^{(K)}]$

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 $a_i^{(K)}$.

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 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 **??** (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

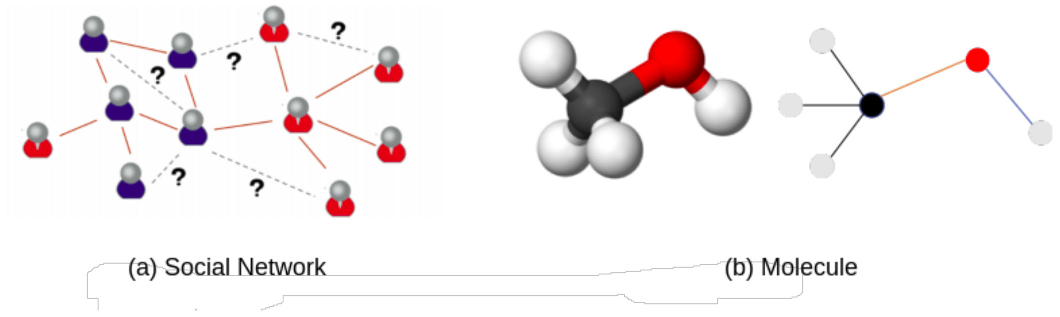


Figure 2.9: Graph models of reality
Abstract representation two real world situations suitable for graphs. (a) Social Network instance with the members depicted as nodes and their relationships as edges. (b) Molecule representation where the nodes are the atoms and the edges are the molecular bonding.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

Figure **??** (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 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 [**surveyGraphEmbedding**]. 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 [**representationLearning**]. 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 [**spectral_networks_local_connected_networks**]. 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 [**cayley_nets**]. In their work, the authors of [**spectral_networks_local_connected_netwo**
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 [**surveyGraphEmbedding**], the problem of graph embedding is related to two traditional research problems: graph analytics and representation learning. Particularly, 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 [**representationLearning**]. In their survey the authors of [**surveyGraphEmbedding**] show a graph embedding taxonomy based on the problem setting and on the used technique. This distinction is showed in Figure **??**.

