Politecnico di Torino

Master of Science Degree in Mechatronics

Engineering

Master of Science Degree Thesis

An Artificial Intelligent-based strategy for Sheltering Factor computation

Supervisors
Prof. Giorgio GUGLIERI
Dr. Stefano PRIMATESTA

Candidate
Angelo SCIACCA

April 2021

Summary

Usage of Unmanned Aircraft Systems (UASs) is widely diffused and will involve different aspects of our life in cities in the future. Nowadays UASs are used to accomplish several applications: data collections, surveillance, logistics, search and rescue, to name a few.

Despite the rapid growth of these technologies, currently most of the National Aviation Authorities impose strict regulations to the UAS missions practically hindering autonomous flights. Most of these operations must be performed in Visual Line-Of-Sight (VLOS) and, to guarantee operations over urban area, they require the use of harmless drones, even if in the recent years the authorities permit to perform operations also Beyond Visual Line-Of-Sight (BVLOS) even over cities.

This thesis is based on a solid preliminary work done by the research group in which a risk-aware path planning strategy for UASs has been developed to determine a safe flight mission evaluating the risk caused by the UAS of flying over populated areas. Specifically, the risk is estimated by generating a risk-map and considering several parameters, such as the population density and the sheltering factor. The objective of this thesis is to provide a valid strategy using a Deep Neural Network to estimate the Sheltering Factor of a specific area, these values will be used as inputs for the Risk-Map definition. The results will be then compared with the Sheltering Factor values obtained from the definition of obstacles layer considering the model of the city of Turin from OpenStreetMap (OSM). This research activity is involved within the projects of the Amazon Research Award “From Shortest to Safest Path Navigation: An AI-Powered Framework for Risk-Aware Autonomous Navigation of UASs".

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Analysts estimate an increase in the use of UASs in the coming years. One of the reasons for this growth is their versatility, and therefore their use in different areas. It is inconceivable how the presence in the skies of these systems represents a danger to the underlying population. The current regulation in force limits the operability of UASs in urban areas, so it is necessary to amend this regulation, for the full exploitation of UASs in these areas. The importance of a *risk-based* approach to safe UAS flight is recognized by the European Aviation Safety Agency and the Federal Aviation Administration[1].

The thesis work to be discussed is founded on a much larger project, related to the activities of the Amazon Research Award "*From shortest to safest path navigation: an AI-powered framework for risk-aware autonomous navigation of UASs*", granted to prof. Alessandro Rizzo and in collaboration with prof. Giorgio Guglieri and Dr. Stefano Primatesta. The aim of this project is to evaluate the risk for lightweight UASs in urban environment, to achieve a risk-aware navigation over safe routes. For the correct analysis and assignment of risk, a *risk map* has been built, over which a spatially resolved *risk factors* are computed. Specifically, the risk is defined as the hourly probability of a fatal accident. The *risk factor* is computed through a multi-layer formalization, using *Artificial Intelligence* (AI) *techniques* over public domain data. The knowledge of the risk factor facilitates the development of path planning strategies that optimize the overall mission risk rather than the flight time or course length. As *risk factors* along the route can change, risk maps will be dynamically updated during flight, and the planned trajectory will be rapidly updated to keep pursuing the safest route[2]. The overall proposed project presented in the paper "*From shortest to safest path navigation: an AI-powered framework for risk-aware autonomous navigation of UASs*", is reported in the following image.

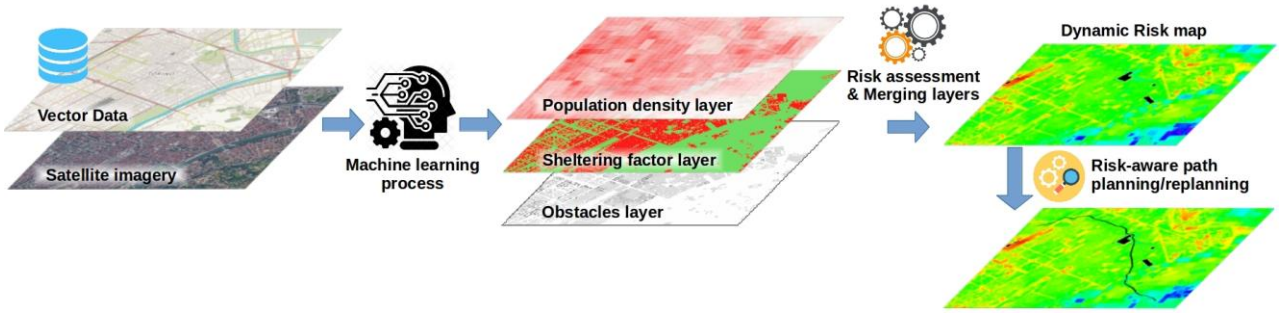

Figure 1: Graphical representation of the project.

1.1 RISK ASSESSMENT DESCRIPTION

In a preliminary work of the research team, a probabilistic risk assessment was introduced, based on the construction of a *risk map*.

Risk map is a two-dimensional location-based map quantifying the risk to the people on the ground for each cell in the map, and it represents a common tool used for the assessment field. In the *Figure 2* a representation of a *risk map* and its associated notations, are reported.

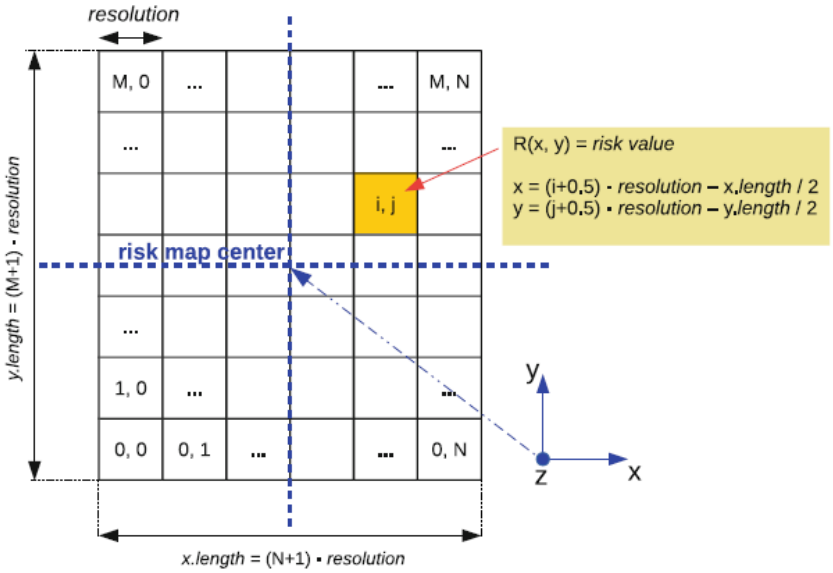

Figure 2: Representation of the risk map and its notation.

The total mission risk is computed adding the risk factor value of each elements, of the *risk map*, along the covered path. The *risk map* consists of several layers, each of which is a location-based map, including useful data of homogeneous nature. These layers have the same characteristics and dimensions as the *risk map*.

The layers considered in [3] are:

- *Population density layer*: defines the population density and distribution in the area interested by the risk map;
- *Obstacles layer*: defines the height of obstacles in the map;
- *Sheltering layer*: defines the sheltering level of each cell map;
- No-fly zone layer: defines in which zones the flight is not allowed.

For a complete risk assessment, several factors concerning the characteristics of the drone and the characteristics of the environment were considered, as well as uncertainties in parameters and wind. The risk is computed using various descent event types, such as ballistic descent, uncontrolled glide, parachute descent, and fly-away. It includes also no-fly zones and obstacles at the flight altitude [3].
For the generation of the *risk map*, two main steps must be performed. First, with the risk assessment, 4 events maps are generated, one for each descent event considered. The inputs for risk assessment are the sheltering factor layers and the population density, as well as drone parameters and environment characteristics. In the last step the 4 different events maps are individually combined with no-fly zone layer and obstacle layer, to obtain the final *risk map*. The main architecture of the *risk map* generation is shown in *Figure 3*.

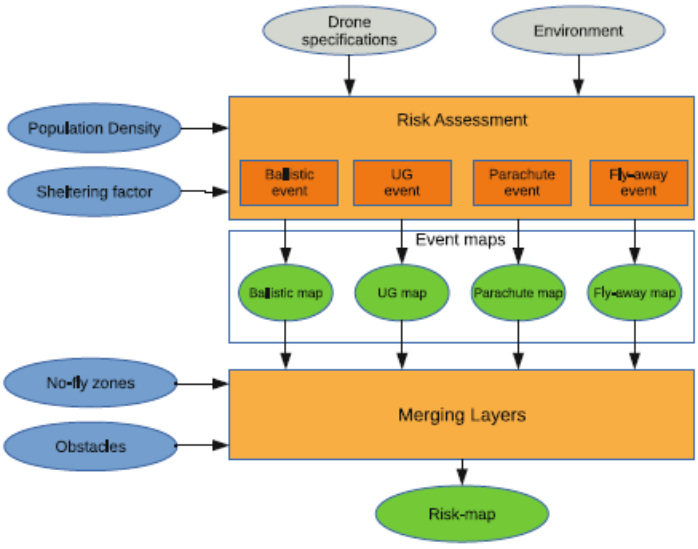

Figure 3: Architecture of the risk map generation.

The *risk map* resulting, consists of several tiles each of which corresponds to a square area of $100m \times 100m$. The risk is represented on a colour scale that goes from fuchsia which corresponds to the minimum value 0 to red which represents the maximum value, the latter does not have a default value but depends on the scenario considered. In *Figure 4* an example of *risk map* is shown, where the maximum value is equal to $8 \times 10^{-6}\, h^{-1}$.

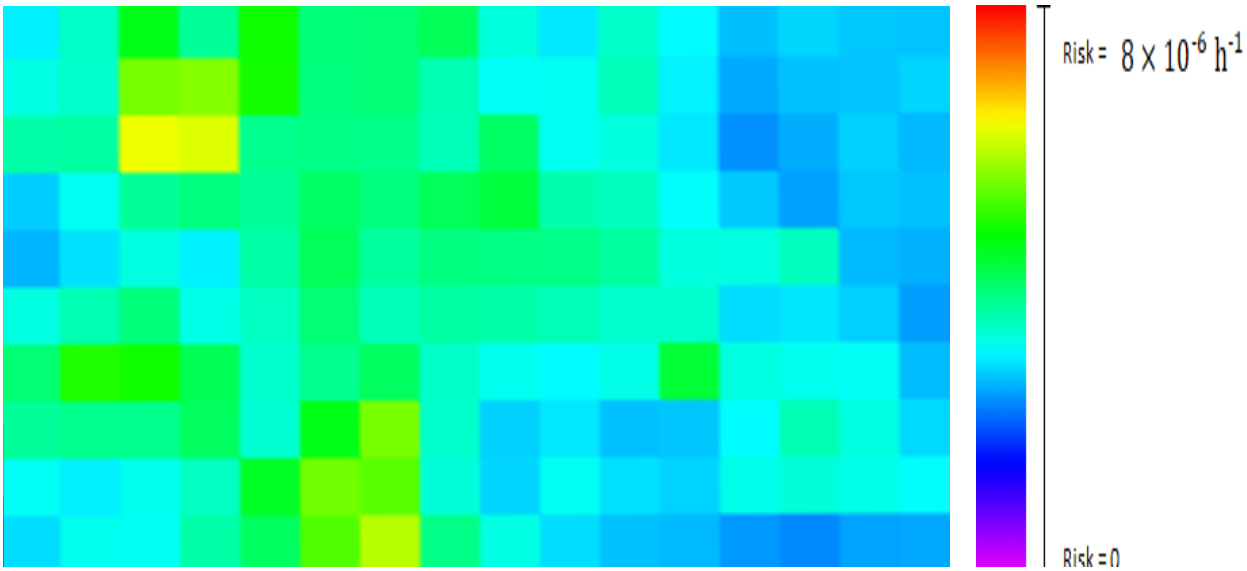

Figure 4: Risk map with colour scale example.

The probabilistic risk assessment, introduced in the first work of the research team, defines the *risk value* as the hourly probability in time to have a casualty, due to a sequence of three conditional events:

- The loss of control of the vehicle with uncontrolled crash on ground;
- The impact with at least a person after the crush;
- The impact results in a fatality.

Therefore, the probability to have a casualty is defined in the following way:

$$P_{casualty}(x,y) = P_{event} \times P_{impact}(x,y) \times P_{fatality}(x,y)$$

The first term of this equation indicates the probability that the UAS will lose control with the consequent uncontrolled descent with crush on ground. Different values of P_{event} for each descent event and the independence between failure are assumed. The second term of the equation, P_{impact}, represents the probability of bumping a person when the UAS crashes on ground after an uncontrolled descent. This value is function of the population density and the area subject to crush. To conclude,

$P_{Fatality}$ is the probability to produce fatal injuries after a person has been affected, this term is a function of the kinetic energy of the UAS at the impact moment and of the *sheltering factor*. The use of this last factor is of major importance for the calculation of the $P_{Fatality}$, as it indicates the presence of obstacles in the impact area that reduce the kinetic energy of the UAS to impact. As can be seen in the following equation, the probability of the fatality may decrease significantly as the *sheltering factor* increases:

$$P_{fatality}(x,y) = \frac{1-k}{1-2k+\sqrt{\frac{\alpha}{\beta}}\left[\frac{\beta}{E\,[E_{imp}(x,y)]}\right]^{\frac{3}{E[S(x,y)]}}}$$

with

$$k = \left(\left[\frac{\beta}{E\,[E_{imp}(x,y)]}\right]^{\frac{3}{E[S(x,y)]}}\right)$$

where:

- S sheltering factor;
- E_{imp} kinetic energy at impact;
- α impact energy needed to cause a fatality probability of 50% when S=6;
- β impact energy needed to cause fatality when S approaches to zero.

The next paragraphs will define *sheltering factor* and describe how that factor was calculated in the previous work [3] and how it is computed using the proposed strategy described in this work. The computation of this factor represents the core of the thesis.

1.2 SHELTERING FACTOR ESTIMATION STATE OF THE ART

The *sheltering factor* is an absolute real number between zero to infinite, indicating the level of shelter provided to the people in the area by buildings, trees, cables, and any other element capable of reducing the kinetic energy of the UAS at the impact. High *sheltering factor* values correspond to lower risk factor values. In [3] the value of the *sheltering factor* is defined in a fixed range, specially from 0 to 10, where 0 corresponds to an area without shelter and 10 to an area with the maximum sheltering factor. An example of areas with low and high sheltering factors are shown in *Figures 5* and *6*, respectively.

Figure 5: Aerial image of a typical area with a very low Sheltering Factor.

Figure 6: Aerial image of a typical area with high Sheltering Factor.

Sheltering factor value for a given area is contained within the sheltering factor layer, which is a location-based map represented with a matrix S, where the individual elements of the matrix are $S(x, y)$. In the work introduced in [3] the sheltering factor was computed using the OpenStreetMap[4] data, from those data a tridimensional model of the city can be extracted and the *Sheltering Factor* layer is generated, it is grayscale; white is 0 and black is 10.

The sheltering factor layer in the *Figure 7,* consists of a series of tiles each representing an area of size $100m \times 100m$, an average *sheltering factor* value has been assigned to each surface that varies in a range between 3 and 8.

Figure 7: Sheltering Factor layer example.

The limited presence of available public data, their inaccuracy and incompleteness, and the variability of the granularity and resolution represent the limits of the method proposed in [2].

1.3 CURRENT WORK

In this thesis work, an AI-based strategy is proposed to systematically construct the sheltering factor layer using the available data. The sheltering factor layer will be estimated using satellite images fed to pre-trained deep neural network. Image segmentation tool, is used in this case to obtain more accurate sheltering value, compared to those obtained using the method discussed earlier.

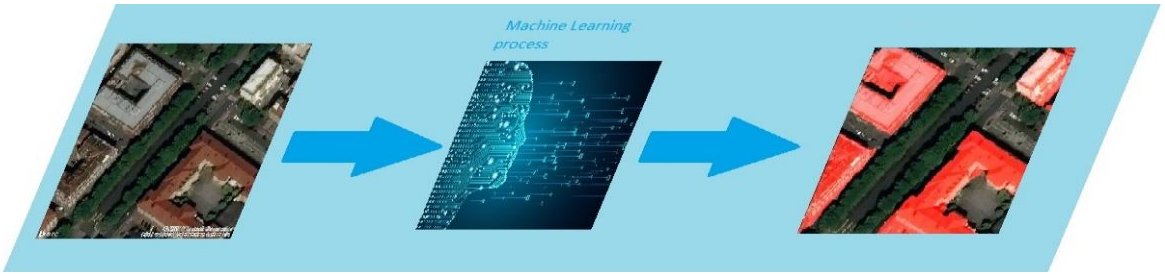

Figure 8: Workflow of the proposed AI-based strategy.

In *Figure 8* is described in a very simply manner the overall mechanism used to obtain the average sheltering factor of a particular area. The model implemented will be tested using a certain number of images of the city of Turin, it represents the ideal testbed for the proposed AI-based strategy.

1.4 STRUCTURE OF THE THESIS

The thesis is organised into five chapters, each of them has the aim to provide an explanation of all the topics examined during the development of the work. Each individual chapter has the task of shedding light on the different aspects of AI-based proposed strategy, whether practical or purely theoretical. Chapter 1 is introductory and has the aim to contextualise the thesis work. Chapter 2 provides a broad overview of the field of artificial intelligence, paying particular attention to the concept of Deep Learning; in this chapter will be provided several theoretical notions that are fundamental for the development of the entire project. Chapter 3 examines the programming language used and the libraries composing it, highlighting the reasons why this language was chosen. Chapter 4 represents the central body of the thesis, in fact the network and the strategy used for the calculation of the sheltering factor are outlined in detail. In Chapter 5, the results obtained is analysed and then compared with those achieved using the method discussed in Chapter 1; the comparison will be useful in defining the strengths and weaknesses of both strategies. To conclude, it will be established the guidelines for future improvements to the strategy presented.

Chapter 2

INTRODUCTION TO NEURAL NETWORKS

In this chapter some basic concepts related to the Neural Networks (NNs) and Deep Learning are explained, in order to provide fundamental concepts and to comprehend which are the main principles behind them.

Technology will make life easier in the next years, nowadays an increasing number of companies strongly trust in learning algorithms to facilitate our daily routine. These technologies are typically associated with artificial intelligence, machine learning, deep learning, and neural networks; these terms are often considered as synonyms, but it is important to specify how these are different and how each of them plays a specific role. To describe the relationship between AI, Machine Learning, Deep Learning and Neural Networks it is useful to consider them like Russian nesting dolls (*Figure 9)*, where basically each term is a component of the previous one. Neural Network make up the backbone of deep learning algorithms.

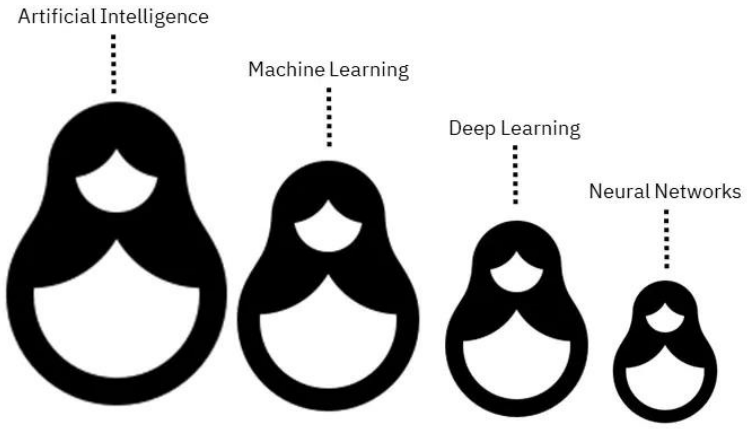

Figure 9: Relationship between artificial intelligence, machine learning, deep learning, and neural networks.

The concept of Artificial Neural Network was inspired by human biology and the way neurons of the human brain function together to understand inputs received from human senses. Who firsts worked on the Neural Network theory in 1943 were the neurophysiologist *Warren McCulloch* and the mathematician *Walter Pitts*; they wrote a paper on how neurons work which represents the starting

point for the future development of the Neural Networks. In 1949 the Canadian psychology *Donald Hebb* examined in depth the concept of neurons in his book, *The Organization of Behaviour,* and he comprehended that neural pathways strengthen over between neurons that tend to fire at the same time; this theory was called *Hebb's rule*:

> *"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased*[5]*."*

In 1958 the psychologist *Frank Rosenblatt* introduced the concept of Perceptron, the oldest neural network still in use today, that was a system with a simple input output relationship, modelled on a McCulloch-Pitts neuron definition. A McCulloch-Pitts neuron get in inputs, takes a weighted sum and returns '*0'* if the sum is below a given threshold and *'1'* otherwise. An interesting characteristic of the Perceptron was that the different weights can be 'learnt' through consecutive passed inputs, while minimizing the difference between desired and actual output. The end of the 1950s represents the turning point for Neural Networks; in these years and precisely in 1959 *Bernard Widrow* and *Marcian Hoff* of Stanford released models they called ADALINE and MADALINE (Multiple ADAptive LINear Elements) which were the firsts Neural Networks to be applied to a real-world problem. The latter was used to remove the noise from the telephone line, and it is still used. The enthusiasm of the AI scientific community in those years was extinguished with the publication of the book *Perceptrons* in 1969. The authors *Marvin Mincsky* and *Symor Papert* proved all the limits of perceptron theory developed by Frank Rosenblatt and the problems with Neural Nets. The effect of this book was powerful and dried-up funding to an extent that, for the next 10–12 years, no-one at the largest research institutions at the time and thereby the smaller ones too, would take on any project that had the doomed Neural Networks as its premise. The situation did not recover until 1982 when Japan announced its intention to begin its fifth-generation effort on Neural Networks, and few years later the American Institute of Physics in 1985 established a "Neural Networks in computing" annual meeting followed by the first International Conference on Neural Networks by IEEE in 1987. Thanks to continuous innovation, the recovered interest and their multiple applications, the Neural Networks in recent years have become one of the most used and appreciated technology.

2.1 BASIC NEURAL NETWORK STRUCTURE

The Neural Network architecture is composed by different layers (*Figure 10)*, each of which contains a specific number of neurons, and they represent the basic unit of a NN. There are three type of layer:

- *Input Layer:* It is the outermost layer. All the neurons in this layer receive inputs from an external source or from other nodes. Each of them is linked with other nodes from next layer, and every single connection has a specific weight. Weights are assigned to a neuron according to the relative importance against other inputs. The single node values from the input layer are multiplied by the corresponding connection weights and summarized, the resulting value represents the input for the hidden layer.
- *Hidden Layer:* This layer is defined "hidden" because it is not visible from the external world. All the relevant computation of a Neural Network takes place here. The hidden layer acquires all the inputs from the Input Layer and perform all the required operation to generate a result, which is provided to the Output Layer. Depending on the input values coming from the outermost layer, the hidden layer has a predefined "activation" function that determines whether or not this node will be "activated" and how "active" it will be.
- *Output Layer*: It represents the final layer where it is possible to appreciate the result.

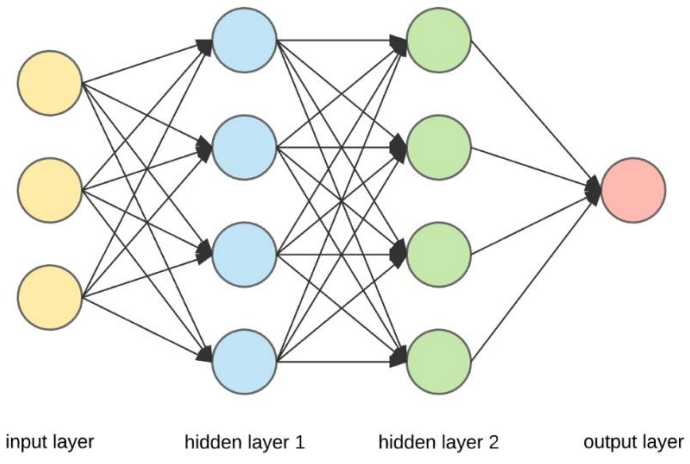

Figure 10: Fully Connected Neural Network structure.

2.1.1 Neural Networks Artificial Neurons

Artificial Neuron is a mathematical function, based on a model of biological neurons, where each neuron takes input, weight them separately, sums them up and passes this sum through a nonlinear function to produce output. Artificial Neuron represents the elementary unit in an artificial network. Each neuron contains an internal state called *activation signal*, all these units are linked to each other and the connections have different weights. The first neural network unit studied was the perceptron by Frank Rosenblatt in 1957. In the Minsky and Papert's book, the perceptron is defined as

> *"a device capable of computing all predicates which are linear in some given set […] of partial predicates*[6]*."*

The design of the perceptron was aroused by the structure of the neurons inside the human brain. *Dentrites* can be considered as the input layer (x_1,x_2,…,w_n and *costant*), and the *Axon* is like the weighted sum using weights (w_0,w_1,..,w_n), which are processed through the activation function to provide output response. In the *Figure 11* and *12*, the representations of perceptron and human neuron are reported to figure out all the similarities between these two elements.

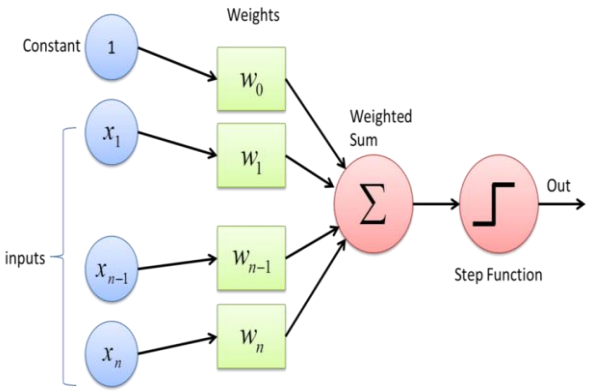

Figure 11: Perceptron.

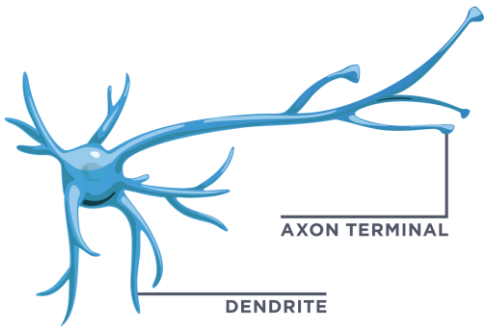

Figure 2.4: Human neuron

It is important to note that the weight of an input is related to the strength of the particular node, while the input bias value guarantees ability to shift the activation function curve up or down.The output of perceptron can be expressed in such algebric way:

$$Output = \begin{cases} 0\ if \sum_i w_i x_i - constant \leq 0 \\ 1\ if \sum_i w_i x_i - constant > 0 \end{cases}$$

To determine the output of each perceptron like *1* or *0* an Activation Function is used. Activation function defines the output from the neuron in terms of its combination. It can be basically differentiated into 2 types:

- *Linear activation function*
- *Non-linear activation function*

Using a linear activation function like the one reported in *Figure 12*, it is easy to understand that the output will not be limited between any range, and this doesn't help with the complexity or various parameters of usual data that is fed to neural networks.

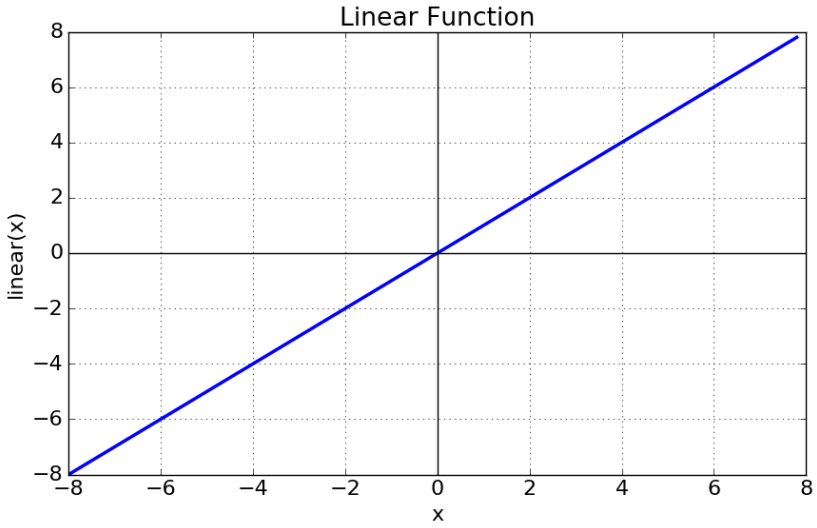

Figure 12: Linear Function.

The nonlinear are the most used activation functions. Nolinearity helps the model to generalize or adapt with the range of data and to differentiate between the output. The nonlinear activation function are mainly divided on the basis of their range or curve:

- *Sigmoid or Logistic Activation Function.* It is used for models whose aim is to predict probability as an output. Since propability of anything exists only between the range of *0* and *1*, sigmoid is the right choice. It is mainly used in feed-forwards net.
- *Softmax function.* It is a more generalized logistic activation function which is used for multiclass classification.
- *Hyperbolic tanget Activation Function.* This type of activation function is similar to the sigmoid but with better performances. Hyperbolic tanget activation function will map negative input as strongly negative and the zero inputs will be mapped near zero in the *tanh* graph. This activation function is mainly used in feed-forwards net.
- *ReLu (Rectified Linear Unit) Activation Function.* This is another nonlinear activation function that is used in almost all the convolutional neural networks or deep learning networks. An inportant advantage of the ReLu function is that it is more computationally efficient the the previous non-linear activation functions. The problem of this activation function is that all the non-positive values become zero immediately which decreases the ability of the model to fit or train from the data properly. Then, any negative input given to the ReLu activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

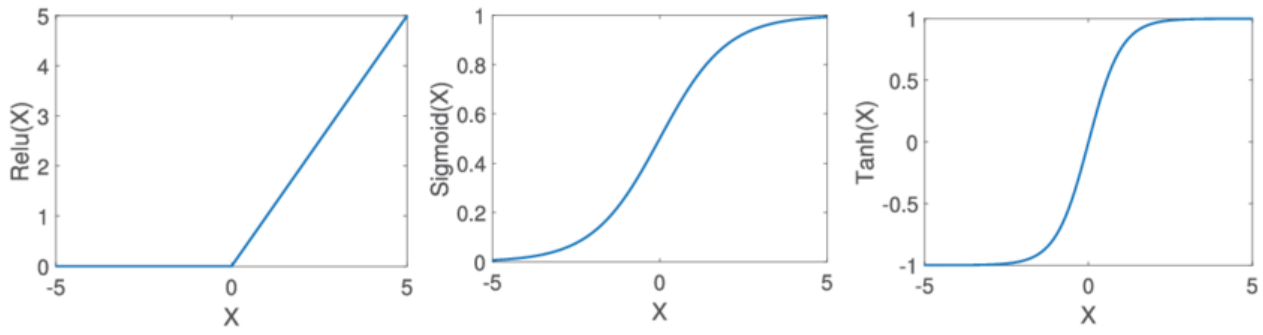

Figure 13: ReLu Function, Sigmoid Function, Tanh Function.

2.1.1.1 Gradient Descent

Given a certain input, the neurons of an artificial neural network "*fire"* through a mathematical process, that produces an output based on *bias* and *weight* values. Each *epoch*, or *iterations*, the machine learning algorithms tries to predict the output value and tries to improve the prediction tweaking the weights. It is useful to introduce a function which can measure how well the model is to obtain a prediction, this function is called *cost function.* This function will penalize any error our model makes (by assigning a cost) with respect to the current parameter values. In order to obtain the parameters that yields the best machine learning model performance, the *cost function* must be minimized. The tool that can be used to minimize the cost function is the *gradient descent*, it is an efficient optimization algorithm that attempt to find a local minimum of the *cost function*. A *local minimum* is a point where the *cost function* is lower than all neighbouring points of the *cost function*. Common cost functions are:

- *Mean squared error*
- *Cross-entropy loss (log loss)*

In this case the *MSE* is chosen to completely explain the concept of *Gradient Descent.*

$$C(w,b) \equiv MSE(w,b) \equiv \frac{1}{2n}\sum \|y(x) - a\|^2$$

where:

- w represents all the weights of the network;
- b represents all the biases in the network;
- x is the input;
- y is the actual output;
- a is the predicted output.

The goal of the *gradient descent* algorithm is to minimize the *cost function*, this is achieved performing two steps iteratively. The first step consists of computing the gradient (slope), the first order derivative of the function at that point:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots\ldots, \frac{\partial C}{\partial v_n}\right)$$

where v is a generic vector, representing *weights* and *biases*, from which the *cost function* depends. The second step consists of make a move in the direction opposite to the gradient:

$$\Delta v = -\alpha \nabla C$$

Where α is small parameter, called *learning rate,* it determines the size of the steps by the gradient descent algorithm. Summing up, what happens is that every step the gradient according to the last equation finds an even more minimum until it reaches a local minima. In the best case, it could correspond also to the global one.

2.2 Deep Learning

Deep learning is defined as a machine learning technique that creates, Artificial Neural Network (ANN) to reproduce the structure and the behaviour of the human brain in processing data and creating patterns for use in decision making. The meaning of the adjective deep is explained in the *Youshua Bengio*'s book *Deep Learning:*

> *"The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning*[7]*."*

Deep learning or hierarchical learning is composed by a huge number of hidden layers, typically more than 6 but usually much higher, and each of them performs a nonlinear process to extract relevant feature from data and transform data into different levels of abstraction. Deep learning is a machine learning technique that is powered by massive amounts of data. As per *Andrew Ng*, the chief scientist of China's major search engine Baidu and one of the leaders of the Google Brain Project,

> *"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms."*

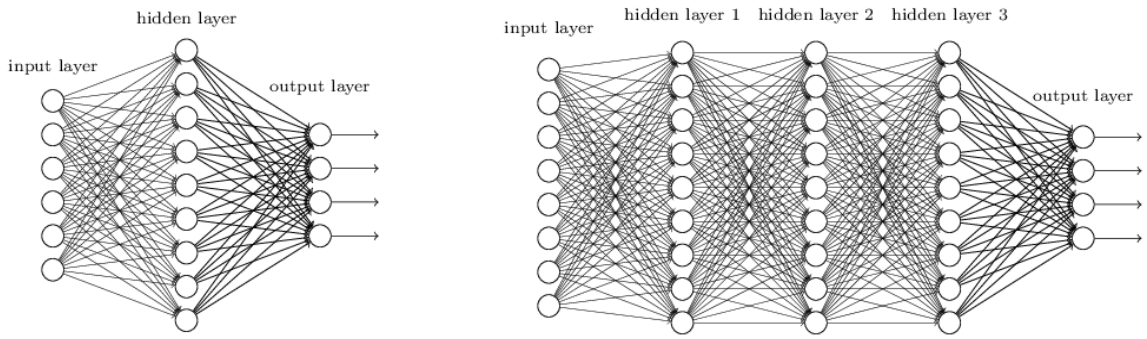

Figure 14: "Non-deep" Neural Network VS Deep Neural Network.

A big advantage with deep learning, and a key part in understanding why it is becoming popular, is that if a larger Neural Network (NN) is trained with more and more data, their performance continues to increase, and this is in general not valid for the others machine techniques that reach a plateau in performance (*Figure 15)*.

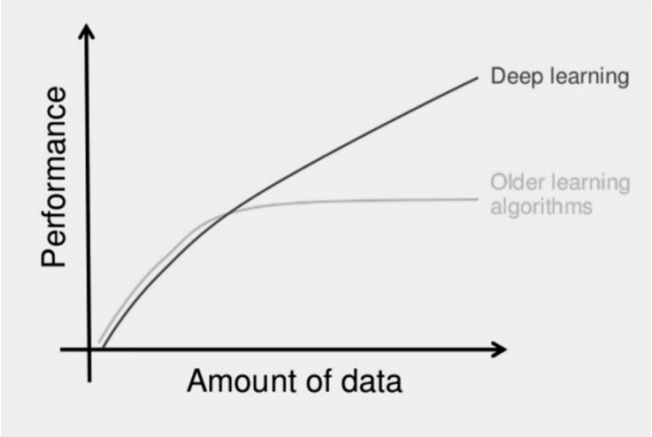

Figure 15: Plot of how science techniques scale with amount of data.

Unlike the traditional machine learning algorithms, deep learning eliminates the need of domain expertise and hard-core feature extraction, and this is why deep learning algorithms is able to learn data features from data in an incremental way.

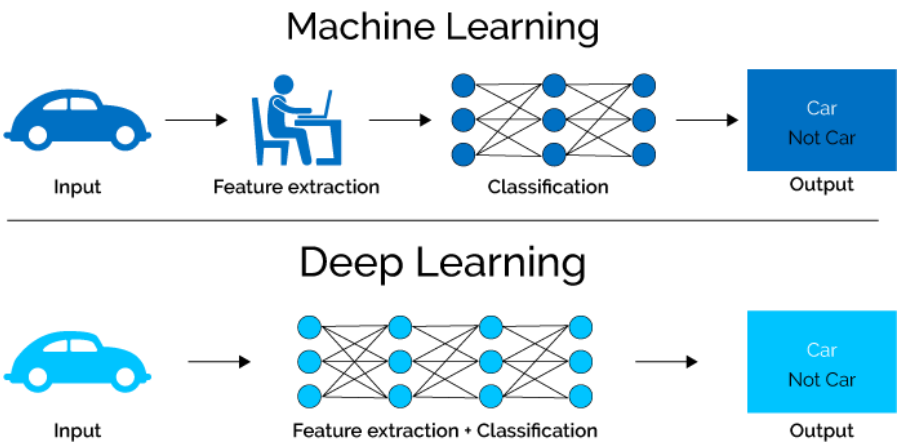

Figure 16: Change in work pipeline.

Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance[8][9][10].

2.2.1 Deep Learning for Computer Vision

Computer vision is subfield of the artificial intelligence and machine learning, which studies how computers see and understand digital images and videos.

> *"Computer vision is the automated extraction of information from images. Information can mean anything from 3D models, camera position, object detection and recognition to grouping and searching image content* [11]*."*

Recent computer vision technologies are based on deep learning algorithm; it spans all tasks performed by biological vision systems, including "seeing" or sensing a visual stimulus, understanding what is being seen, and extracting complex information into a form that can be used in other processes [12]. Computer vision is used in several applications like, autonomous vehicle, image classification, face recognition, identifying objects in images, video analysis and classification.

2.3 IMAGE SEGMENTATION

Image segmentation is a computer vision algorithm, it implicates dividing an image into segments to simplify Image or video analysis. An image is a collection of different pixels, image segmentation grouped together all the pixels that present similar characteristics. Newest image segmentation techniques are powered by deep learning technology. Here are several deep learning architectures used for segmentation based on their main technical contributions: Convolutional Neural Networks (CNNs), Fully convolutional Networks (FCNs), to name a few.

Depending on the granularity of the image analysis it is possible to define two types of segmentation techniques:

- *Semantic Segmentation*: The pixels of image are classified into relevant classes of objects.
- *Instance Segmentation*: Identifies each instance of each object in an image. It differs from semantic segmentation in that it does not categorize every pixel.

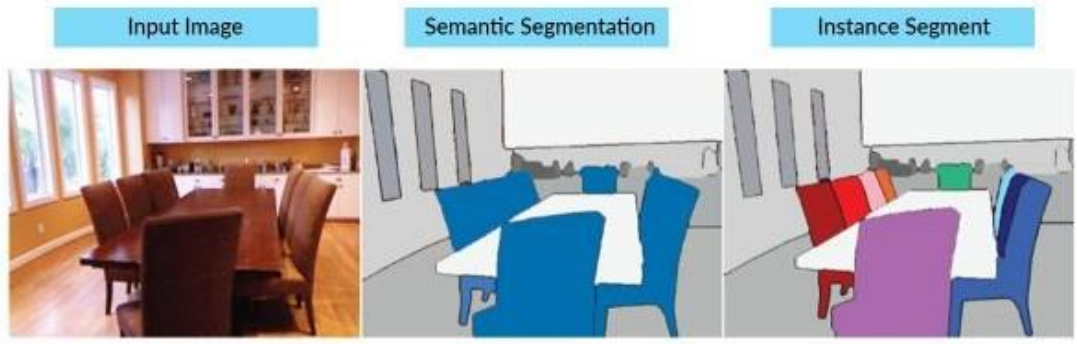

Figure 17: Image segmentation techniques.

Image segmentation figures out the relations between objects, as well as the context of objects in an image. Applications include face recognition, number plate identification, and satellite image analysis and autonomous vehicles use it to understand their surroundings.

2.3.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is one of the most used deep learning architectures especially for computer vision tasks like image classification, face recognition, identifying and classifying objects, and analysis and processing in robots and autonomous vehicles.
CNN employs a three-dimensional structure, with three neural networks to analyse the red, green, and blue layers of a colour image. CNNs mainly comprise four type of layers:

- Convolutional layers + ReLu
- Flattening layers
- Pooling layers

The convolutional layer and the pooling layer together form the *i-th* layer of a Convolutional Neural Network (CNN). Depending on the complexities in the images, the number of such layers may be increased for capturing low levels details even further, but at the cost of more computational power. Consider as input a $4 \times 4 \times 3$ RGB image that has been separated in three colour planes. Hence, for a given RGB image of size 255×255 (Width $\times$ Height) pixels, we will have 3 matrices associated with each image, one for each color channels, it is worth to consider the input image like a 3-dimensional structure called input volume ($255 \times 255 \times 3$).

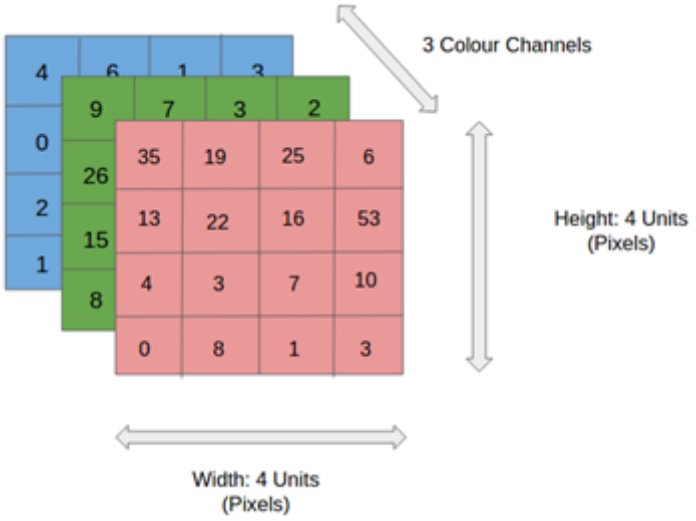

Figure 18: RGB input image matrices example.

The intent of the CNN is to reduce the images, especially in the case where the input volume has huge dimension, into a form which is easier to process without losing properties which are essential for obtain a good prediction.

2.3.1.1 Convolutional Layer

From a mathematical point of view convolution is a function derived from two given functions by integration which express how the shape of one of this function is changed by the other one. The aim of the convolution layer is to extract high-level features from the input image. The elements involved in the convolution operation are the input image, the Kernel (or filter) and the feature map which is the result of the convolution operation of the firsts two elements.
Filters are integral component of the layered architecture, they are simple matrices that are multiplied by submatrices of the image, they slide right till it parses all the image width and then hop down to the left beginning until the whole image is not covered[13]. In *Figure 19*, a simple example is briefly presented.