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 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 **??**). 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

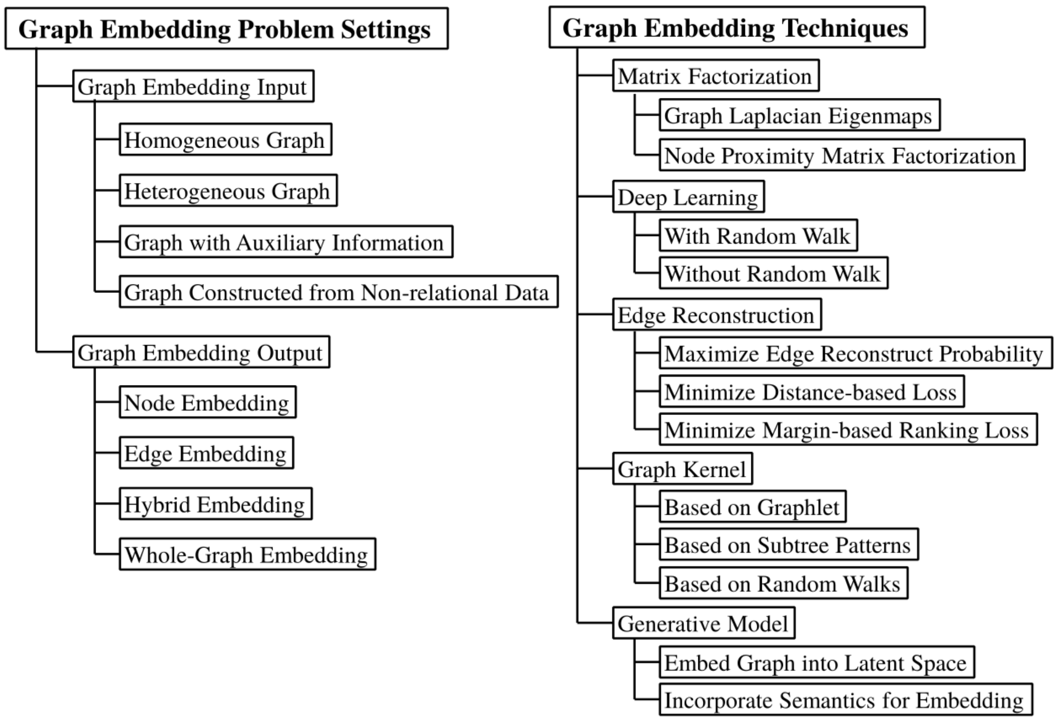

Graph Embedding Problem Settings
Graph Embedding Input
Homogeneous Graph
Heterogeneous Graph
Graph with Auxiliary Information
Graph Constructed from Non-relational Data
Graph Embedding Output
Node Embedding
Edge Embedding
Hybrid Embedding
Whole-Graph Embedding
Graph Embedding Techniques
Matrix Factorization
Graph Laplacian Eigenmaps
Node Proximity Matrix Factorization
Deep Learning
With Random Walk
Without Random Walk
Edge Reconstruction
Maximize Edge Reconstruct Probability
Minimize Distance-based Loss
Minimize Margin-based Ranking Loss
Graph Kernel
Based on Graphlet
Based on Subtree Patterns
Based on Random Walks
Generative Model
Embed Graph into Latent Space
Incorporate Semantics for Embedding

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

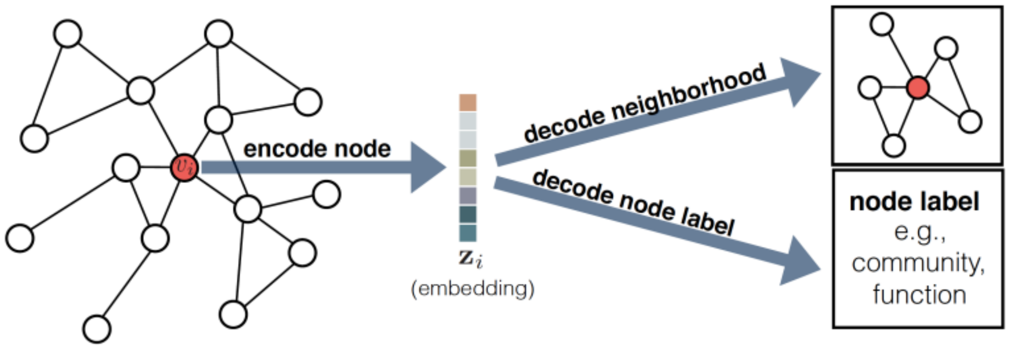
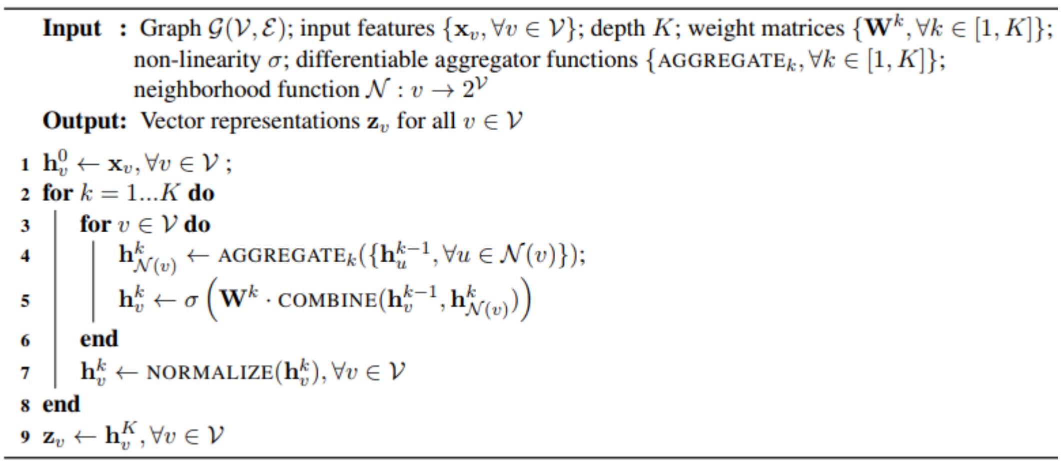


Figure 2.11: Encoder-decoder approach
Figure taken from [**representationLearning**] representing an overview of the encoder-decoder approach. First, the encoder maps the node, v_i, to a low-dimensional vector embedding, $\mathbf{z}_i$, based on the node's attributes and/or local neighborhood structure. 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.