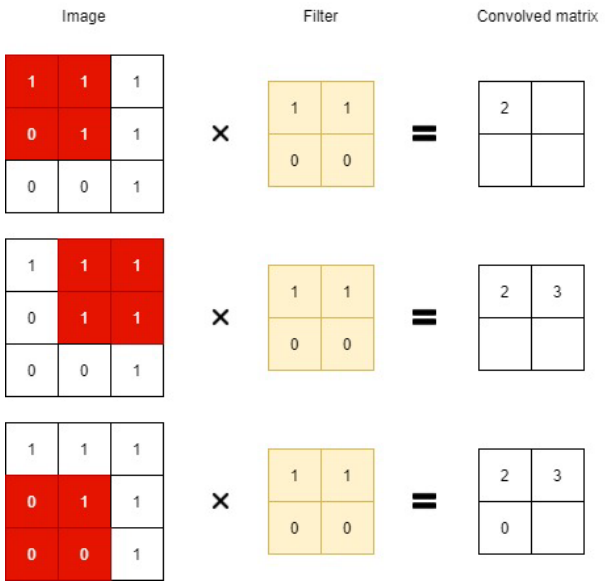

Figure 19: Convolution operation example.

As could be seen, the number of cells that the filters slide to the right, or down, after each computation is equal to 1, so called *Non-Strided* kernel, but it could be increased as a project preference. There are

two possible results to the operation, one in which the convolved matrix dimension is smaller than the dimension of the input images and the other in which the dimension of the convoluted matrix are either increased or remains the same. This is done by applying *Valid Padding* in case of the former, or *Same Padding* in the case of the latter. Convolutional Neural Network (CNN) required more than one single convolutional layer, because the first convolutional layer is responsible to extract low level features such as edges, colours, gradient orientation. Using additional layers architecture adapts to the high level feature as well, obtaining a network which has the healthful understanding of images in the dataset. To provide a complete description of the convolution layer, the ReLu step is introduced. ReLu is the acronym of Rectified Linear Unit for a non-linear operation. The output is $(x) = \max(0, Input)$, ReLu is an element-wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The objective of ReLu is to introduce non-linearity in CNN since most of the real-world data we would want our CNN to learn would be non-linear. Most of the data scientists use ReLu because the ReLu function has better performances than the other ones.

2.3.1.2 Pooling Layer

Pooling layer section further reduces the spatial dimension (Width × Height) of the *Feature Map,* to decrease the computational effort required to process the data. This layer is not only responsible of the dimensionality reduction of the convoluted feature, but it is also useful for extracting dominant features which are rotational and positional invariant. Pooling operation can be of different types: *Max Pooling* and *Average Pooling*, the former one returns the maximum value from the portion of the image covered by the Kernel while the latter on returns the average of all the values from the portion of the image covered by the filter. *Max Pooling* also performs the Noise Suppressant, for this reason it is preferable respect the *Average Pooling*. Similar to the convolutional layer, the pooling operation affects the output dimension. However, the same concept of *Same Padding* and *Valid Padding* can be applied also for this layer.

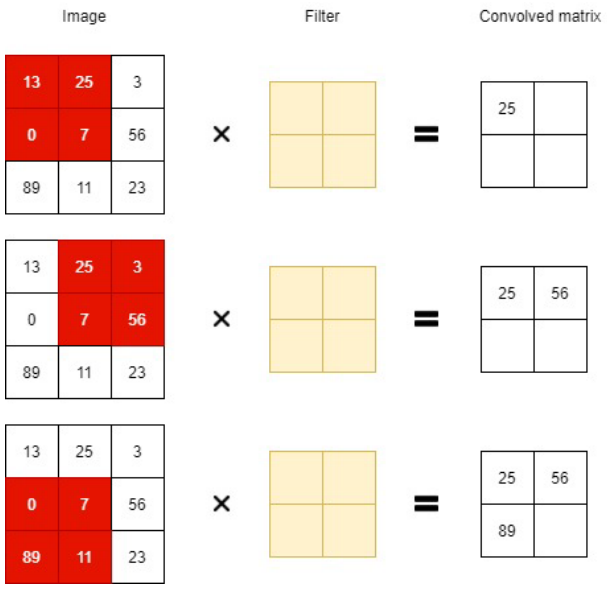

Figure 20: Pooling operation example.

2.3.1.3 Flatten Layer

Flatten layer is used in order to change the shape of the convolved and pooled maps into a column vector, in this way the data are changed into the correct format for a dense layer to interpret. This layer is connected to the final classification model, which is called a *fully connected* layer. In other words, we put all the pixel data in online and make the connections with the final layer.

2.3.1.4 Fully connected layer

The Fully connected layer is a traditional Multi-Layer Perceptron that uses a Softmax activation function in the output layer, the SoftMax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sums to one. Fully connected neural network controls the final probabilities and decide which class the images belong to. In *Figure 21* a complete scheme of a CNN is reported to group all the previous layers discussed and to understand how they are interconnected.

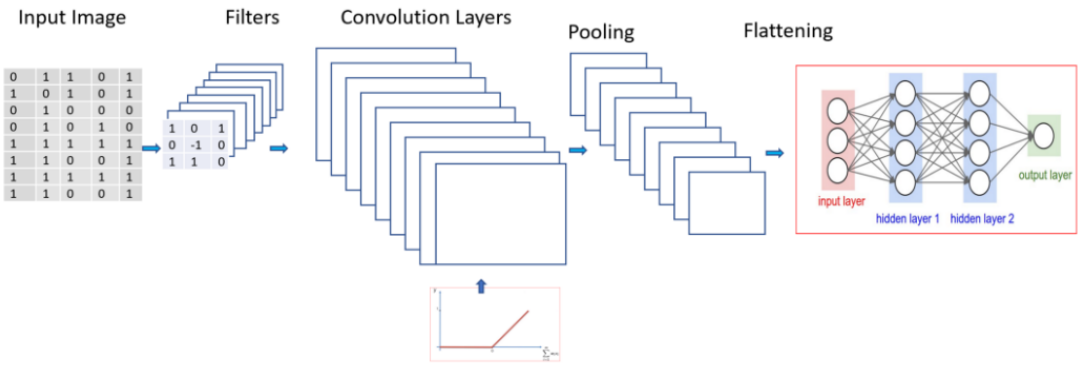

Figure 21: Convolutional Neural Network architecture.

2.3.2 Fully Convolutional Network

Long *et al.* [14] proposed one of the first deep learning works for semantic segmentation, using a fully convolutional network (FCN). FCNs employ solely locally connected layers, such as convolution, pooling and upsampling. FCN is a deep neural network that goes beyond the limits present in the CNN. As reported in the above part of the thesis the CNNs have Fully Connected Layers which limit the dimension of the input image, if the size of the input is bigger than the requested dimension of the fully connected layer, the image must be resized. In other words, the FCN can be simply considered a Convolution Neural Network where the fully connected layer is exchanged with a 1×1 convolutional layers. This reduces the total number of parameters and increases the computational speed reducing latency. A FCN acts on an input image of any size and produces an output of corresponding (possibly resampled) spatial dimension [14].

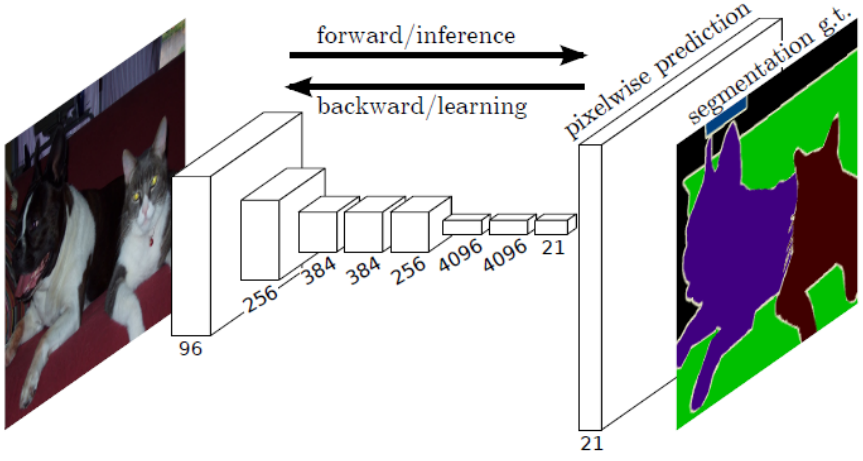

Figure 22: Original Fully Convolutional Network architecture from [15] .

In *Figure 22,* the image of the original Fully Convolutional Network architecture is reported. The model will learn the main feature of the image using a CNN extractor that represents the *encoder* part of the model. As the input passes through the different Convolutional layers, it gets downsampled; then the output is moved to the *decoder* section of the model that is composed by other Convolutional layers, popular decoders are FCN-32, FCN-16, and FCN-8. The *decoder* upsamples in the image gradually to its original size, in this way the segmentation mask image is obtained. There are different methods that can be used to upsample the resolution of the processed image. Despite to pooling operation, which downsample the resolution by summarizing a local area with a single value (ie. *Average or max polling)*, the "*unpooling*" operations upsample the resolution by distributing a single value into a higher image resolution. The most diffused methods used to upsample an image are:

- *Nearest-Neighbor*
- *Bed of Nails*
- *Max Unpooling*
- *Transpose Convolution*

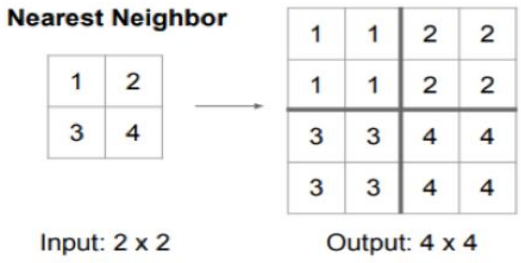

Figure 23: Nearest-Neighbor upsample method example.

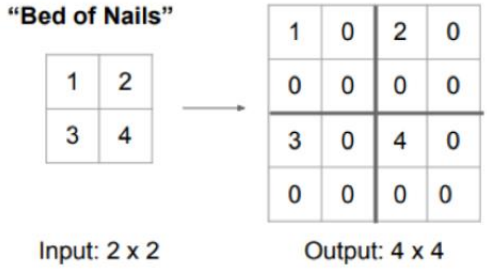

Figure 24: Bed-Nails upsample method example.

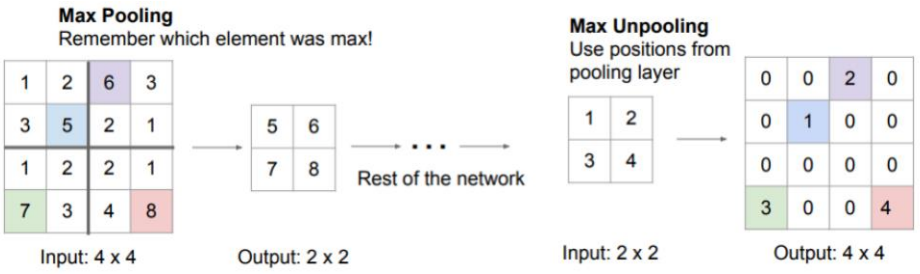

Figure 23: Max upsample method example.

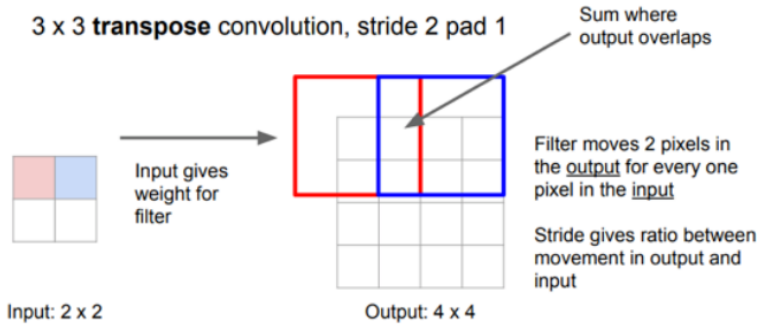

Figure 24: Transpose Convolution upsample method example.

"Fully Convolutional Networks for Semantic Segmentation" by Long et al. introduced the idea of *skip connections* into FCNs to improve segmentation accuracy. Skip connections in deep architectures, as the name suggests, skip some layer in the neural network and feeds the output of one layer as the input to the next layers (instead of only the next one). Skip connections combine the coarse final layer with finer, earlier layers to provide the local predictions that "respect" global position. Refer to *Figure 25* for a diagram of the skip connection architecture. To create FCN-16s, the authors added a 1x1 convolution to pool4 to create class predictions and fused these predictions with the predictions computed by conv7 with a 2x upsampling layer. For FCN-8s, they added a 2x upsampling layer to this output, and fused it with the predictions from a 1x1 convolution added to pool3 [14].

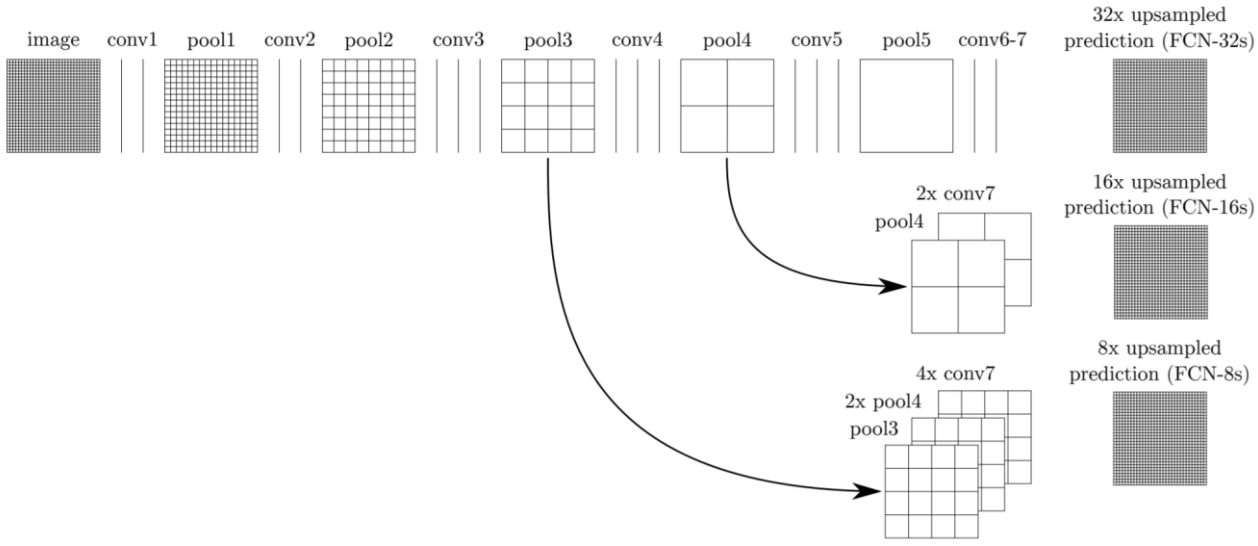

Figure 25: Fully Convolutional Network architecture with skip connection.

The *Figure 26* shows that FCN-16s provides much finer segmentation than the standard FCN-32s, and FCN-8s even finer segmentation (much closer to ground truth).

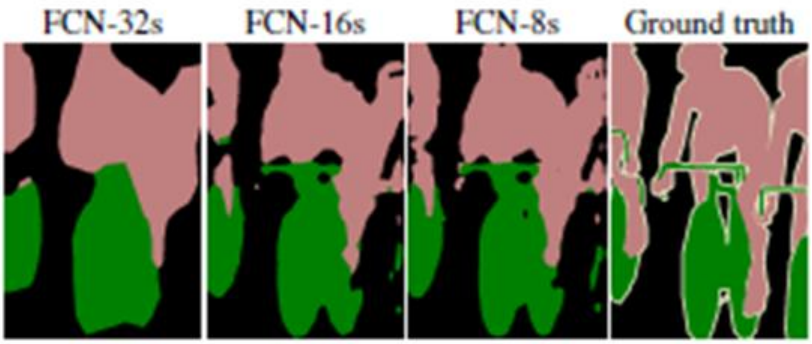

Figure 26: Performances of the different decoder.

The accuracy table below quantifies the segmentation improvement from skip connections.

	Pixel acc.	Mean acc.	Mean IU	f.w. IU
FCn-32s-fixed	83.0	59.7	45.4	72.0
Fcn-32s	89.1	73.3	59.4	81.4
FCN-16S	90.0	75.7	62.4	83.0
FCN-8S	90.3	75.9	62.7	83.2

Table 1: Comparison of skip FCNs.

2.3.2.1 U-NET

U-net is a convolutional neural network designed to Biomedical Image Segmentation, proposed by *Olaf Ronneberger*, *Philipp Fischer*, and *Thomas Brox* in the paper "*U-Net: Covolutional Networks for Biomedical Image Segmentation*". U-net architecture was designed such that it works with few images and yields more precise segmentation [16].

The name U-net is due to the shape of its architecture, this is clear from *Figure 27*.

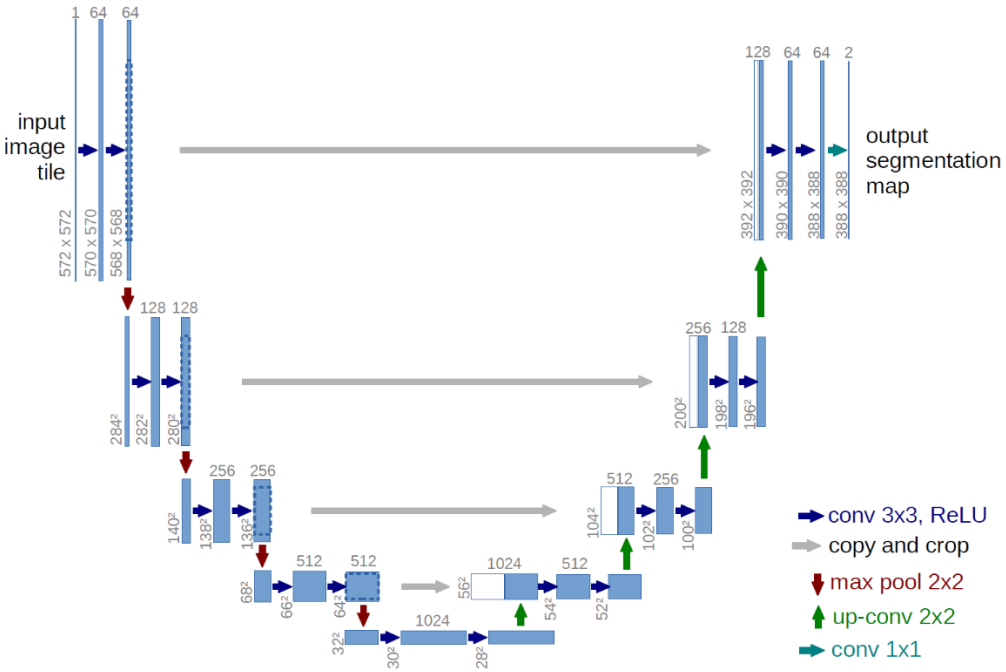

Figure 27: U-Net architecture.

In the U-net architecture, there are two well-defined paths: contraction path (*left side*) and expansive path (*right side*). The former (also called *encoder),* is a stack of convolutional each followed by a *Rectified Linear Unit* and *max pooling* operations, during the contraction path the spatial information is reduced while feature information is increased, the *encoder* is usually a pretrained classification network such as VGG/ResNet. The latter (also called *decoder*) is the symmetric expanding path which combines the feature and spatial information through a sequence of up-convolutions and concatenations with high resolution feature from the contracting path, this mechanism is called "*skip connection*". This mechanism consists of skipping layers and providing the output of one layer to another one by skipping a few layers between them. Skipping connections allow combining low-level feature maps with higher-level ones, which enables precise pixel-level localization [17].

The difference between U-net and the previous discussed architecture FCN is the upsampling method. In the FNC, the same level downsampling feature map and upsampled feature map are simply added and upsampled right away. On the other hand, in U-net, it is concatenated and then goes through some convolutional layer for additional processing [18].

Chapter 3

SOFTWARE

In this chapter we introduce software, tools, APIs or whatever has been involved in this work. Moreover, it will be discussed the reasons why Python was chosen as the programming language to develop the model. There are only few programming languages that can be used for deep learning and Python is one of them. Python is an interpreted object-oriented programming language, it offers developers to build powerful backend systems for Python AI projects and has a huge number of libraries for Machine Learning. In the next paragraphs some of those libraries which have been of fundamental importance for the development of the entire thesis project will be presented and described.

3.1 BING MAPS REST SERVICES APPLICATION PROGRAMMING INTERFACE

The Bing Maps REST Services Application Programming Interface (API) provides a Representational State Transfer (REST) interface to perform tasks such as creating a static map with pushpins, geocoding an address, retrieving imagery metadata, or creating a route [19]. API REST is an application programming interface that conforms to the constraints of REST architectural style. REST stands for Representational State Transfer, it is an architectural style for distributed hypermedia systems, which provides interoperability between local computer script and Internet following a client-server approach. REST can be described as a set of guidelines and "*constraints*". An application programming interface (API) to be considered a REST must respect the following criteria [20]:

- *Client-server* architecture made up of clients, servers, and resources, with requests managed by means HTTP;
- Stateless client-server communication;

- Cacheable data that streamlines client-server interactions;
- Layered System;
- Uniform interface.

Using HTTP or HTTPS protocols, a REST Service API is accessible using the python library *requests*. Bing Maps REST Service API makes available a simple way to create a static map that can be downloaded. The image is given as response of the HTTP *GET* request, to save it a *for* cycle is implemented to be sure that the image data are decompressed, those data are saved in 128 byte chunks in binary mode file and the saved into a *jpg* format. BING Maps Service API gives the possibility to obtain customized images, setting a predefined number of parameters, that are useful for training and testing the developed model. As it is evident, one required parameter is to specify where is needed the static map, it can be specified in two different ways:

- *CenterPoint:* Providing the geographical coordinates of the middle point of an area and also to set a *zoomLevel* is possible to acquire the image of the corresponding area;
- *BoundingBox*: This second method is suitable if we want to acquire the image of an area, whose dimensions are known a priori, that is specified using four parameters: *south latitude*, *north latitude*, *west longitude*, *east longitude*. These four parameters represent the geographical coordinates of the vertices of the image.

Using Bing Maps REST Service API is possible to get image of:

- Street map;
- Aerial, and overhead view;
- Aerial with Labels, and overhead view with road and buildings in addition;
- Roads.

Figure 28: Aerial image of Turin acquired using BING map API.

3.2 PYTHON

"Python is the most powerful language you can still read."

Pau Dubois

Python is a famous general-purpose programming language that can be utilized for a huge number of applications. It supports multiple programming paradigm, including *structured* (procedural), *object-oriented* and *functional programming*. *Object-oriented paradigm* is a paradigm used in programming based on the concept of "*object*", that consists of data and code: data in the form of *fields* (*attributes or properties*), and the code in the form of procedures (*methods*).

Python is often described as "*batteries included language*" due to its comprehensive standard library, these libraries are available in source or binary form without charge for all major platforms and can be freely distributed.

Python is one of the programming languages that is observing growth and popularity year by year. In order to understand why Python is becoming so popular some of the most important advantages are reported [21]:

- Python is simple, easy to learn syntax emphasized readability and therefore reduces the costs of program maintenance;
- Mature and supportive Python Community;
- Hundreds of Python Libraries and Frameworks;
- Versatility, Efficiency, Reliability, and speed;
- Big data, Machine Learning and Cloud Computing.

In this work Python is used for Machine Learning purpose, it represents the go-to choice for Artificial Intelligence project [22].

> *"Python seems to be winning battle as preferred language of MachineLearning. The availability of libraries and open source tools make it ideal choice **for** developing ML models."*

Different characteristics that place Python among the top programming languages for Machine Learning, Deep Learning and AI are listed below:

- Free and open-source nature;
- Exhaustive libraries;
- Smooth implementation and integration;
- Used for Soft Computing.

In the next paragraphs the most important Python packages involved in this work will be introduced.

3.2.1 Pytorch

PyTorch [23] is a library for Python programs that facilitates building deep learning projects. It emphasizes flexibility and allows deep learning models to be expressed in idiomatic Python [24].

Figure 29: Pytorch LOGO.

PyTorch was released with the feature to perform fast mathematical operations on dedicated hardware components, which makes it convenient to design neural network architectures and train them on individual machines or parallel computing resources. *PyTorch* library makes available two high-level features[25]:

- Tensor computing (like Numpy) with strong acceleration via *graphics processing units* (GPU);
- Deep neural networks built on a tape-based automatic differentiation system.

As it has been said in *chapter 1*, deep learning allows users to perform a wide range of complex tasks, therefore they need tools that are flexible, that can be easily adapted to such a large range of problems, and efficient, to allow the training of big dataset in reasonable time.
Due to clear syntax, streamlined API, and easy debugging *PyTorch* represents the right choice for deep learning application. To make easy the representation of a deep learning machine, PyTorch provides a core data structure, called *Tensor* (*Figure* 30), which can be considered a multidimensional array, holds numbers, vector, matrices or arrays in general. *Tensors* are similar to NumPy arrays but with the difference that the former one can be used on GPU.

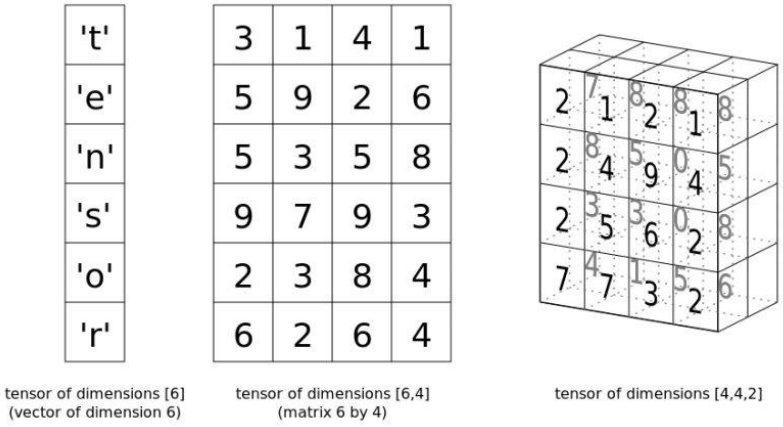

Figure 30: Tensor example.

PyTorch has two characteristics that make it particularly relevant for deep learning: first, it provides accelerated computation using GPUs, often yielding speedups in the range of $50x$ over doing the same calculation on a CPU; second, PyTorch provides facilities that support numerical optimization on generic mathematical expressions, which deep learning uses for training [24].

The core *Pytorch* modules for building neural networks are located in *torch.nn* , which contains common neural networks layers (*fully connected layers, convolutional layers, activation function, and*

function loss) and other architectural elements. Using all these available elements it is possible to building a NN model, in order to train it some additional elements are needed: *source of training data*, an *optimizer* to adapt the model to the training data, a way to get the data to the hardware that will perform all the operations required for training the model.

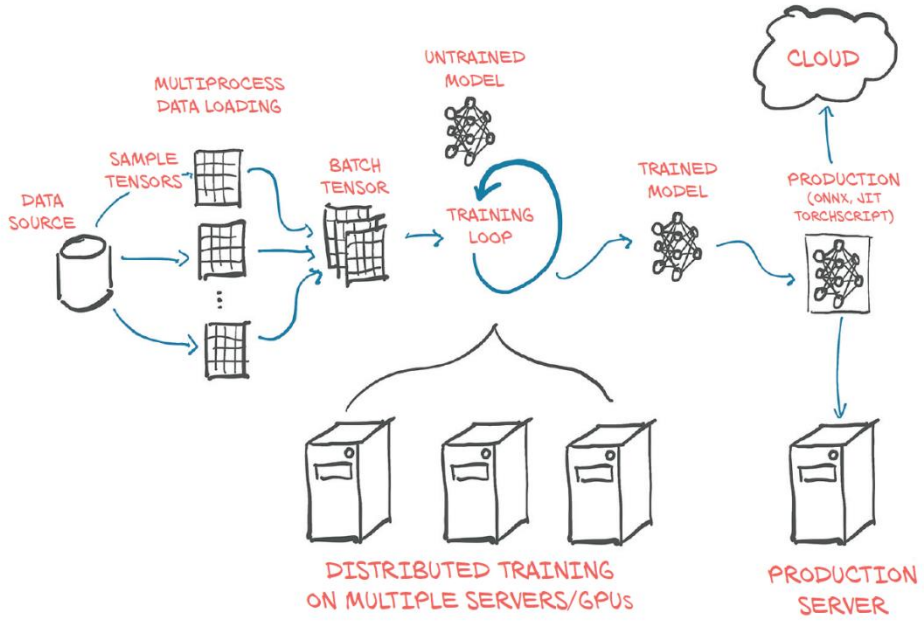

Figure 31: PyTorch overall project structure.

In *Figure 31* a Pytorch project structure with all the required steps to obtained a trained model is reported, to better understand how Pytorch supports deep learning project.

Training data before reaching the model must be converted into a something PyTorch can understand: *tensors*. The step between the original custom data and a PyTorch tensor is the *Dataset* class PyTorch provides in *torch.utils.data.* PyTorch guarantees an efficient and fast data loading process. The instances provided in *DataLoader,* can generate child processes to load data from a dataset in background so that it is ready and waiting from the *training loop* as soon as the loop can use it [24]. At each iterations of the *training loop,* the model is evaluated on the samples obtained from the data source, then the outputs of the model are compared to the desired outputs (target) using some *criterion* or *loss function*. PyTorch has a variety of loss functions, and they are provided in *torch.nn* submodel. After that the output and the target are compared, it is needed to push the model a little to move outputs to better resemble the target, and this is where autograd engine comes in; but also an optimizers is needed to produce updates, and this is what PyTorch offers in *torch.optim*. In order to use more elaborate hardware like multiple GPU or multiple machines that contribute their resources to train a

large model, *torch.nn.parallel.Ditributed-DataParallel* and the *torch.distributed* submodule can be employed to use additional hardware [24]. At the end of the training loop, a model whose parameters have been optimized on the required task is obtained (*trained model*).

3.2.2 ImgAug

ImgAug is a Python library for image *augmentation* in machine learning project. *Data Augmentation* is a technique that can be used to get bigger the size of the training dataset by obtaining modified data from the original one. This technique is mainly used in order to prevent *overfitting,* or if the original dataset is too small to train on. It is also a good practice to use DA to obtain better performance from the developed model, in other words it is possible to improve the performance of the model by augmenting the data already available. *Data Augmentation* is frequently used when building a deep learning model. This work will focus on image augmentations. Here are listed the augmentation techniques that are used for images [26]:

- *Geometric transformations;*
- *Color space transformations;*
- *Kernel filters;*
- *Random Erasing;*
- *Mixing images.*

Every deep learning model has its own augmentation methods or even a whole library. For its simple API and for the big number of augmentations techniques *Figure 32*, ImgAug is used.

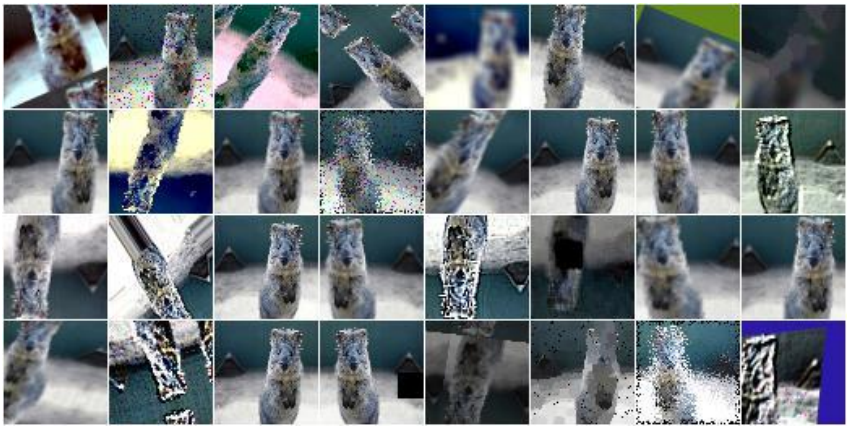

Figure 32: Different augmentation techniques.

3.2.3 Scikit-image

Scikit.image [27] is an open-source image processing library for the python programming language [28]. It can be considered as a collection of algorithms for image processing. Among all the algorithms available in this library the most important are:

- *Segmentation;*
- *Color space;*
- *Feature detection;*
- *Analysis;*
- *Filtering.*

Scikit-image is designed to work with scientific libraries as *Numpy* and *Scipy*.

3.2.4 Requests

Requests is an elegant and simple HTTP library for Python, built for human beings [29]. This HTTP library allows to send HTTP/1.1 requests in a very simple manner, without manually add query strings to URL. In the previous section we introduced the Bing Maps Service API which provides an interface as a *server,* and now we are introducing the Python library *Request* which provides implementation as a client. The concept is straightforward, the library allows to send specific HTTP request to a server and if the request has been done successfully the REST API gives back the image as response. As told before, one of the most important advantages of using Requests library is that there is no need to manually add query string to URL, this makes the creation of the complete URL simpler.

```
"query": {
        "mapLayer": "Basemap,Buildings",
        "imagerySet": "AerialWithLabels",
        "mapSize": "300,300",
        "key": "Token"
},
```

Figure 33: Query code example.

Figure 33 shows a query example used in the code. The script reads this query from a *.json* file and just adds it as a parameter of the function *get*. Finally, to complete the URL, the running script computes every step the necessary GEO reference. Just in case, was developed both methods to catch images either the one with only the center point or the one with the boundary area.

3.2.5 Pyproj

Pyproj is the Python interface to *PROJ*, which performs *cartographic transformations* and *geodetic computations*. *PROJ* is a coordinate transformation software that converts geospatial coordinates from one coordinate reference system (CRS) to another, it supports more than hundred different map projections and can transform coordinates between datums using all the geodetic techniques. Pyproj package contains two classes to perform cartographic transformations and geodetic computations, here are listed [30]:

- The *Proj class* can convert from geographic (*longitude*, *latitude*) to native map projection (x,y) coordinates and vice versa, or from on map projection coordinate system directly to another;
- The *Geod class* can perform forward and inverse geodetic computations. The forward method computes latitude, longitude and back azimuth of a terminus point given the latitude and longitude of an initial point, plus azimuth and distance. The inverse computation involves determining the forward and back azimuths and distance given the latitudes and longitudes of an initial and final point.

Input coordinates can be provided as Python arrays, list/tuples, scalars or numpy/Numeric/numarray arrays.

3.2.6 Pycocotools

Pycocotools is a python API that assists in loading and visualizing the annotations in COCO. COCO stands for Common Objects in Context, it is a dataset made available by Microsoft team designed for: object detection, segmentation, person keypoints detection, stuff segmentation and caption generation. COCO consists of 91 common objects categories with 82 of them containing more than 5000 labeled instances. In total, the dataset has 2,500,000 labeled instances in 328000 images. DATASET

annotations are stored using *JSON* file, *JSON* stands for *JAVAscript Object Notation* is a format used for data exchange. In every *.JSON* file we can locate 4 different parts:

- Info;
- Image;
- License;
- Annotation.

In the following image we can appreciate the four different fields that make up a JSON format file[31].

```
{
    "info": info,
    "licenses": [license],
    "images": [image],
    "annotations": [annotation],
}

info{
    "year": int,
    "version": str,
    "description": str,
    "contributor": str,
    "url": str,
    "date_created": datetime,
}
license{
    "id": int,
    "name": str,
    "url": str,
}
image{
    "id": int,
    "width": int,
    "height": int,
    "file_name": str,
    "license": int,
    "flickr_url": str,
    "coco_url": str,
    "date_captured": datetime,
}
```

Figure 34: Compelete .json file format.

Focusing on the annotation field for object detection (segmentation), it contains several information such as: *category ID*, *area*, *box*, *segmentation mask* for target (*Figure 35)*. The segmentation format depends on whether the instance is a single object or a set of objects:

- *Single object*: corresponds to "*iscrowd=0*" and the polygon format, list of polygon vertices around the object, will be used;
- *Set of object*: corresponds to "*iscrowd=0*" and the RLE format will be used.

```
annotation{
  "id": int,
  "image_id": int,
  "category_id": int,
  "segmentation": RLE or [polygon],
  "area": float,
  "bbox": [x,y,width,height],
  "iscrowd": 0 or 1,
  }
```

Figure 35: Annotation field and its subfield of .json file.

In addition to what has already been said, each object will have a rectangular box, the coordinates of the left corner at the top, and the length of that rectangle are represented in the form of arrays. The "area" represents the aria of encoded mask. Finally, the categories field in the annotation structure stores the "id" of the category to which the current object belongs, and the name of the supercategory to which it belongs [31].

This Python API is crucial for the purpose of this thesis, as matter of fact the dataset used to train our model contains satellite imagery and the corresponding annotations in MS COCO format. *Figure 36* summarizes in a short and effective way what is discussed in this paragraph, and also helps us to understand how get a mask of an object from a list of polygon vertices.

Figure 36: Example of COCO dataset format.

Chapter 4

PROPOSED AI-BASED STRATEGY

In the following chapter, the strategy used to calculate the Sheltering Factor will be discussed. The model used will be described in detail, focusing on its training process and the way in which the network processes its estimation, and on how these can be used for the calculation of the Sheltering Factor. The main objective of this thesis is the calculation of the average Sheltering Factor of a given area identified by the geographical coordinates (longitude and latitude) of its central point. In order to calculate this factor, it is therefore necessary to identify the individual buildings present in the aerial image and see in what percentage they occupy the area analysed. The solution to this problem is the use of a deep neural network able to automatically extract buildings from high-resolution satellite images (*Figure 37)*. As stated in *Chapter 2*, the implementation of a deep neural network architecture requires the construction of a considerable number of hidden layers. It is also necessary to obtain precise and accurate results a large dataset, in the case of this thesis work composed by label images. The strategy adopted is therefore based on the use of a pre-trained deep neural network able to perform segmentation, which produces as output the extraction of buildings. Among the countless deep neural network architecture available, the one that has met the requirements of the thesis work is the winner of the international *Mapping Challenge Competition* promoted by the crowdAI. The winning team of the competition proposed a solution that is based on the U-net architecture discussed in S*ection 2.3.1.6*. The proposed network has the popular encoder-decoder architecture with skip connections, but it also has a couple of essential changes that make it usable for semantics as well as for instance segmentation task. One of the changes made to the original U-net network was to replace the traditional encoder with a *ResNet* architecture pretrained on *ImageNet*, a big dataset designed for the purpose on *Computer Vision Research*. The use of a *ResNet* architecture, used as an encoder, allows to solve the difficulty in training very deep neural network; these problems are the *Vanishing Gradient* problem and *Exploding Gradient*.

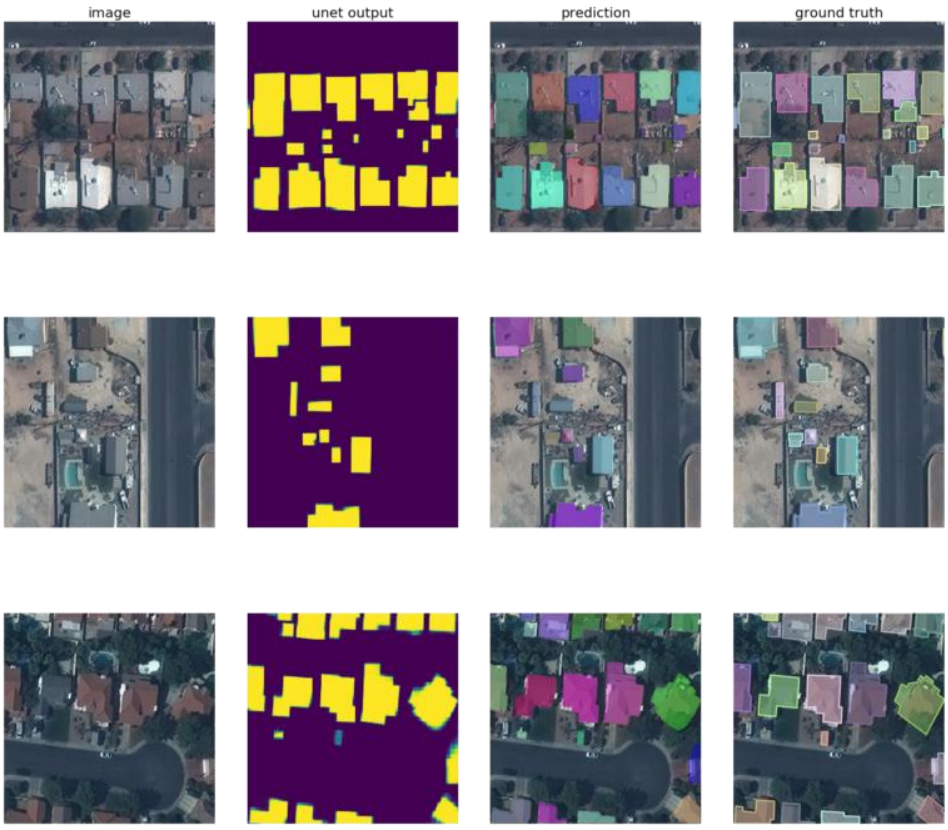

Figure 37: Output of Deep Neural Network.

4.1 THE VANISHING GRADIENT PROBLEM

The *Vanishing gradient* problem is caused by the large number of layers, that use certain activation functions, make the gradients of loss function approaches zero, making network complex to train. In *Chapter 2* the concept of cost function was introduced, explaining how to calculate it and why it is so important for training a neural network. The terms *cost* and *loss* functions almost refer to the same meaning, but *loss* function mainly applies for a single training set as compared to the cost function which deals with a penalty for a number of training sets or the complete batch. So, it is possible to define the *loss* function as a component of the *cost* function, in fact the *cost* function is calculated as the average of the *loss* functions, which are computed at every instance.

To give a clear and efficient description of the *Vanishing gradient* problem, consider *Figure 38* where the sigmoid function and its derivative are represented. From *Figure 38* can be observed that for large values of the sigmoid function correspond very small values of the derivative. When more layers are used, it can cause the gradient to be too small for training to work effectively.

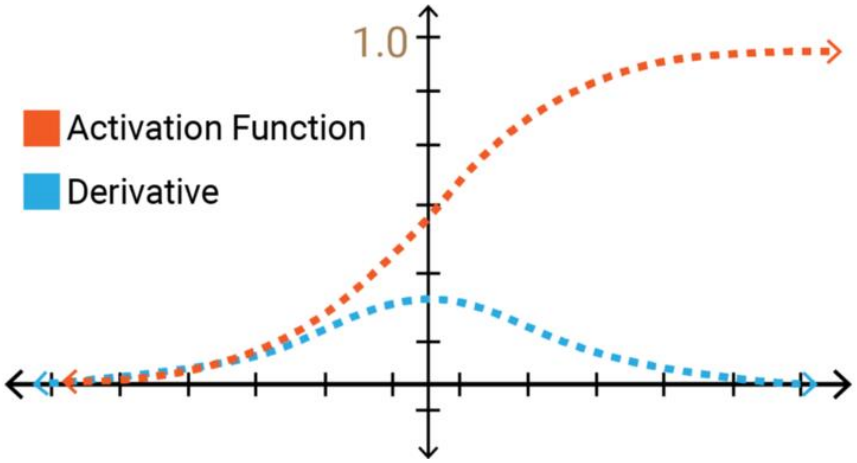

Figure 38: Sigmoid Function and its derivative.

For gradient calculation, an algorithm called *Backpropagation* is used, this is used with the purpose of making sure that all changes that minimize the *cost* function, involve all the weights and biases of all layer together. Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one. By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers [32]. If more than one layer uses the sigmoid function, a number of small derivatives are multiplied together, then the gradient decreases exponentially as we propagate down to the initial layers. Small gradient means that the weights and biases of the initial layers of the network are not effectively updated within each training session. As said in the previous paragraphs, the first layers of a neural network prove to be of fundamental importance because they are able to recognize the key elements of the input data, so their incorrect functioning can lead to overall inaccuracy of the whole network. A possible solution to this problem might seem to be the use of different activation function such as *ReLu Figure39,* but it will be explained how *ReLu* also presents gradient vanishing problem.