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 [**deepWalk**] and node2vec [**node2vec**], 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 [**performanceRandomWalk**]. 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) [**DNGR**] and Structural Deep Network Embeddings (SDNE) [**SDNE**] 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 informations 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 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 [**kipf_semi_supervised**]. During the encoding phase the neighborhood aggregation methods build up the representation in an iterative/recursive fashion. As showed in [**representationLearning**] the procedure can be represented with an algorithm whose pseoudocode can be seen in Figure **??**. 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



```
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 $\sigma$; 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 $v \in \mathcal{V}$
1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for $k = 1...K$ do
3     for $v \in \mathcal{V}$ do
4         $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5         $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6     end
7     $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$
8 end
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

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

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 **??**. Graph Graph Convolutional Networks (GCN) [**kipf_semi_supervised**], column networks [**column_networks**] and GraphSAGE [**graphSAGE**] all follow the neighborhood aggregation principle but differ primarily 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 [**graphSAGE**]. 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 **??**) 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 [**molecular_fingerprints**] 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 **??**. 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. [**deferrard**] and Bruna et al. [**spectral_networks_local_connected_networks**], 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 [**deferrard**] introduce an approximation of the encoders by using the Chebyshev polynomials. However, as stated in [**representationLearning**], the introduced approximations make the graph coarsening methods conceptually similar to the algorithm in Figure **??**.

Regarding the taxonomy showed in the survey [**surveyGraphEmbedding**] 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 **??** 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 [**graphSAGE**] 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 [**deferrard**] 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 [**representationLearning**], 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 [**deferrard**], are called spectral models. As stated in [**monet**] 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 [**monet**] 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 nodes. Still in the context of spectral methods the authors of [**transfer_learning_on_graphs**] 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 [**surveyGraphNeuralNetworks**] proposed a categorization of the deep learning methods on graphs that can be seen in Figure **??**. We worked mainly with the unsupervised deep learning methods,specifically with graph auto-encoders such as *node2vec* [**node2vec**], and with the semi-supervised graph convolutional networks such as *graphSAGE* [**graphSAGE**] and the convolutional neural networks in the spectral domain of [**deferrard**]. It is important to highlight that in our work the algorithms which are categorized as semi-supervised in [**surveyGraphNeuralNetworks**] 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 **??** 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).