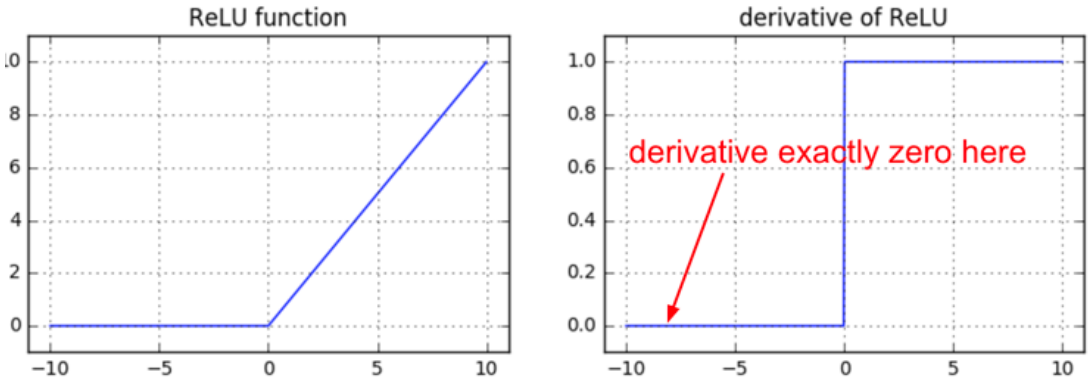

Figure 39: ReLu function and its derivative.

As you can see from the images above right, the derivative of ReLu can be either 1 or zero. So multiplying the derivatives by several layers will never achieve the same degradation in the case of the sigmoid function. There is only one problem with the ReLU activation – sometimes, because the derivative is zero when the input is less than zero, certain weights can be "*killed off*" or become "*dead*". This is because the back-propagated error can be cancelled out whenever there is a negative input into a given neuron and therefore the gradient will also fall to zero. This means there is no way for the associated weights to update in the right direction. This can obviously impact learning [33].

4.2 EXPLODING GRADIENTS

Exploding gradients problem occurs when large error gradients are accumulated and this result in very large updates to neural network weights during training. When this kind of situation occurs, what you get is an unstable neural network whose training will never be completed.

4.3 RESNET

ResNet is a Convolutional Neural Network (CNN) architecture, it is called "*micro-architecture*" or even "*network-in-the-network-architecture*", and it consists of a series of *building blocks*. This type of architecture uses, the previous discussed, *skip connection* to take the input from one layer and feed it to on another layer even much deeper in the neural network. *ResNet* gives the opportunity to train very deep networks.

Residual block are the building blocks of *ResNet*, to better understand how the residual block works, consider the scheme in *Figure 40*.

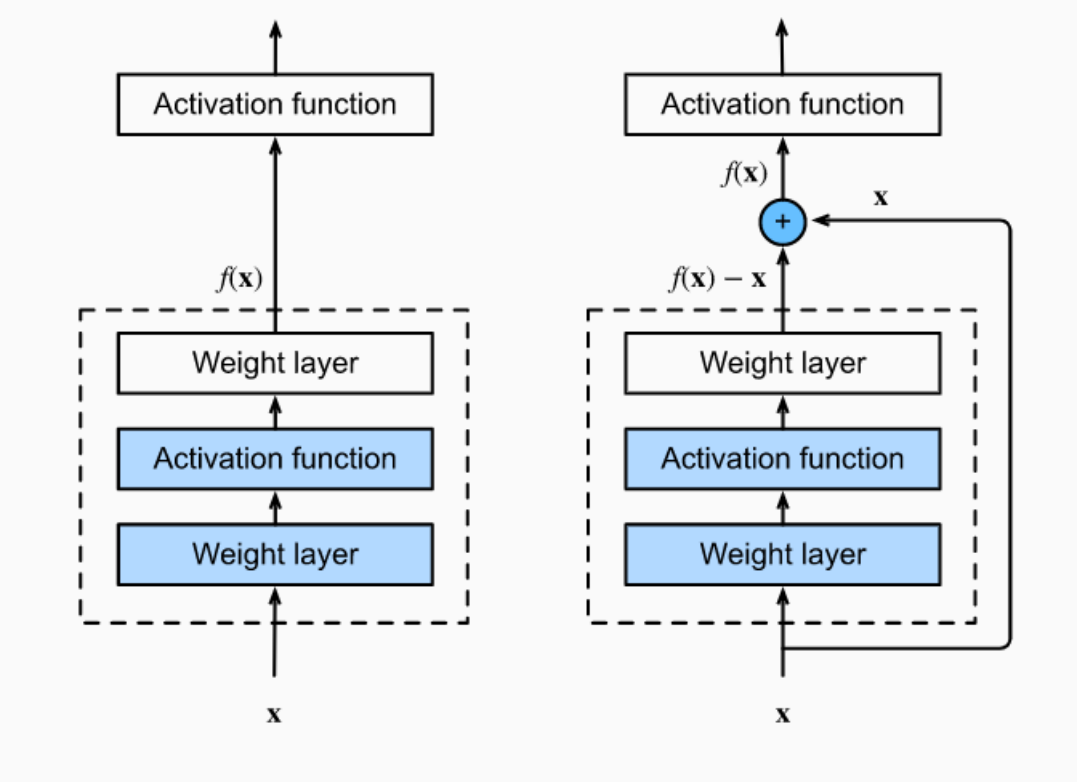

Figure 40: Difference between a regular block (left) and a residual block (right).

To the left is a standard block, to it is applied an input x, the portion within the dashed lines learns the learning mapping $f(x)$ which will later be used as input for activation function on the top. On the right the residual block of *ResNet* is shown, here you can notice a bold line that brings the input to an addition operator, in this case the portion inside the dotted line will have to learn the residual mapping

$f(x) - x$, hence the residual block name.

To implement the sum between $F(x) + x$, the x input size must be equal to the size of the residual mapping. It is possible then define two types of residual blocks:

- *Identity Block*: If the input and the residual mapping have the same dimension (*Figure 41);*
- *Convolutional Block:* A convolutional block is added in the shortcut path to modify the dimension of the input x (*Figure 42*).

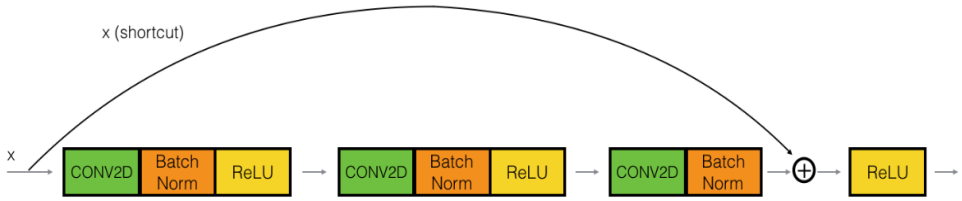

Figure 41: Identy block structure.

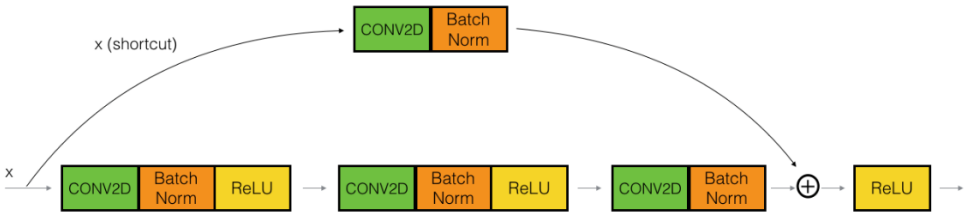

Figure 42: Convolutional block structure.

Thanks to the use of a *ResNet* network it is possible to train very deep neural networks, even with more than 100 layers. Using this architecture, you can get a very low training error as the number of layers in the network increases. The same argument is not valid in the case of a standard network, where initially a decrease in training error is observed as the number of layers in the network increases, and then suddenly increases, causing a degradation in network performance (*Figure 43)*.

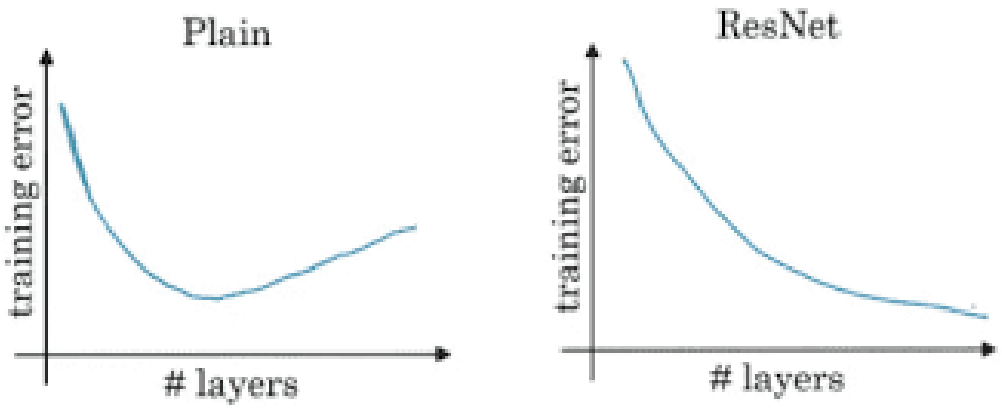

Figure 43: Training error trend as number of layers changes.

From the number of residual blocks that are stacked together, you can get several variants of the *ResNet* that are identified by the number of layers, such as ResNet34, ResNet101, ResNet152 to name a few. In the following table will show the detailed specifications for each of these architectures.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 2: ResNet architecrure specifications.

As mentioned above, the *ResNet* is used as an encoder of a U-net architecture, the best results are obtained using ResNet101 architecture. The architecture of the residual network *ResNet101*, is reported in *Figure 44* in order to give an understandable and detailed description. Each Block indicates

the convolution operation and channel width associated. The network is having number residual connections. The convolution layers are distributed into 5 Sections each of them with a different colour. The first block consists of only a single unit. Second block consists of 9 units, third block has 12 units, fourth block has 69 units, and fifth block consists of 9 units. The detailed specifications of each block are as shown in *Table 4.2*.

Layer Name	Output Size	No. of Units
Convolution Layer 1	112x112	[7x7, stride 2, channel 64] x 1
Convolution Layer 2	56x56	[3x3 max pool, stride 2] x 1 [1x1, channel 64] x 3 [3x3, channel 64] x 3 [1x1, channel 256] x 3
Convolution Layer 3	28x28	[1x1, channel 128] x 4 [3x3, channel 128] x 4 [1x1, channel 512] x 4
Convolution Layer 4	14x14	[1x1, channel 256] x 23 [3x3, channel 256] x 23 [1x1, channel 1024] x 23
Convolution Layer 5	7x7	[1x1, channel 512] x 3 [3x3, channel 512] x 3 [1x1, channel 2048] x 3
Output	1x1	Average pool, 1000d-fc, soft-back

Table 3: ResNet 101 specifications.

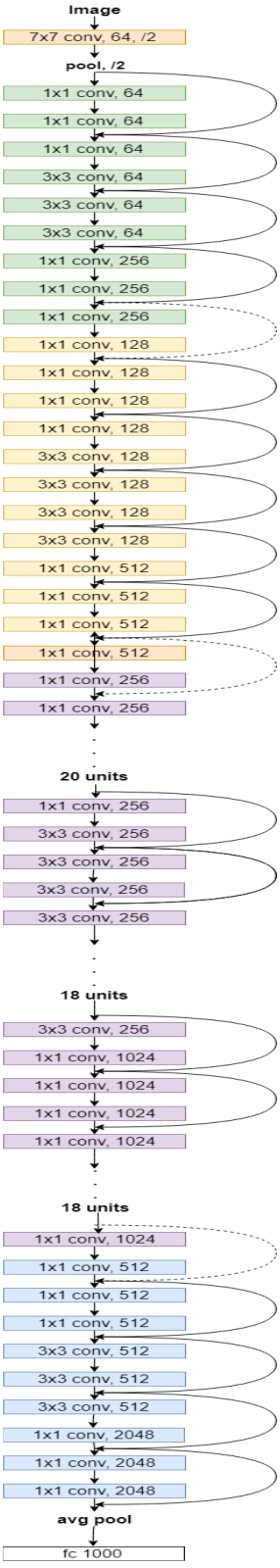

Figure 44: ResNet 101 architecture.

4.4 TRAINING

The model is trained using a Dataset provided by the organization of the *Mapping challenge competition,* the dataset is composed by more than 400000 tiles (as *300x300* pixel *RGB* images). Dataset is divided into two different *Datasets section*, they are:

- *Training Set*: Training set includes 280741 tiles (as 300x300 pixel RGB images) of satellite imagery, along with their corresponding annotations in MS-COCO format.
- *Validation Set*: Validation set includes 60317 tiles (as 300x300 pixel RGB images) of satellite imagery, along with their corresponding annotations in MS-COCO format.

The training scheme followed by the network's team of developers is described here to understand the training strategy used and how they acted on the *hyperparameters* and *parameters* of the model. Before proceeding with the description of the training process it is essential to define the difference between *hyperparameters* and *parameters* of a model.

- *Model parameters:* They are the parameters that are estimated by the model from the data at its disposal, for example the *weights* of a deep neural networks.
- *Model hyperparameters*: They are all those parameters that cannot be estimated by the model using the available data, for exemple: *learning rate*, *number of epochs* and *batch size*.

Choosing the right combination of hyperparameters is crucial if you want to get maximum performance out of the models.
The first step in the training process is the initialization of the model with pre-trained weights. Having good initial weights can place the neural network close to the optimal solution. This allows the Neural network to come to the best solution quicker. After the initialization, the model was trained using a dataset of 50k images, using a learning rate equal to 0.0001 and weighs 0.5. After this first training, the model was re-trained this time using the complete dataset and keeping the learning rate values and network weights unchanged. Once this second training is finished, the *learning rate* is reduced to *0.00005* and *weight* equal to the previous values, to optimize the performance of the model.
Hyperparameters that have previously been reported included *batch size* and the *number of epoch*. The first indicates the number of samples that are contained in a single batch to be processed by the network at once, while the second indicates the number of times that all data must be processed by the model before completing the total training process. Both are very important in the training phase of a neural network, depending on those two hyperparameters the training procedure can last hours or days. In the *Figure 45* the hyperparameters used to obtain the best performances are reported.

```
Training schedule
epochs_nr: 100
batch_size_train: 20
batch_size_inference: 20
lr: 0.00005
```

Figure 45: Hyperparameters used to train the model.

4.5 SHELTERING FACTOR COMPUTATION

The computation of the sheltering factor represents the main contribution of this thesis work. The use of the pre-trained network, described in the previous paragraphs, makes it easy to create an algorithm capable of calculating the sheltering factor of a given area.

The first step is to create a DATASET of images that the model will process. Through the access with RestFul architecture on the Bing Maps API, satellite images are collected and saved in a specific folder. To obtain customized images that meet the requirements of the model used, different parameters must be configured such as: pixel resolution, surface size of the imagine and resolution distance. These parameters are contained within a .json file, which is used precisely for the exchange of data between the client and the server. In *Figure 46* the .json file used to set the parameters of the acquired image is show, where "*mapsize*" is the pixel resolution of image and the "*Celldimension*" are the dimension of the geographical surface of the image.

Figure 46: Parameters of the customized images within .json file.

```
{
        "baseLink":
"https://dev.virtualearth.net/REST/V1/Imagery/Map/Aerial",
        "query": {
                "mapLayer": "Basemap,Buildings",
                "imagerySet": "AerialWithLabels",
                "mapSize": "300,300",
                "key": "Token"
        },
        "cellDimension": {
                "Height": 100,
                "Width": 100
        },
        "savingPath": "Path",
        "resolutionDistance": 0.001
}
```

After completing the capture of the images, each of them is cropped to remove any tag present that may alter the calculation of the sheltering factor. In the following images, you can see the difference between the original image and the cropped image.

Figure 47: Aerial image of Turin before crop.

Figure 48: Aerial image of Turin after crop.

After these pre-processing procedures are completed, the built DATASET will be processed by the model. To process these data the model will use a technique called Test-Time Data Augmentation during inference. Using this technique inference is performed on multiple altered versions of the same image, the obtained predictions are then aggregated to get higher overall accuracy (*Figure 49)*.

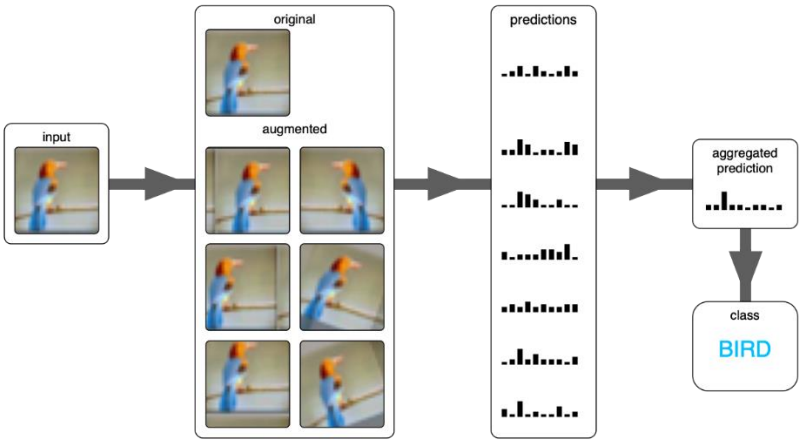

Figure 49: Test-Time Data Augmentation technique.

An annotation file in .json format is created for the prediction of each image according to the COCO dataset Format. Using the function API "*decode*", it is possible to obtain the binary mask of each image *Figure 50*.

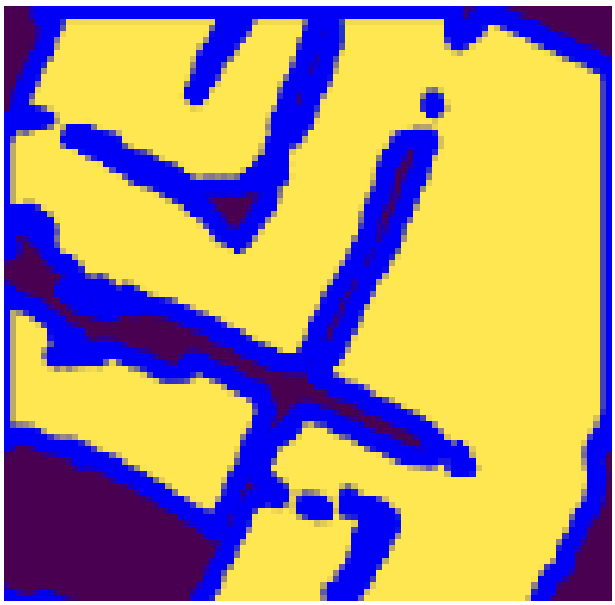

Figure 50: Binary Mask of an aerial image.

The binary mask is a NUMPY. NDARRAY of size 300x300, which contains Boolean values (*TRUE & FALSE*). For the computation of the total area occupied by the buildings in the image, it is necessary to count all the elements, which correspond to a building, contained within the mask matrix. Using the area occupied by all the individual buildings in the image, it is possible to calculate the ratio of the total area of the image to that occupied by the constructions. In the next images this value is considered as the ratio of the red area over the total area of the image.

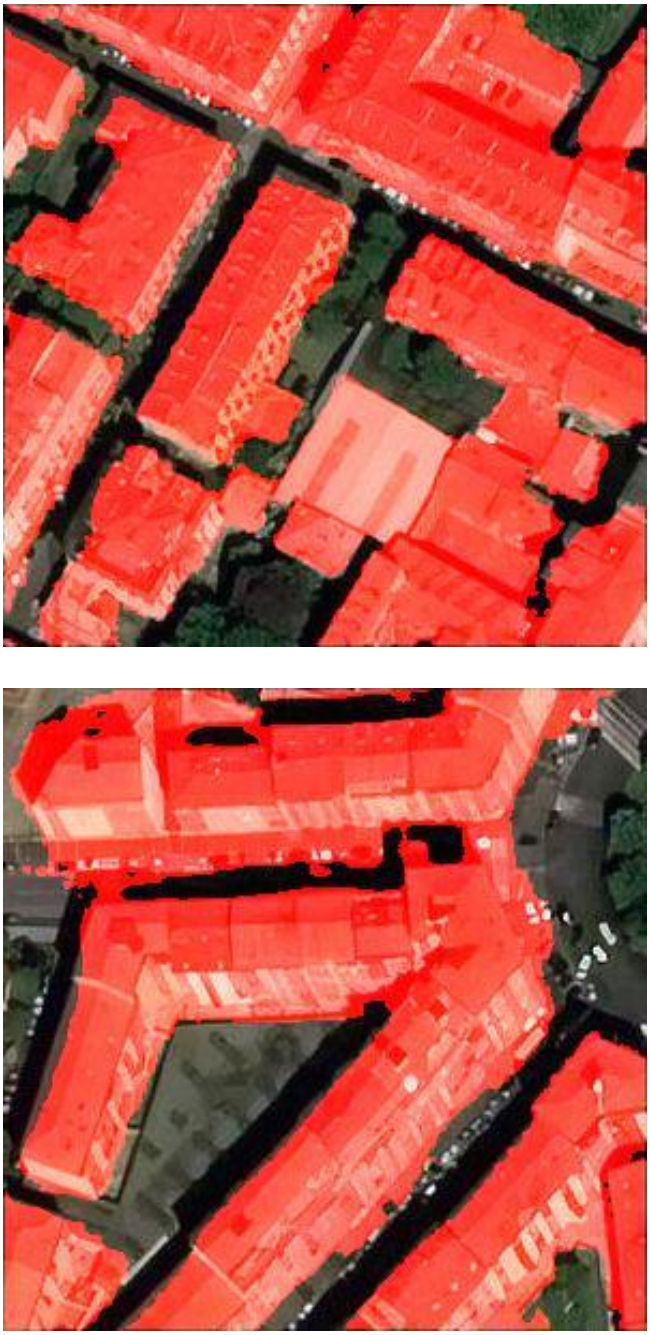

Figure 51: Prediction of aerial image examples.