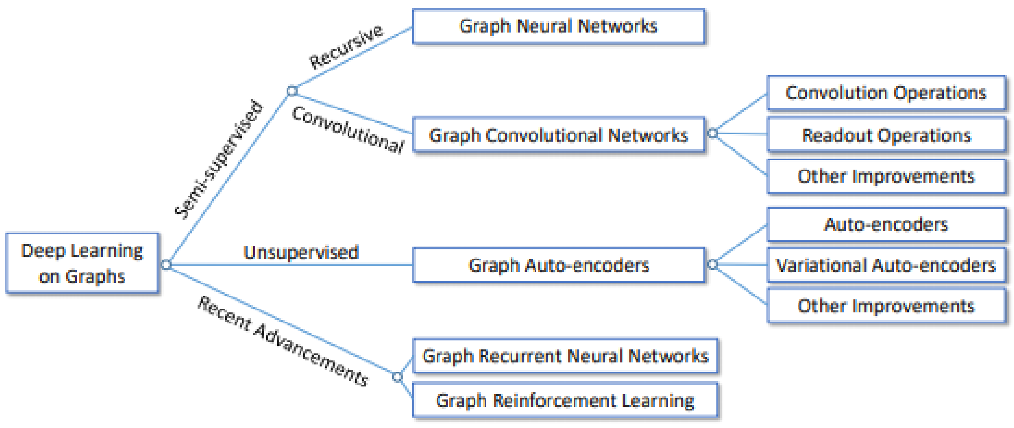


Figure 2.13: Categorization of deep learning methods on graphs
Figure taken from [**surveyGraphNeuralNetworks**]. The authors divide the existing methods into three categories: semi-supervised, unsupervised and recent advancements. The semi-supervised methods can be further divided into Graph Neural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

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-

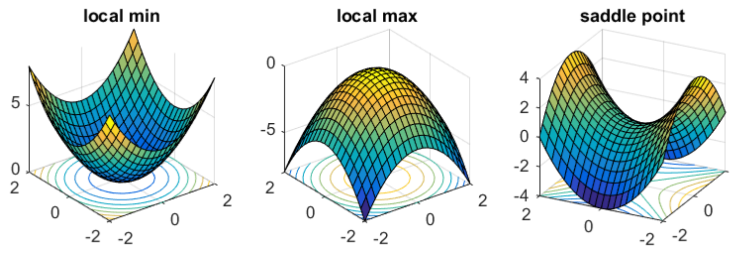


Figure 3.1: Critical Points
Figure illustrating the 3 types of critical points of a continuous surface.

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 **??**.

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) [**surface_networks_rana**]. An example of surface network for a portion of the Latschur Mountains in Western Carinthia, Austria can be seen in Figure **??**. 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.

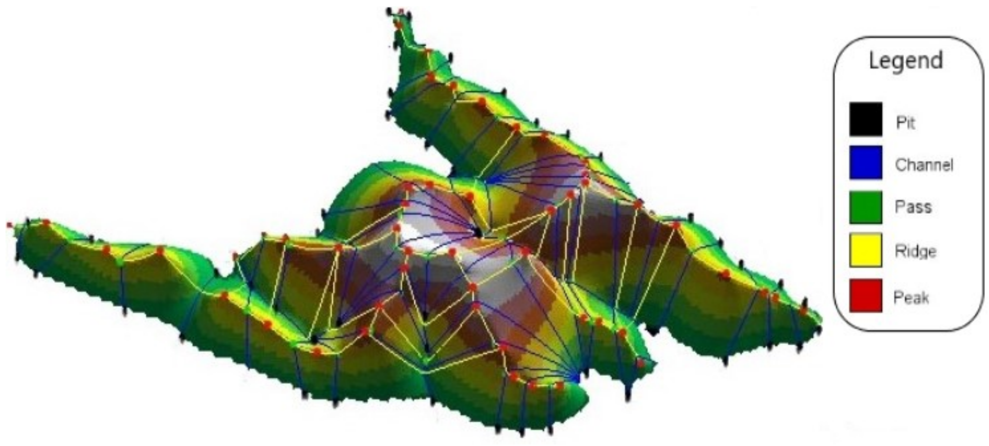
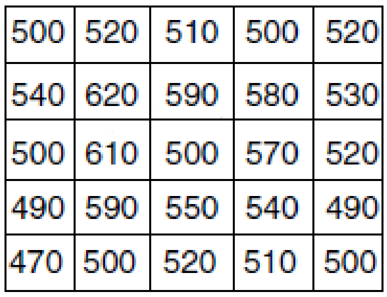


Figure 3.2: Surface Network
Figure illustrating a surface network. The critical points are evidenced with red for peaks (maxima), black for pits (minima) and green for passes (saddles). In morphology the connections between saddles and maxima points are called ridges and here represented with a yellow 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 [**surface_networks_rana**].



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 [**Farr2007RGP**]. 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 **??**.

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 **??**.

The method of Shigeo Takahashi [**extracting_surface_topology**] 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 [**extracting_surface_topology**] 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 [**peackerAndDouglas**]. 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 **??** 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 [**wood_book**] 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 crit-

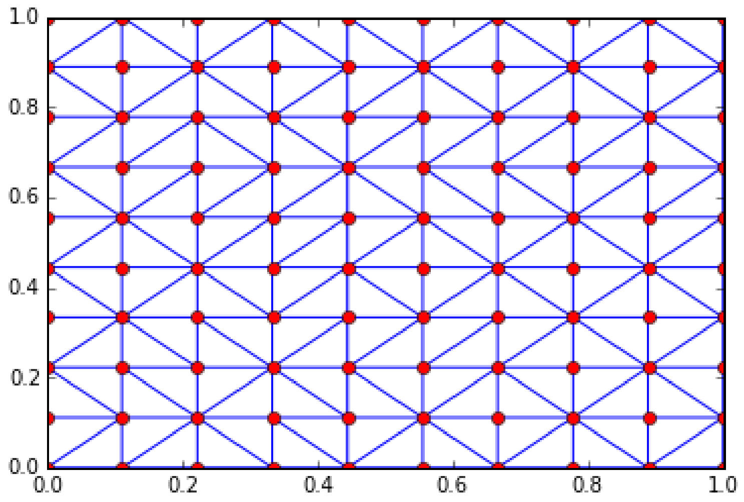


Figure 3.4: Delaunay Triangulation
Figure illustrating a portion of DEM and the Delaunay triangulation. The red dots represents the cells of the raster while the blu lines how the triangulation is setting the neighbours of each cell.

ical lines. The great benefit of the bi-variate quadratic surfaces is that they can 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 [**Schneider**] 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

3.2.1 Logistic Regression

3.2.2 Node2Vec

3.2.3 GraphSAGE

...others...

Chapter 4

Learning to find mountains

4.1 Methods under evaluation

Prominence, Isolation, Peak Clasification, Landserf Fuzzy Clasification, Unet,

4.2 Input Data

In Chapter **??** 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 [**Farr2007RGP**]. 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 **??** 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 the SRTM1 DEMs with resolution of 1 arch-second, that in areas realtively far from the poles correspond approximately to 30m, i.e. we have a measurement of elevation every 30m. 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. 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 **??** for building surface networks from DEMs we opted for the algorithm of Shigeo Takahashi [**extracting_surface_topology**]. 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, involving the invalidity for the formula. The proposed algorithm in [**extracting_surface_topology**] 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 those points that are adjacent to P in the triangulation. An example referred to the DEM raster in Figure **??** can be seen in Figure **??**.

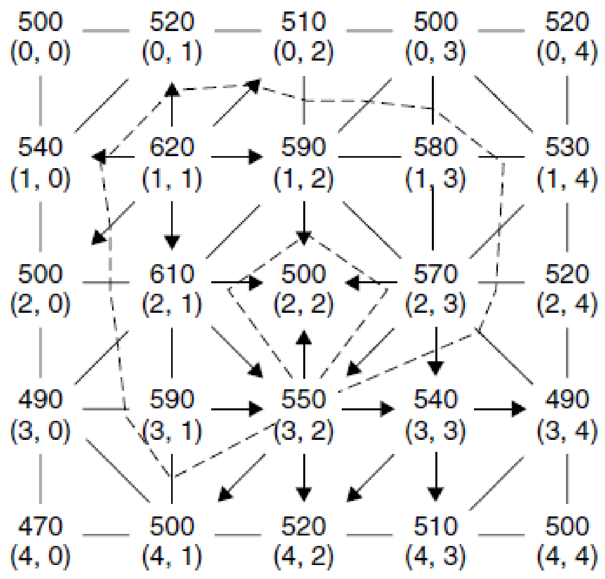


Figure 4.1: Triangulated DEM
Figure illustrating how the neighborhood is established between the cells.