To proceed with the calculation of the sheltering factor, a range of values between 3 and 8 is defined. These two values correspond to the average values that are assigned to an area. A sheltering factor equal to 3 defines an area where on average there is no presence of buildings, but people are sheltered by trees, cars and other objects. A sheltering factor equal to 8 is assigned to areas where the presence of buildings is consistent. The difference between these two values, is multiplied by the ratio of the

two areas defined above, the result obtained from this product is subtracted S_{low}. To sum up the above, the following formula is given:

$$S_F = (S_{high} - S_{low}) * \frac{A_c}{A_{Tot}} + S_{low}$$

- S_{high} = 8, it represents the maximum bound of the Sheltering Factor;
- S_{low} = 3, it represents the minimum bound of the Sheltering Factor;
- A_c = Area covered by the buildings;
- A_{Tot} = Total area of the aerial image.

The formula above is used to compute the average *S* of three different areas of Turin. The resulting values will be than compared with those obtained using the method discussed in the *Chapter1*.

Chapter 5

FINAL RESULTS

In this chapter, the average Sheltering Factor values obtained will be analyzed and discussed. As already mentioned in *Chapter 1,* Turin represents the ideal testbed for the proposed AI-based strategy, the model will be evaluated on three different datasets of images, indicated as Turin 1, Turin 2, Turin 3. Each dataset contains images that refer to areas with precise characteristics, which have the task of highlighting the weaknesses and strengths of the model used in this thesis.
For a complete analysis and evaluation of the AI-based strategy, used for the determination of the Sheltering Factor, the values obtained will be compared with those obtained using the method described in *Chapter 1*. The method discussed in the first chapter uses a three-dimensional map of the city of Turin extracted from the Open Street Map. To measure the closeness of agreement between the measured values of the two method, and to evaluate which of the two guarantees the best performance depending on the scenario considered, different statistical elements will be used. The statistical elements used for the method comparison are: *Correlation Factor*, *Scatter Plot* and *Difference plot* (Bland-Altman plot).

5.1 CORRELATION FACTOR

Correlation coefficient is a numerical measure of some type of correlation, meaning a statistical relationship between two variables [34]. It is an index of the strength of linear association between two methods. The correlation coefficient is between -1 and 1 and in particular:

- Correlation Factor > 1: the data series, obtained by their respective methods, are said to be directly related, or positively correlated, and the closer the value of this index is to 1, the stronger the positive correlation;
- Correlation Factor = 0: the data series are not correlated;
- Correlation Factor < -1: Data sets are said to be inversely correlated or defined as negatively correlated, and more the value of the index tends to -1, stronger is the negative correlation.

The correlation coefficient is one of the most used reported statistic in method comparison studies. Depending on the area considered and therefore from the scenario considered, this index will take on several values that will be commented on later.

5.2 SCATTER PLOT

Scatter Plot is a mathematical diagram using Cartesian coordinates used to show the relationship between two different methods. Using a scatter plot is possible to recognize if the data concentrates around some curve, and then define a possible "law" that links the values on the axis of the axle with the values shown on the coordinate axis. In the scatter plots that will be shown in the next paragraphs, the values obtained using the three-dimensional map of the city on the horizontal axis will be reported, and the values obtained in this thesis on the vertical axis. Depending on the shape taken by the point cloud it is possible to make assumptions about the relationship between the two methods. In *Figure 52* will be shown the types of correlation possible between the two methods.

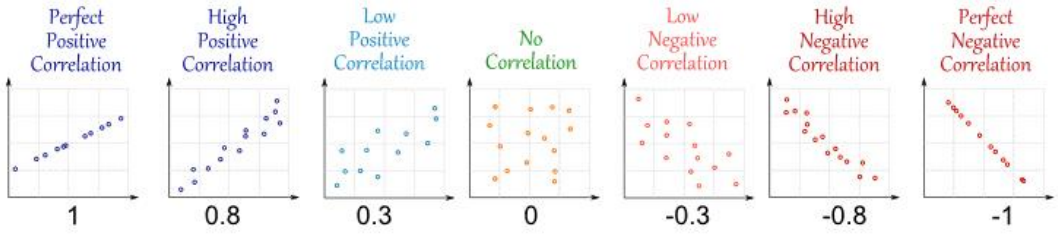

Figure 52: Relationship scatter plot examples.

5.3 DIFFERENTIAL PLOT (BLAND-ALTMAN PLOT)

The Bland-Altman plot is used to compare two measurements of the same nature; it is a diagram of dispersion in which the difference of the two measures is reported on the ordinates and the measure of reference is reported on the abscises, obtained as an arithmetic mean of the two measures. We Consider two sets of quantitative data, obtained using the two comparison methodologies, and each of which consists of *n* samples.

X_1 indicates the Sheltering Factor value obtained by using the deep learning network and X_2 the value obtained using the available data OSM. From this pair of measures, it is possible to obtain two more quantities:

- The average of the two values for each sample *n;*

$$\bar{X}_i = \frac{X_{1i} - X_{2i}}{2}$$

- The difference (*d*) between the two values for each of the *n* samples;

$$d_i = X_{1i} - X_{2i}$$

The Bland-Altman plot of *Figure 53* is formed by plotting the difference d_i on the vertical axis versus the averages $\bar{X}_i$ on the horizontal axis. A horizontal line representing the bias is drawn at $\bar{d}$ and two additional lines, known as *limits of agreement,* are added to plot at $\bar{d}$ - 1.96S_D and $\bar{d}$ + 1.96S_D, where S_D is the *standard deviation,* these two lines constitute the confidence interval. In statistic, the standard deviation is a measure of the amount of variation or dispersion of a set of values. The value of $\bar{d}$ indicates whether one of the two methods underestimates or overestimates the index relative to the

other. If the points in the graph are within the limits of agreements, the two methods are considered to provide consistent results.

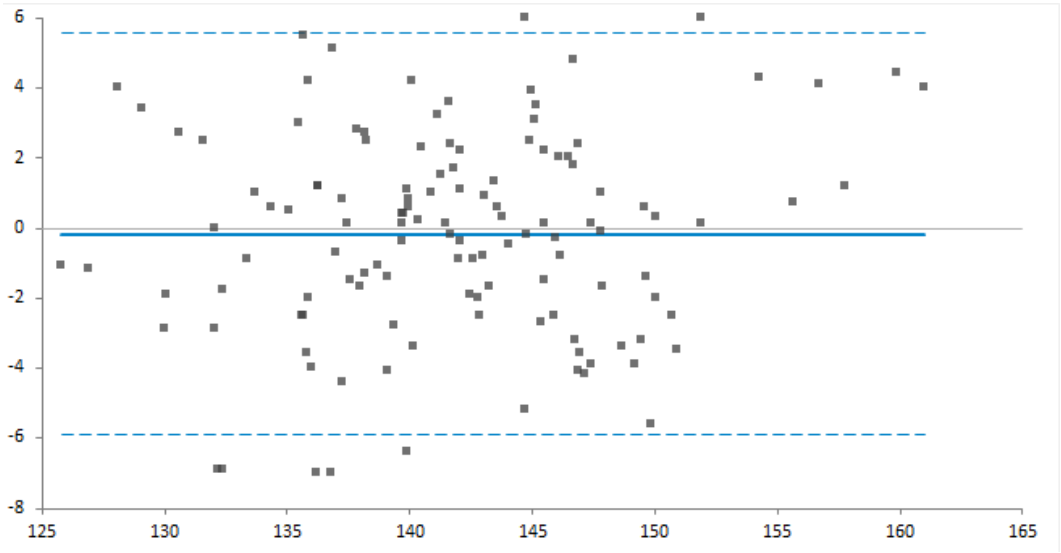

Figure 53: Differential (Bland-Altman) Plot.

The use of this plot with the support of masks generated by the deep neural network, will serve to understand when the strategy proposed in this work provides more real and accurate data than the method described in *Chapter 1*.

5.4 TORINO 1

This dataset is composed by 160 aerial images depicting the city center of Turin (*Figure 54)*. It is easy to understand that the high presence of buildings guarantees a shelter to the populations in that area, so it is normal to get high sheltering factor values.

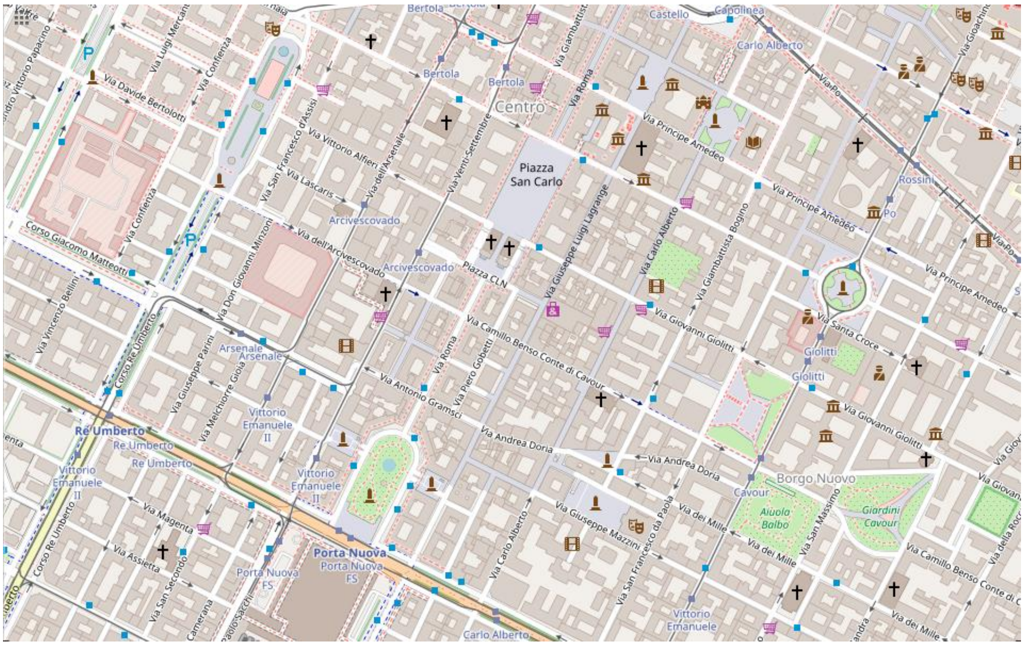

Figure 54: Map representing the "Torino 1" scenario from Open Street Map.

In this scenario, both methods appear to have the same behaviour, because an increase in the values of one corresponds to an increase in the values of the other.

The above can be observed from the scatter plot of *Figure 55,* where on the x-axis the values obtained in the previous work are reported while on the y-axis the values of Sheltering Factor obtained in this work.

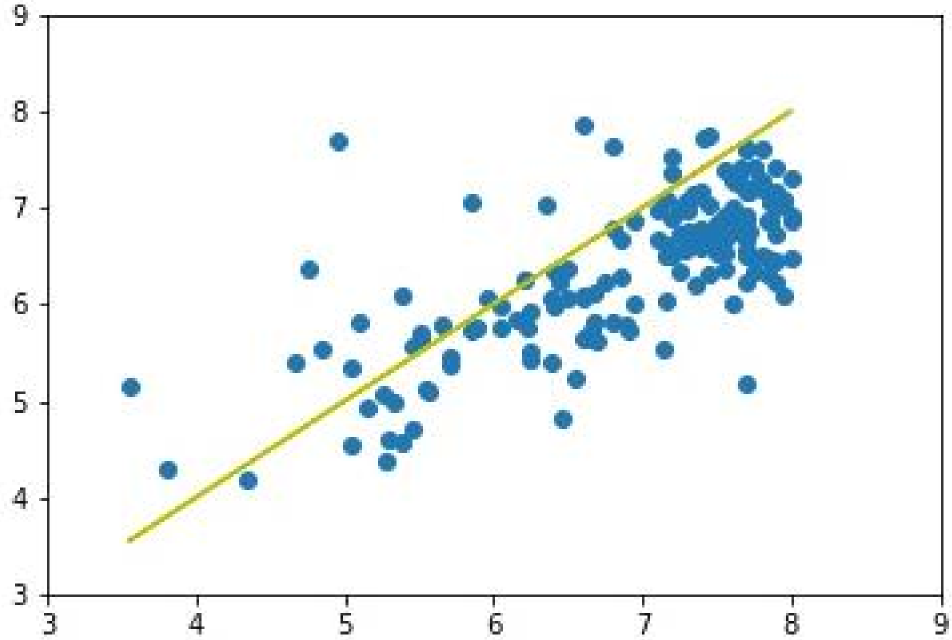

Figure 55: Scatter plot of results obtained in the "Torino1" scenario.

The points form a constant-width band around the identity line, values present a relationship that is defined as moderate, positive, and linear. The linearity of the relationship between the two different sets of values is confirmed by the value of the correlation coefficient 0.7210, the fact that this factor is very high indicates that the two sets of values are strongly related.

Now the differences between the two methods will be analysed in detail, for this purpose the Bland-Altman is reported here.

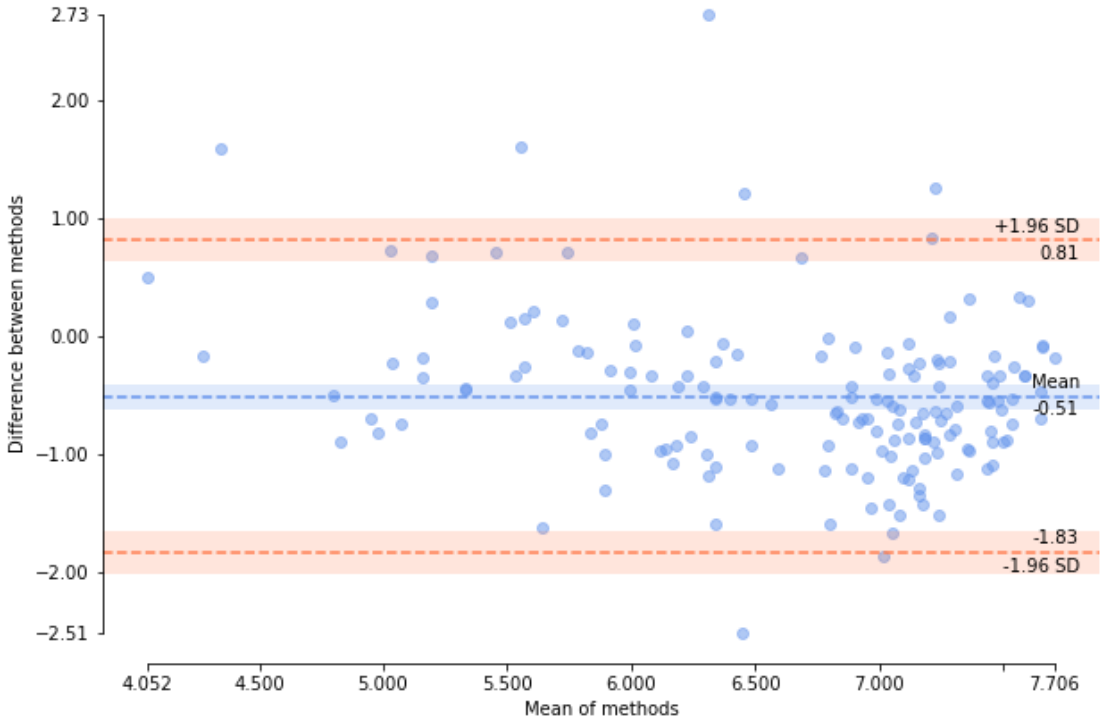

Figure 56: Differential plot of the results obtained in "Torino1" scenario.

The most important information in the graph is provided jointly by the average value -0.51 and the confidence interval of the differences at 95%, which varies between -1.83 and 0.81. The negative value of the average difference between the two methods indicates that the proposed AI-based method underestimates the average value of the Sheltering Factor, compared to the other method. Considering the confidence interval, you can observe how more than 95% of the points are within it, so it can be said that in this scenario the two methods are interchangeable. Only a few points are outside of the confidence interval, in these cases the two methods are not congruent to each other. To better understand what happens at these points, two notable points are considered, namely the maximum and

minimum of the differential plot. The maximum value of the difference between the two methods is equal to 2.7291 and corresponds the area of *Figure 57*.

Figure 57: Aerial image of the area with maximum difference between methods in "Torino 1" scenario

The value of such a high difference is because the deep neural network mistakenly considers the square present in the imagine as a building and, thus, altering the final value. This is clearly visible in the following image, where the prediction of the model is reported.

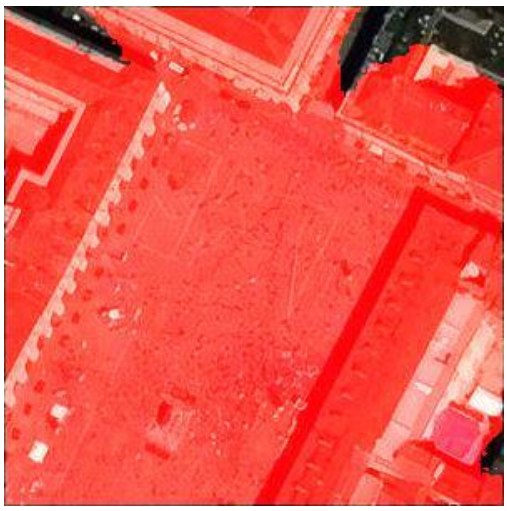

Figure 58: Prediction of area with wrong average Sheltering Factor 7.679 in "Torino 1" scenario.

The other example as mentioned above concerns the minimum difference between the two methods which is equal to -2.5095, in this case the proposed model underestimates the average value of the sheltering factor, compared to the other method. This is because the model in this area has not been shown to be able to locate all the buildings present in *Figure 59*.

Figure 59: Aerial image of the area with minimal difference between methods in "Torino 1" scenario.

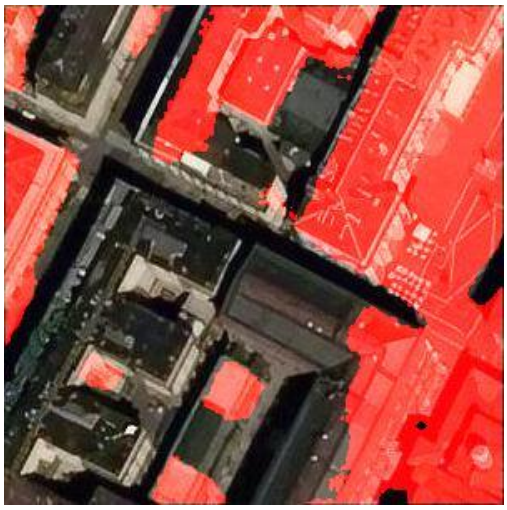

Figure 60 :Prediction of area with wrong average Sheltering Factor 5.1904 in "Torino1" scenario.

The model used in this work has proven to be very performing on this dataset of images. The values obtained from it appear to be in line with those obtained using OSM, and this is indicated by the very low value of $\bar{d}$. The similarity of the values obtained from both methods is most evident from the sheltering layers in the following figures.

Figure 61: Sheltering layer obtained using OSM for "Torino 1" scenario.

Figure 62: Sheltering layer obtained using deep neural network for "Torino 1" scenario.

The proposed method proves to be more restrictive on average than that discussed in *Chapter 1*, this is evident from the comparison of risk maps obtained using both approaches. In the case of the AI-based strategy higher risk factors are obtained, therefore values in the colour scale that are closer to red, which represents the maximum risk factor (*Figure 63)*.

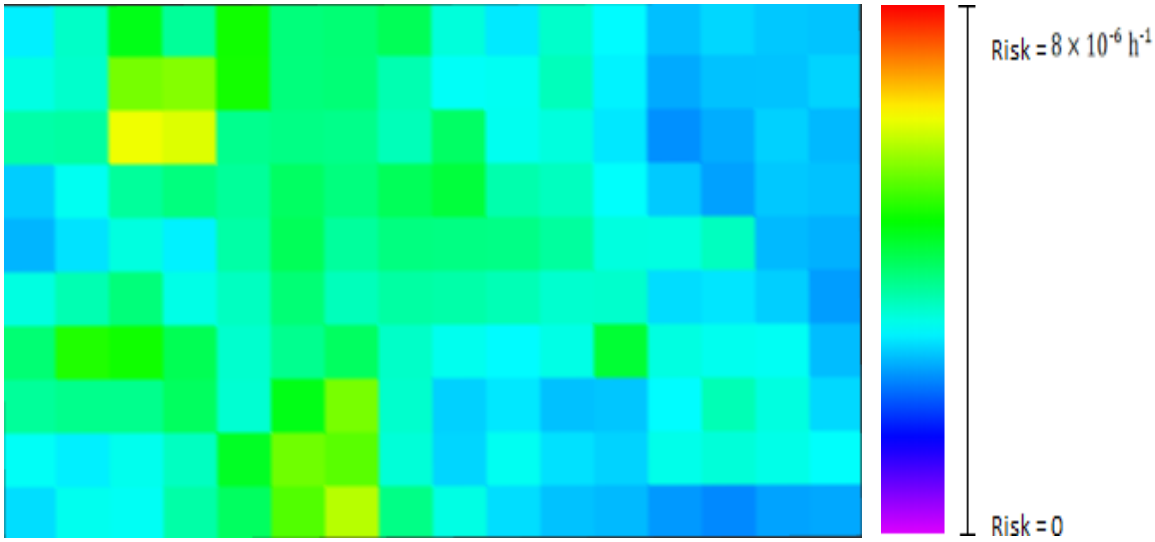

Figure 63: Risk map obtained using OSM for "Torino 1" scenario.

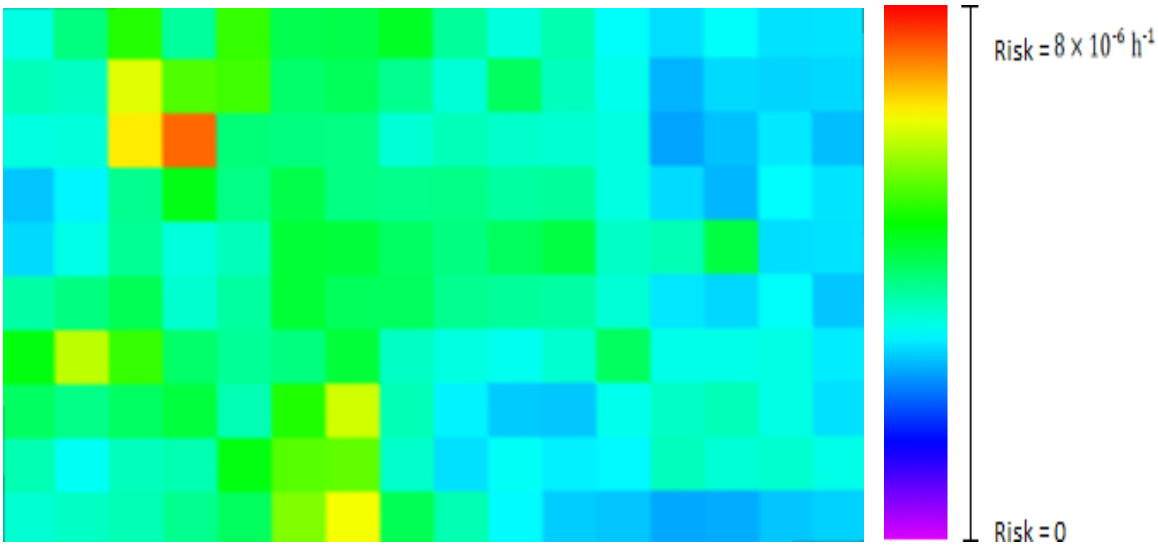

Figure 64: Risk map obtained using deep neural network for "Torino 1" scenario.

The use of the risk map obtained using the sheltering layer shown in *Figure 64*, limits UASs operations in these specific areas represented in the images that are included in the dataset. In the scenario of Torino 1, the sheltering factor values obtained using OSM can be considered as reference values. The fact that the values obtained by both methods are similar is to be considered a very good result. *Figure 65* and *Figure 66* will show the prediction obtained by the deep neural architecture in the case of Torino 1.

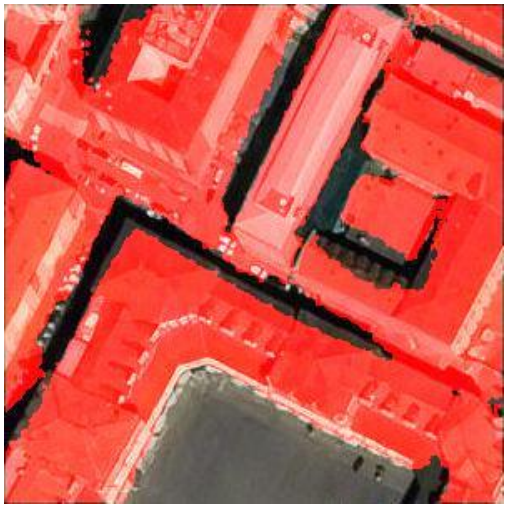

Figure 65: Prediction of area with average Sheltering Factor 6.7617 in "Torino1"scenario.

Figure 66: Prediction of area with average Sheltering Factor 5.7898 in "Torino1" scenario.

5.5 TORINO 2

This dataset is composed by 220 aerial images, the scenario presents in these images is very different from the one described above, now in fact the city center of the previous dataset leaves room for more peripheral areas of the city of Turin.

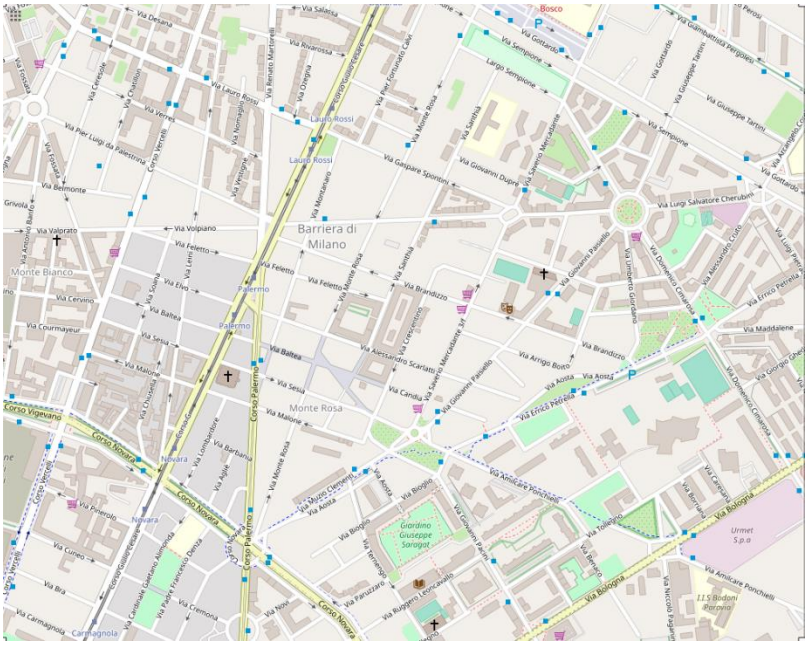

Figure 67: Map representing the "Torino 2" scenario from Open Street Map.

In these areas the presence of housing is no longer so frequent, so what will be observed is a great diversity of values. The range of value obtained in this case is much wider than previously seen, this can easily be guessed by scatter plot in *Figure 68.*

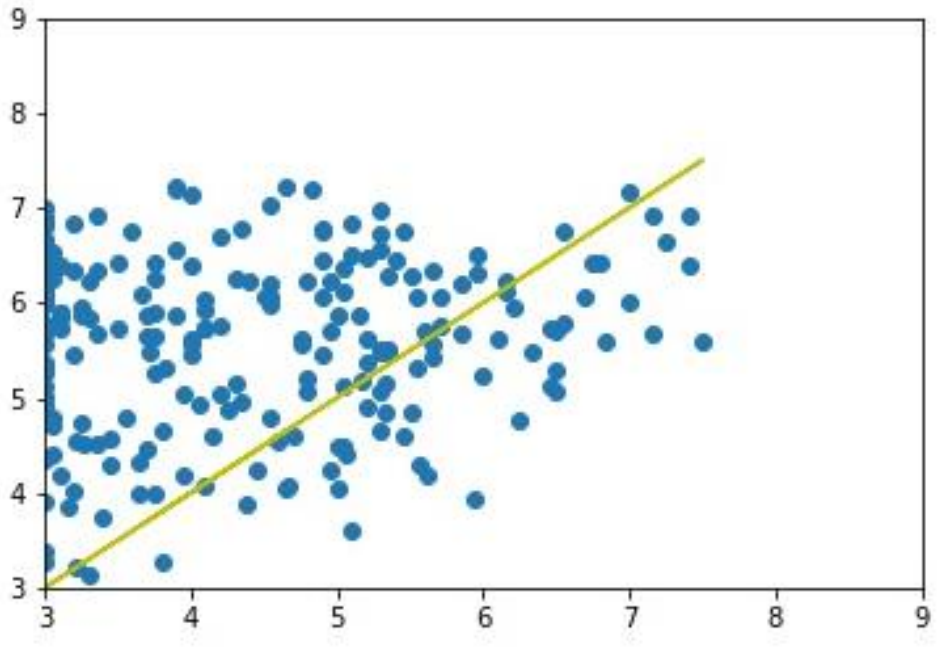

Figure 68: Scatter plot of results obtained in the "Torino2" scenario.

As can be seen from scatter plot above, there is no relationship between the two methods, this is confirmed by the value of the correlation coefficient which turns out to be very small 0.1030.

To highlight the differences between the two methods, the differential plot is reported.

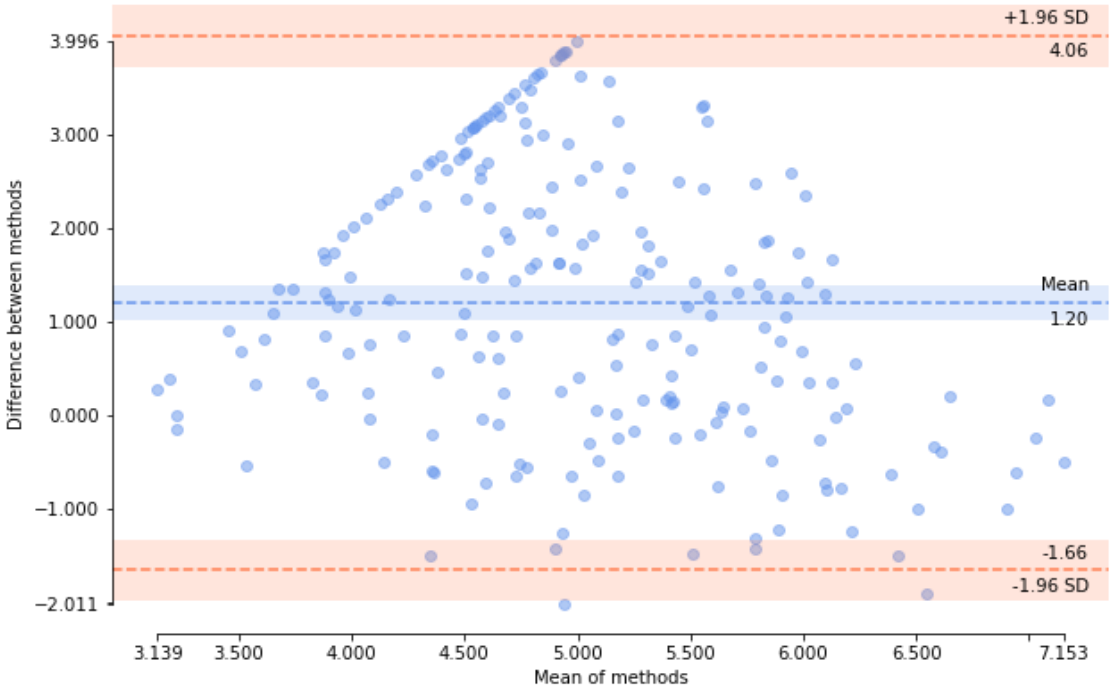

Figure 69: Differential plot of the results obtained in "Torino2" scenario.

Unlike the last set of values, in this case the difference between the values obtained, using their respective methods, begins to be relevant. The value of the average difference between the two methods is equal to 1.20 this means that the model systematically overestimates the value of the average sheltering factor with respect to the other method. Again, most points are within the confidence range, so the two methods might be considered interchangeable, but this is not entirely correct, in fact the variation of difference between method within the confidence range is not negligible. Considering methods comparable is particularly dangerous, because using one rather than the other would run the risk of assigning incorrect sheltering factor values that would limit the flight of UASs in these areas or underestimate the danger of a possible vehicle crash.

The difference between the value sets is due to the inefficiency of the method that uses OSM, to obtain the average sheltering factor of these peripheral areas of the city, in which many buildings are not marked in OSM. The AI-based method proves capable of obtaining results consistent with the definition of sheltering factor given in the previous chapters. To demonstrate how much better the AI-based method is in this scenario than the method that use OSM, the following examples are considered.

Consider the areas where the difference between the two methods is maximum $\overline{d} = 3.9960$ and where it is minimal $\overline{d} = 2.0112$.

Figure 70: Aerial image of the area with maximum difference between methods in "Torino 2" scenario.

Figure 71: Aerial image of the area with minimal difference between methods in "Torino 2" scenario.

The model discussed in this thesis assigned a value of 6.1470 to the area of *Figure 72* and a value of 4.3056 to the area of *Figure 73*. With all available data it is possible to determine the average sheltering factor value assigned to each of these areas using the three-dimensional map of Turin, the first area has been assigned a value of 2,151 while the second area has been assigned a value of 2,294. These two values are well below the range (3 to 8) of the values considered for the calculation of the average sheltering factor, moreover these values are typical of scenarios in which there is no presence of any

building, as however shown in the following images, of the predictions, in both areas there is a considerable presence of buildings.

Figure 72: Prediction of area with average Sheltering Factor 6.1470 in "Torino 2" scenario.

Figure 73 Torino 2- Prediction of area with average Sheltering Factor 4.3056 in "Torino2" scenario.

In the following images the sheltering layers obtained using both methods are reported. It is evident that in the case of the use of the deep neural network the values in the grayscale are closer to black this is because, as mentioned above, this approach systematically overestimates the value of the sheltering factor compared to that obtained using OSM.

Figure 74: Sheltering layer obtained using OSM for "Torino 2" scenario.

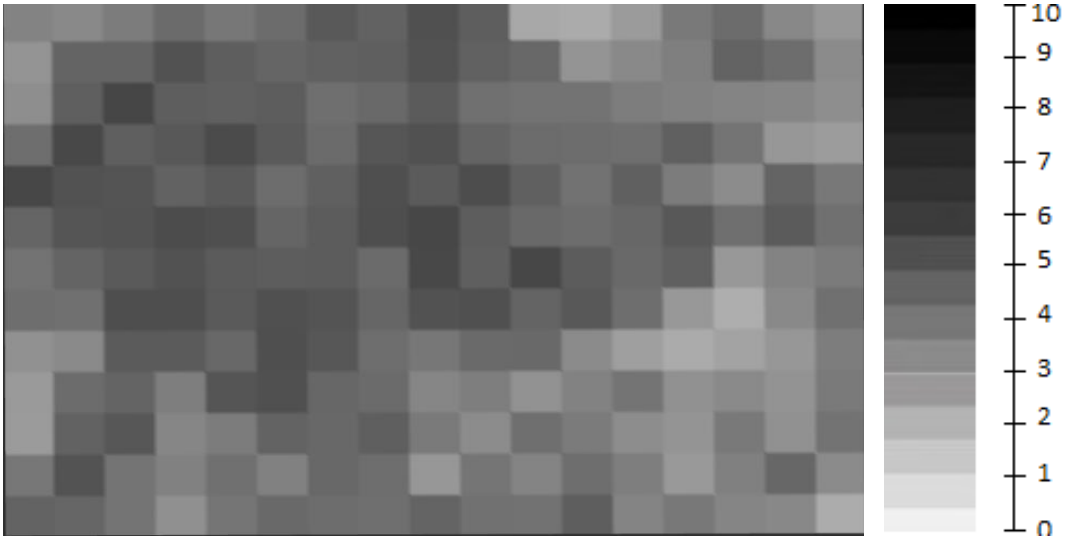

Figure 75: Sheltering layer obtained using deep neural network for "Torino 2" scenario.

The Sheltering Factor values obtained, for images belonging to this dataset, are accurate and consistent with the number of shelter elements in the area. As a demonstration of the better performance achieved by the AI-based strategy than those obtained using OSM, several images of predictions of the model will be shown. These images will highlight the ability of the deep neural network to extract buildings also in particular scenario.

Figure 76: Prediction of area with average Sheltering Factor 4.4913888888888884 in "Torino 2" scenario.

Figure 77: Torino 2- Prediction of area with average Sheltering Factor 4.1947 in "Torino 2" scenario.

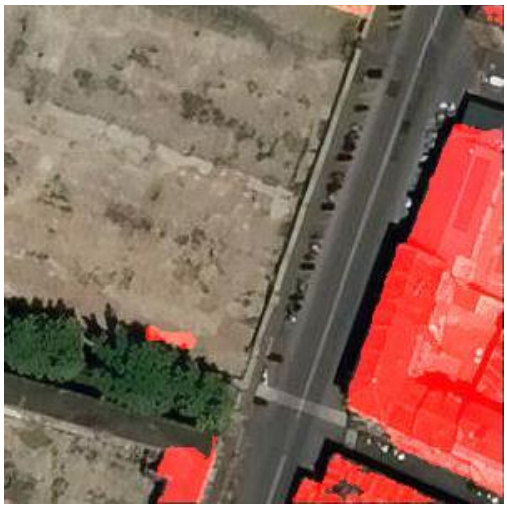

Figure 78: Prediction of area with average Sheltering Factor 3.886 in "Torino 2" scenario.

The discrepancy in the sheltering factor values obtained is reflected in the risk maps obtained, so it is clear that the two methods cannot be defined as interchangeable. The inaccuracy of the values obtained using OSM means that we obtain a risk map which is not true, and incapable of quantifying the real risk (*Figure 79)*. The risk map obtained using the sheltering layer in *Figure 75*, reflects the real scenario present in the area Torino 2, so it guarantees an accurate assessment of the risk (*Figure 80*).

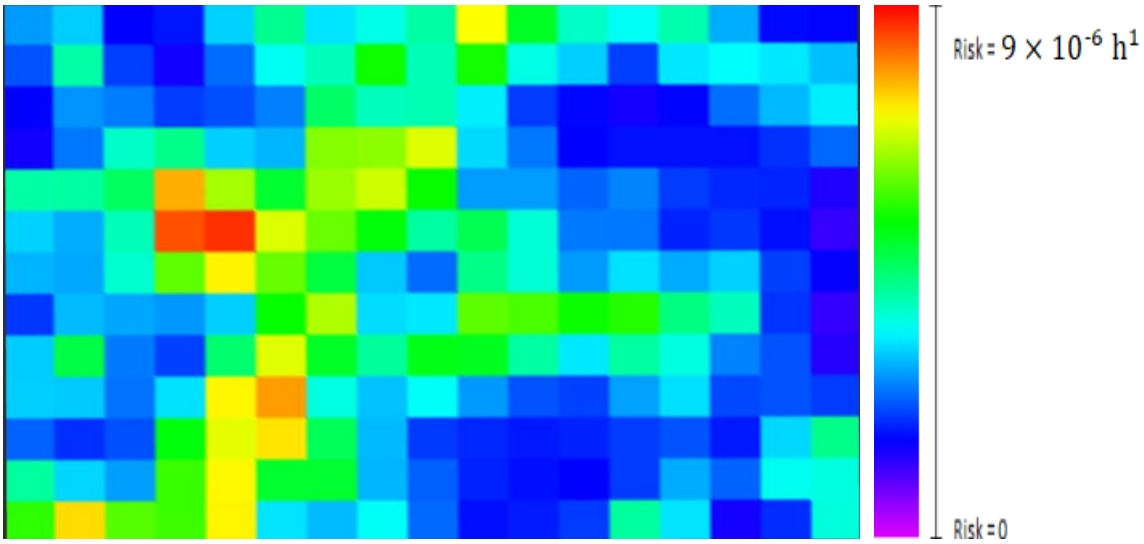

Figure 79: Risk map obtained using OSM for "Torino 2" scenario.

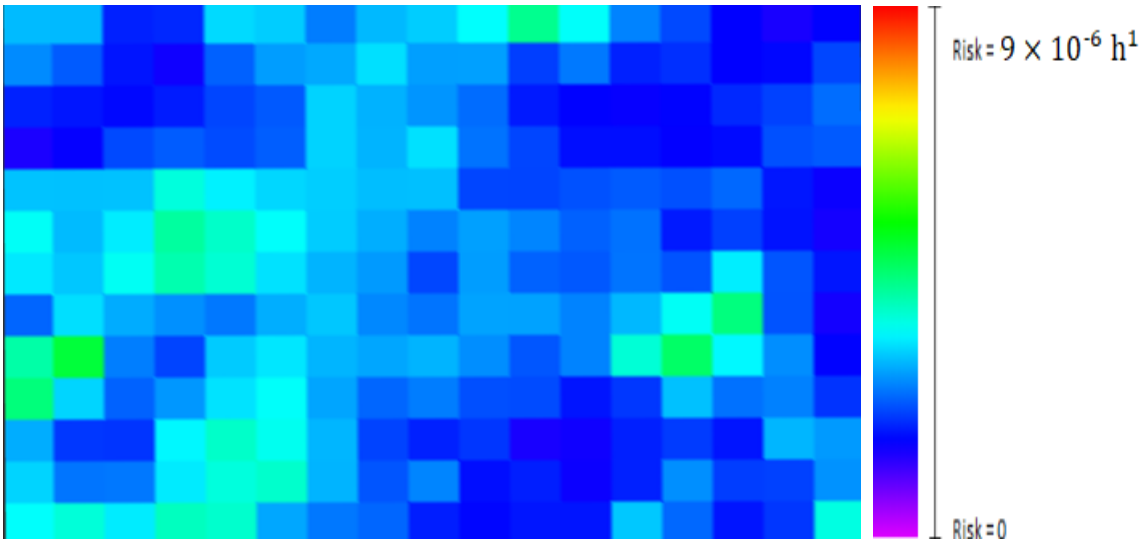

Figure 80: Risk map obtained using deep neural network for "Torino 2" scenario.

5.6 TORINO 3

This dataset is composed by 228 aerial images, the scenario presented in these images is similar the one described above, the difference lies in the fact that there are no longer large buildings (condominiums, palaces), such as those that are usually located in the city, their place was taken from small houses (villas) that are in large numbers. The scenario in this dataset is increasingly dealing with countryside. This is the critical dataset for the performances of both the method.