Image courtesy Shigeo Takahashi [**extracting_surface_topology**].

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 pos-

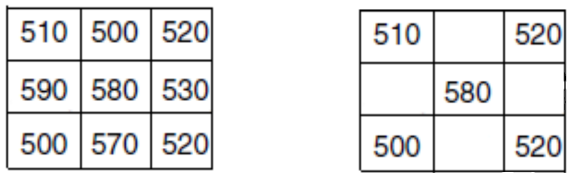


Figure 4.2: Incorrect classification of cell with triangulation
Figure illustrating how the cell with elevation 580, which is not belonging to any morphological class, would be added to the set of maxima in case of the illustrated type of triangulation.

sible 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 **??**.

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

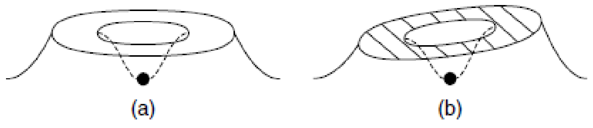


Figure 4.3: Level region
A level region: (a) a level region surrounding a pit and (b) the effect of introducing a second ordering. Image courtesy S. Takahashi

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 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

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 **??** 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 **??**(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 reviewed 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 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 we saw above 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:

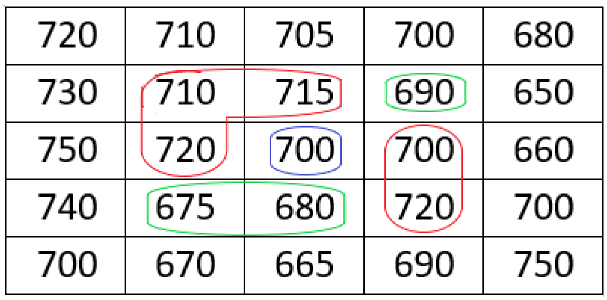
pass $|\Delta_+| + |\Delta_-| > 0, \qquad N_c = 2 + 2m(m = 1, 2, ...)$

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 criteria we described before. 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 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

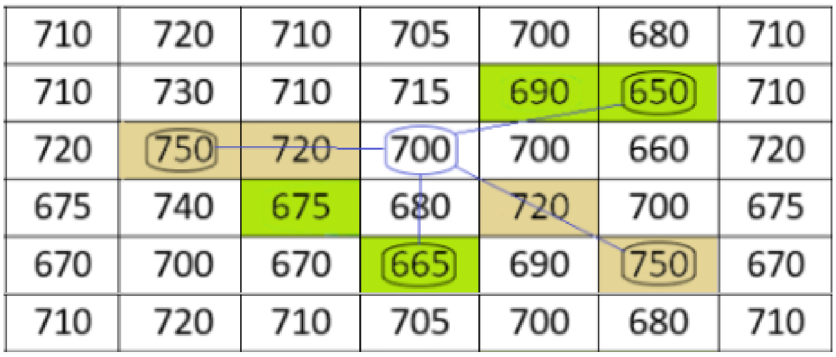


720	710	705	700	680
730	710	715	690	650
750	720	700	700	660
740	675	680	720	700
700	670	665	690	750

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.

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 **??**.