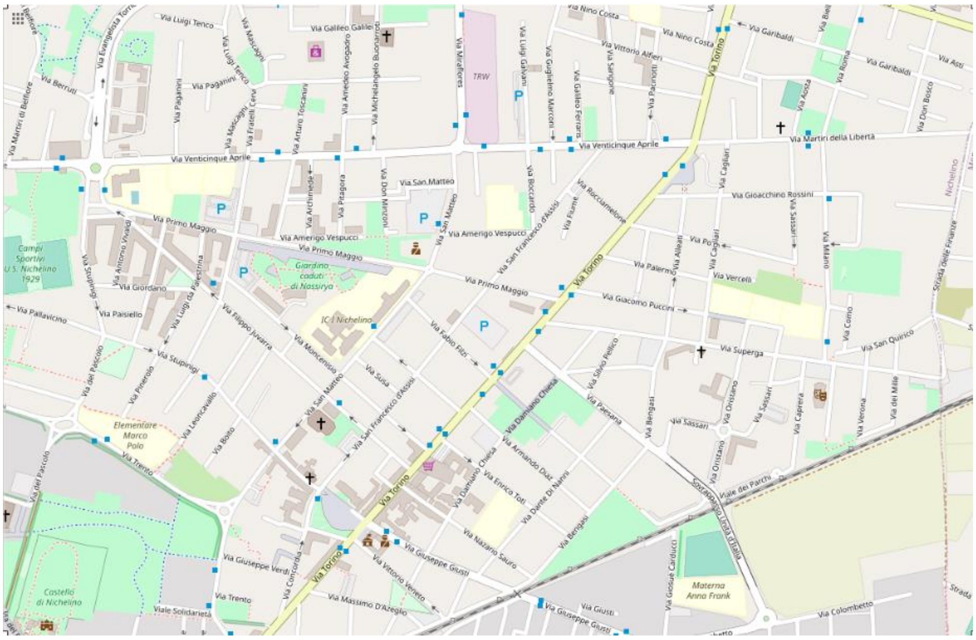

Figure 81: Map representing the "Torino 3" scenario from Open Street Map.

As we can see from scatter plot of *Figure 82*, the two methods seem not to be related in any way, demonstrating that the correlation factor takes on a very small value almost close to zero; correlation factor = 0.0755.

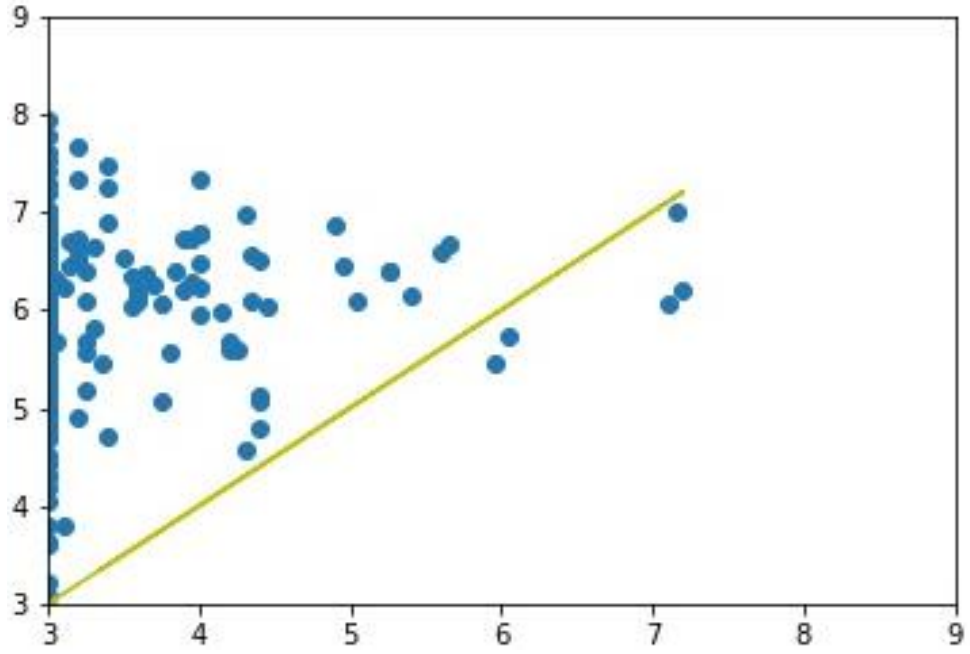

Figure 82: Scatter plot of results obtained in the "Torino3" scenario.

The shape of the point cloud is very different from that seen in the case of the Torino 2 dataset, here the range of values obtained is much smaller.

As can be noted, for each of the images contained in the dataset, the methods produce values that are discordant to each other, this difference is even more noticeable looking at the differential plot of the two methods.

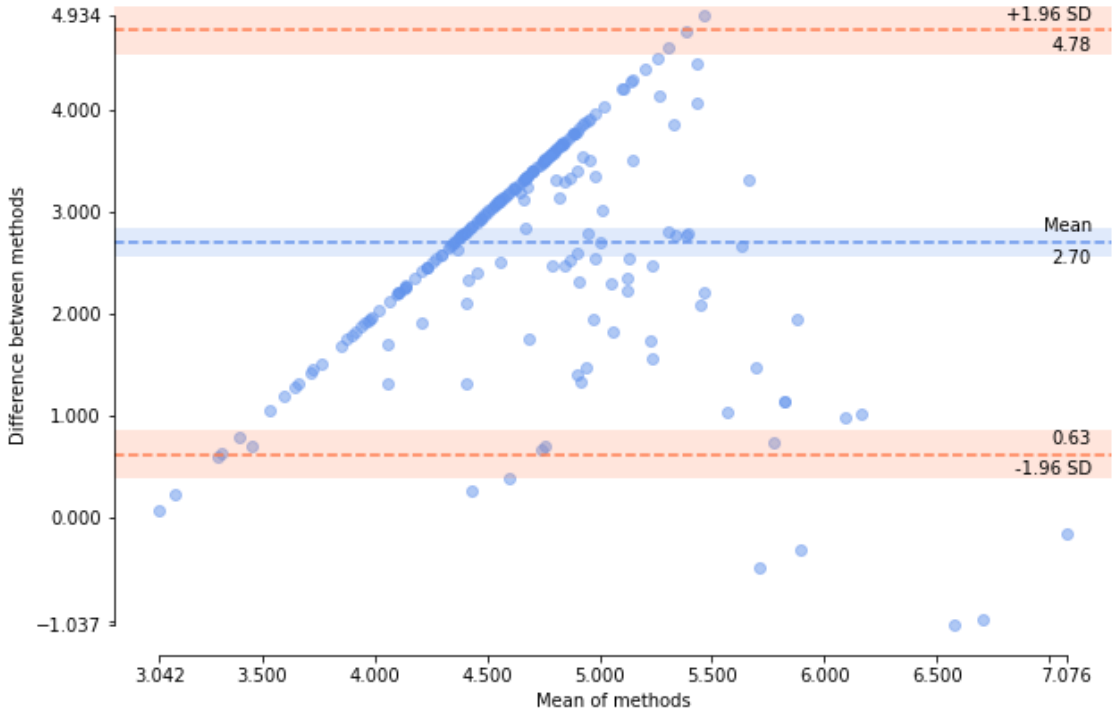

Figure 83: Differential plot of the results obtained in "Torino3" scenario.

From the value of the *mean* it can be seen that the difference between the values obtained is considerable and not negligible, even if more than 95% points are within the confidence range, the two methods cannot be considered interchangeable, as the values obtained would be very different from each other and could give an incorrect assessment of the average sheltering factor of an area. As can be observed from *Figure* 83, the differences between the two methods are not constant, but depend on their value. In this situation it is convenient to experimentally check if it is possible to obtain better results with one method rather than the other. As in the case of the Torino 2 dataset, OSM provides incorrect sheltering factor values, so the AI-based strategy proposed better estimates the sheltering factor in this area. In the case of Turin 3, the difference between the results obtained by the two methods is even greater. The performance of the method proposed in [3] undergoes a further degradation compared to in the case of Torino 2, this is because the three-dimensional model obtained by OSM is incomplete and unable to reproduce the real scenario. The sheltering layer obtained using OSM does not describe the real data of Torino 3 scenario. The average sheltering factor values are very low as indicated also by their colour that is very close to white (*Figure 84)*, this will negatively affect the risk assessment.

Figure 84: Sheltering layer obtained using OSM for "Torino 3" scenario.

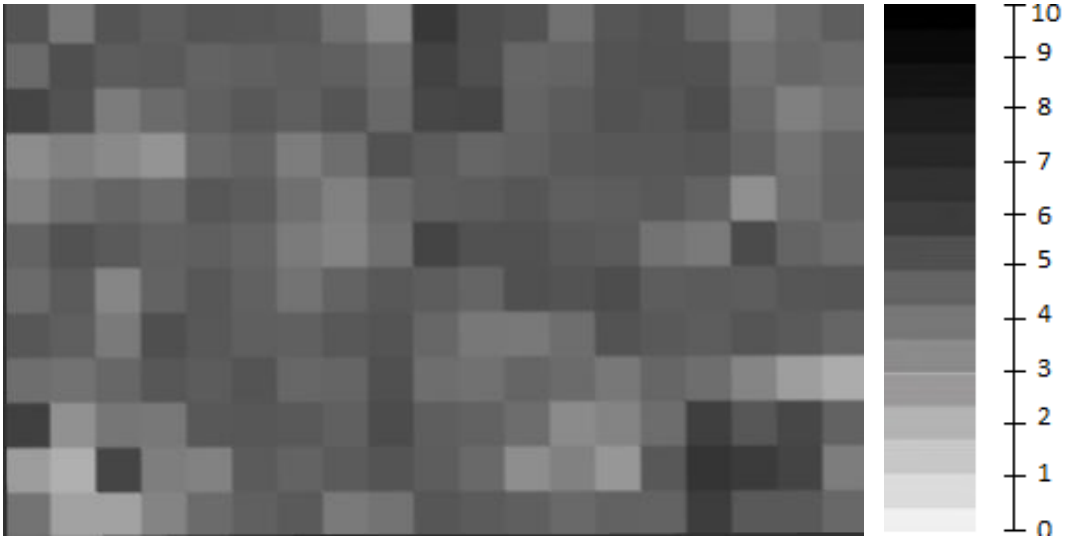

Figure 85: Sheltering layer obtained using deep neural network for "Torino 3" scenario.

The use of the sheltering layer in *Figure 84*, makes it possible to obtain a risk map with high-risk values *Figure 86*, this inevitably affects UASs operations in this area. This goes against the main objective of [2], which is not to limit UASs operations by ensuring security for people in that area. While using the sheltering layer obtained using the proposed strategy, a heterogeneous risk map (*Figure 87*) is obtained whose values reflect the actual risk present in the area, except for some area that presents "abnormal" values.

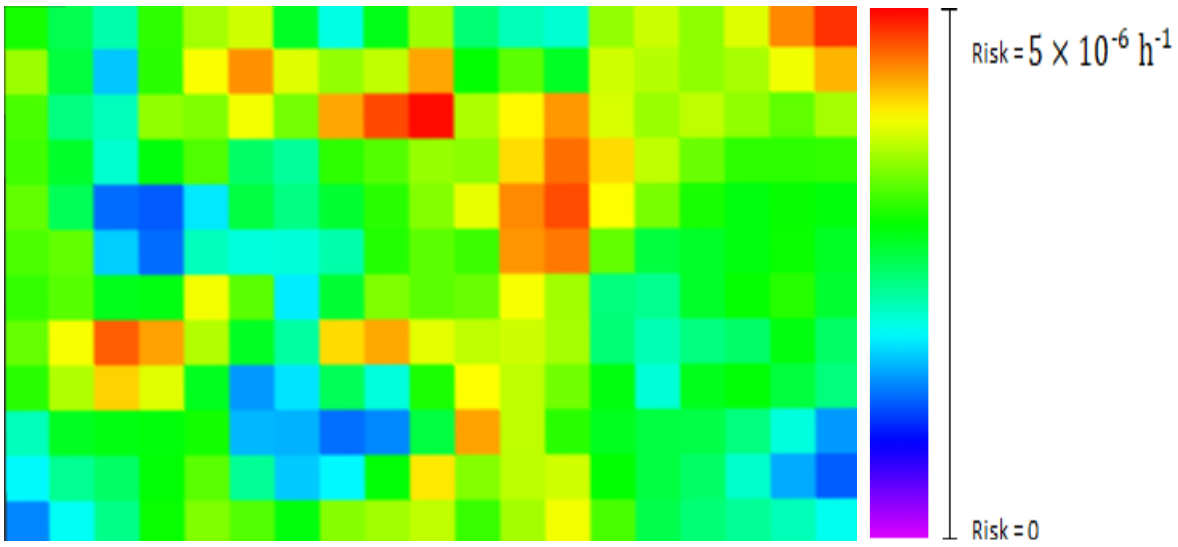

Figure 86: Risk map obtained using OSM for "Torino 3" scenario.

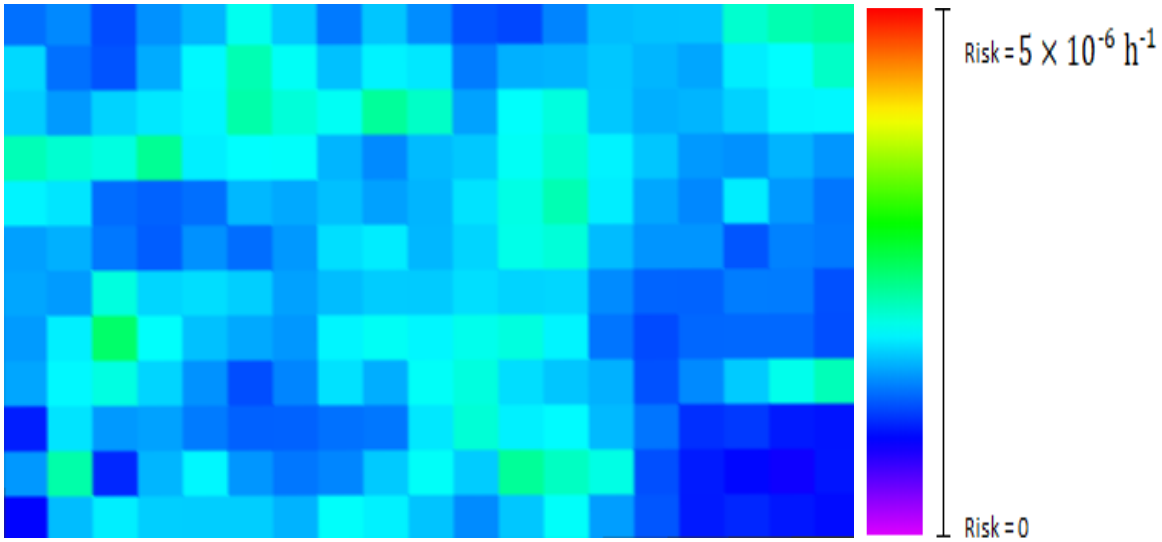

Figure 87: Risk map obtained using deep neural network for "Torino 3" scenario.

What is observed from the experimental data obtained using the deep neural network, is an increase in "abnormal" values that are an indication of the poor ability of the network, to perform buildings segmentation on the images contained in the dataset. In the following images are shown the predictions of the areas which these "abnormal" values are assigned.

Figure 88: Prediction of area with wrong Sheltering Factor 7.2018 in "Torino 3" scenario.

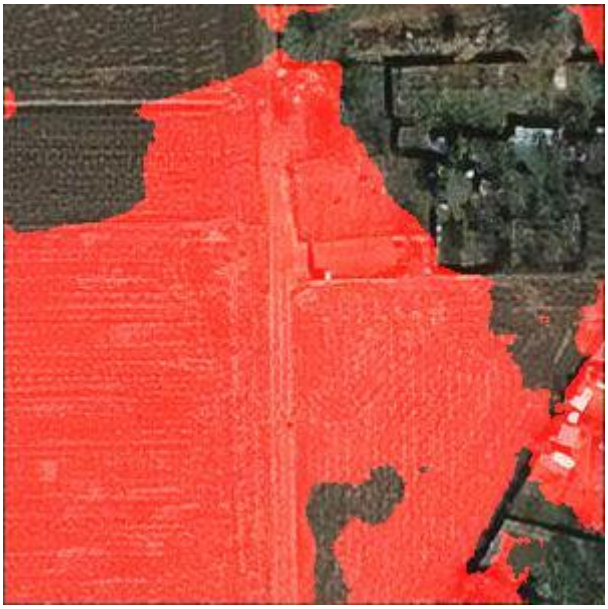

Figure 89: Prediction of area with wrong Sheltering Factor 6.1376 in "Torino 3" scenario.

In conclusion, it can be said that the model used in this work, with all its limitations, represents the best solution between the two methods compared for the dataset Torino3. In the following images will be shown some examples of accurate predictions obtained using the deep neural network.

Figure 90: Prediction of area with average Sheltering Factor 5.9363 in "Torino 3" scenario

Figure 91: Prediction of area with average Sheltering Factor 6.5981 in "Torino 3" scenario.

Figure 92: Prediction of area with average Sheltering Factor 3.789 in "Torino 3" scenario.

CONCLUSION AND FUTURE DEVELOPMENTS

The method proposed in this thesis work is a valid and reliable tool for computing the average Sheltering Factor of a specific area. The use of this technology allows to handle with problems such as: limited availability of public data, their inaccuracy and incompleteness and the variability of granularity and resolution pose.

The AI-based strategy proposed meets all the criteria set out in the literature "From shortest to safest path navigation: an AI-powered framework for risk-aware autonomous navigation of UASs", therefore this thesis represents the starting point for future studies concerning the determination of the sheltering factor layer using the ML technology. Image segmentation will enrich the information contained in the training set identifying objects such as buildings, trees, and cars, to quantify their role in the construction of the sheltering factor. Future works will include the creation of an ad-hoc and complete dataset, which includes images of areas with different characteristics that concern both city and country landscapes.

Even the Deep neural network architecture must be customized, to extract additional objects present in the images, to provide the level of shelter and to find the trade-off between complexity of the network and its performances. In addition, further training and prediction have to be supported by performing hardware that make the work smoother.

To conclude, the most important guideline for any future development is the creation of a sheltering factor reference set of values for each area, so the accuracy of the prediction provided by any other algorithm can be measured. Only in this way a complete and precise analysis of the model is possible and it is also possible to compare different strategies, in order to determine which of these provides the best predictions.

Bibliography

[1] "Torino city lab," *Città di Torino*. https://www.torinocitylab.it/en/.

[2] A. S. Agency, "From shortest to safest path navigation : an AI-powered framework for risk-aware autonomous navigation of UASs Proposed research," pp. 1–5.

[3] S. Primatesta, A. Rizzo, and A. la Cour-Harbo, "Ground Risk Map for Unmanned Aircraft in Urban Environments," *J. Intell. Robot. Syst. Theory Appl.*, vol. 97, no. 3–4, pp. 489–509, 2020, doi: 10.1007/s10846-019-01015-z.

[4] "OpenStreetMap." https://www.openstreetmap.org/#map=5/42.088/12.564.

[5] D. O. HEBB, *The Organization of Behavior: A Neuropsychological Theory*. 1949.

[6] Marvin Minsky and Seymour Papert, *Perceptrons: A introduction to Computational Geometry*. .

[7] Youshua Bengio, *Deep Learning*. .

[8] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 14413–14423, 2020, doi: 10.1109/TVT.2020.3034800.

[9] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3642–3649, 2012, doi: 10.1109/CVPR.2012.6248110.

[10] Y. Zhang, J. Gao, and H. Zhou, "Breeds Classification with Deep Convolutional Neural Network," *ACM Int. Conf. Proceeding Ser.*, pp. 145–151, 2020, doi: 10.1145/3383972.3383975.

[11] Erik Solem, *Programming Computer Vision with python*. .

[12] "No Title." https://deepai.org/machine-learning-glossary-and-terms/computer-vision.

[13] L. Saltalamacchia, "An Artificial Intelligent methodology to estimate the population density of urban areas to compute risk maps for Unmanned Aircraft Systems."

[14] E. Shelhamer, J. Long, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017, doi: 10.1109/TPAMI.2016.2572683.

[15] E. Shelhamer, J. Long, and T. Darrell, “Fully Convolutional Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017.

[16] and T. B. Olaf Ronneberger, Philipp Fischer and Computer, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *IEEE Access*, vol. 9, pp. 16591–16603, 2021, doi: 10.1109/ACCESS.2021.3053408.

[17] V. Iglovikov, S. Seferbekov, A. Buslaev, and A. Shvets, “TernausNetV2: Fully convolutional network for instance segmentation,” *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018-June, pp. 228–232, 2018, doi: 10.1109/CVPRW.2018.00042.

[18] “FCN, UNet, FPN comparison.” https://chadrick-kwag.net/fcn-unet-fpn-comparison/#:~:text=Another slight difference between UNet,conv layers for additional processing.

[19] “The Bing Maps REST.” https://docs.microsoft.com/en-us/bingmaps/rest-services/#:~:text=The Bing™ Maps REST,metadata%2C or creating a route.&text=If you plan to use the REST services from .

[20] “What is REST API.” https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[21] upGrad blog, “Top 10 Reasons Why Python is So Popular with Developers in 2021.” .

[22] C. D. Costa, “Best Python Libraries for Machine Learning and Deep Learning.” .

[23] “PyTorch.” https://pytorch.org/.

[24] A. T. V. F. B. S. C. ELI STEVENS, LUCA ANTIGA, *Deep Learning With PyTorch*. 2020.

[25] “‘PyTorch-About.’” pytorch.org.

[26] “Data Augmentation in Python: Everything You Need to Know.” https://neptune.ai/blog/data-augmentation-in-python.

[27] “Scikit-image.” https://scikit-image.org/.

[28] S. Van Der Walt *et al.*, “Scikit-image: Image processing in python,” *PeerJ*, vol. 2014, no. 1, 2014, doi: 10.7717/peerj.453.

[29] “Requests: HTTP for Humans™.” https://requests.readthedocs.io/en/master/.

[30] “Package pyproj.” https://pyproj4.github.io/pyproj/v1.9.6rel/pyproj-module.html.

[31] “Detailed explanation of MsCOCO data set.” https://www.programmersought.com/article/1211670395/.

[32] “The Vanishing gradient problem.” https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[33] “Vanishing gradient problem Tensorflow.” https://adventuresinmachinelearning.com/vanishing-gradient-problem-tensorflow/.

[34] “Correlation coefficient.” https://en.wikipedia.org/wiki/Correlation_coefficient.