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 maxima found previously. For simplicity suppose that the cell is already the cell of a maxima. Similar procedure is applied for finding the connected



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

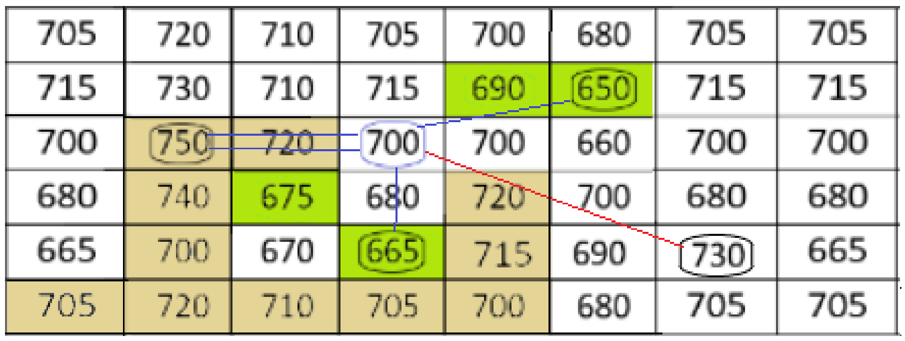
Figure 4.5: Paths and connections to the critical points
This figure shows in brown the ridges, i.e. the paths from the saddle (circled in blue) to the maxima and in green the channles, i.e. the paths to the minima. The blue lines represent the edges in the graph.

minima. Figure **??** 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 **??**, 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 recurrent 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.



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

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.

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* [**grover2016node2vec**], 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:

$$s_{\text{ij}} = \Delta_{\text{dist}_{\text{ij}}} / \mid \Delta_{\text{elev}_{\text{ij}}} \mid \tag{4.1}$$

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:

$$\Delta_{\text{dist}_{\text{ij}}} = R \cdot c \tag{4.2}$$

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

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)}) \tag{4.3}$$

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:

$$atan2(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \land y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \land y < 0 \\ +\frac{\pi}{2} & \text{if } x = 0 \land y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \land y < 0 \\ \text{undefined} & \text{if } x = 0 \land y = 0 \end{cases} \tag{4.4}$$

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

$$a = sin^2(\Delta\phi/2) + cos(\phi_i) \cdot cos(\phi_j) \cdot sin^2(\Delta\lambda/2) \tag{4.5}$$

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* [**graphSAGE**], 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 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 **??** but without the absolute value when computing the difference of elevation:

  $$s_{\mathrm{ij}} = \Delta_{\mathrm{dist_{ij}}} / \Delta_{\mathrm{elev_{ij}}} \tag{4.6}$$

  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 **??** 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 **??** 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

4.4 Ground Truth

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 mostly 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 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, are 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 not all the maxima are "real peaks", but it is not also really rare that the locations of the "real mountains" are tens of meters close to some node of the graph that corresponds to a maxima or, less often, coincident.

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 [**FrenayV14**] 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 lower than 80m. We considered them potentially the same mountain summit

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

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

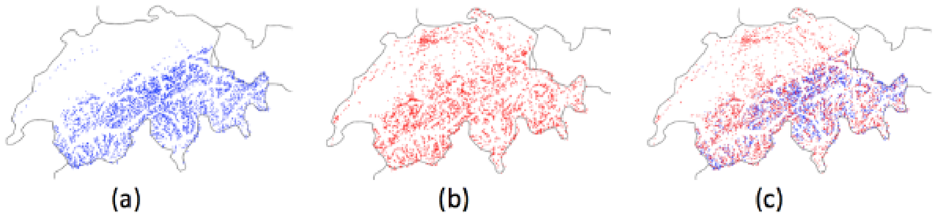


Figure 4.7: Mountain peaks distribution
(a) contains the peaks from SwissNames3D, (b) peaks from OpenStreetMap and (c) their combination in the Switzerland territory. Image courtesy [**AI3D-DL-PE**]

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 [**AI3D-DL-PE**]. Their distribution can be seen in Figure **??**.

4.5 Label assignement to nodes

4.6 Graphs subsampling

4.7 Pre-processing of the input

4.8 Development of Graph learning-based methods

4.9 Dataset

4.10 Evaluation procedure

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 overwhelmingly superior to those of the other types: the input DEM files are matrices of 3601x3601 pixels, of which 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 [**saito2015precision**] for scenarios with highly unbalanced classes.

Chapter 5

Evaluation

5.1 Quantitative analysis

5.2 Qualitative analysis

5.3 Generalization analysis