POLITECNICO DI TORINO

Dipartimento di Ingegneria Meccanica e Aerospaziale

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Tesi di Laurea Magistrale

Sheltering factor recognition via machine learning for fully autonomous UAV urban path planning

Relatori:
Prof. Giorgio Guglieri
Dott. Luca Spanò Cuomo

Candidato:
Andrea Gozzo 219337

20 Luglio 2018

Contents

List of Figures

List of Tables

Abstract

The context of the work done in this thesis involves the smart cities. A smart city is an urban area that collects data in order to efficiently manage infrastructures and resources. The proposed architecture involves the use of three different systems: the autonomous vehicles, the non-autonomous vehicles and the cloud system. The autonomous vehicles have the purpose of acquiring information, the non-autonomous vehicles have the purpose of acquiring information and support the autonomous vehicles, the cloud system processes the monitoring data and execute the path algorithm. The path planner algorithm calculate the path optimizing a risk function, evaluated considering the cells of a risk map. In order to generate the risk map, four different layers are required: occupation layer, population density layer, sheltering factor layer and signal power layer. The aim of this thesis is to improve the performance of the path planning algorithms considering two aspects: the autonomous vehicle crash distance after a failure and automating the generation of the sheltering factor layer. The crash distance was considered differentiating fixed wing UAVs from rotary wing UAVs. For fixed wing, starting from the equilibrium equations of a body, a simple equation has been found. The equation is able to evaluate the crash distance knowing the aerodynamic efficiency and initial altitude of the UAV. The effects of the implementation of the crash distance on the risk map generation are analyzed, in particular it was found that the risk shows a more uniform distribution with larger values of crash distance. Vice versa, for a smaller crash distance, the risk distribution in the risk map is less uniform, with areas characterized by a much higher risk value than others. For the rotary wing, the ballistic equations have been numerically integrated. The obtained trajectories have been compared, considering different values of mass, drag coefficient, altitude, initial velocity and reference surface. The results are presented in two forms, one specific and one more generic. In the specific case, knowing a priori all the characteristics of the UAV considered (mass, drag coefficient, reference surface and flying altitude), it is possible to obtain an equation that evaluate the crash distance as a function of the initial speed. In the second generic case, a table has been generated. This table contains numerous combinations of mass, drag coefficient, reference surface, flight altitude and speed associating each of them the corresponding crash distance. The automation of the generation of the sheltering factor layer was performed by training neural networks in order to solve the problem of image classification. A database was created considering open source satellite images. Two different networks have been trained. The first was made from scratch, in order to be very light and fast. This network considers only one channel of the input images and is composed of a small number of layers. The second network is an already existing neural network, in this case transfer learning has been performed. This network is more complex and the training much slower. In the first case, the training was not successful enough to be used for generating the sheltering factor layer. In fact, the trained network accuracy is relatively small and presents overfitting. Necessary improvements in order to increase this network performance are described, including the implementation of a variable learning rate and the consideration of all three image channels instead of only one. In the second case, the network gave satisfactory results. Accuracy is relatively high with no overfitting observed. The improvements that could increase the performance of this network mainly concern the database. In particular, using a better database with higher image resolution and quality is suggested as future development.

Chapter 1

Overview of the main project

A smart city is an urban area that collects data from a multitude of sensors to provide information used to efficiently manage infrastructures and resources. The main purpose of a smart city is to improve the quality of life, improve the environmental conditions, reduce energy consumption, ensure the safety of the community. The data is collected, processed and analyzed in order to monitor and control traffic and transportation systems, schools, libraries, hospitals etc. Given the huge quantity of data and information to analyze, a solid monitoring system is required. The infrastructure that an efficient monitoring system needs is made of many sensors to acquire data, a networking equipment to transmit it and a number of computing devices for data computing and management. On a such large scale it is unthinkable to use traditional measurement techniques, therefore remote sensing system is proposed. The advantages of remote sensing technology are many: it allows retrieval of data in places difficult to access, it is possible to cover very large areas, it is convenient for repetitive measuring since it allows collection of more data in a short period of time, it can retrieve large amounts of data and so on.

1.1 The multi-modal architecture

The architecture described in this thesis employs a multi-modal strategy that includes unmanned and manned configurations. The unmanned configuration is composed of Unmanned Aerial Vehicles (UAVs), Unmanned Ground Vehicles (UGVs) and Unmanned Surface Vessels (USVs). To coordinate between manned and unmanned configurations is developed a Cloud-based framework used to collect data and manage all vehicles involved in monitoring applications. The architecture is composed by three main blocks: the autonomous vehicles, the non-autonomous vehicles and the Cloud.

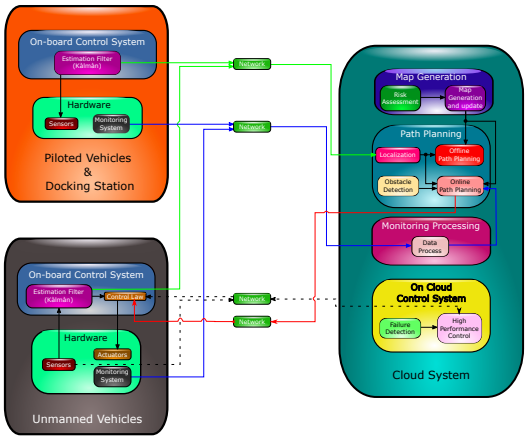


Figure 1.1: The System Architecture

The autonomous vehicles tasks are to acquire the information and guarantee the stability of the vehicles. The non-autonomous vehicles tasks are to acquire information and give support to the autonomous vehicles. The cloud has the purpose of process the monitoring data and execute the path planning algorithms. The cloud system is divided in four blocks:

- The map generation block provides the risk maps to the path planning block.
- The path planning evaluates the trajectories of the autonomous vehicles optimizing the navigation risk.
- The monitoring processing receives data from manned and unmanned vehicles in order to respect the mission requirements.
- The cloud control system optimizes tasks requirements in function of the constrains.

1.2 Risk Evaluation

The risk map is used by the path planner to evaluate the path that minimize the risk function. The map and each layer are divided into smaller areas called cell. Using the information provided by the layers and information about the UAV the risk associated to each cell is calculated. The equation used for evaluating the risk is:

$$R_{C_{cell}}(x, y, \tau) = A_c D_P P(fatality|exposure) P(E_{IntFail})$$

where:

- R_C is the frequency of fatality in the cell considered expressed in $fatalities/h$;
- x, y are the coordinates of the cell considered;
- τ is the time during which the UAV flies over a certain cell.
- A_c is the critical area related to one person expressed in m^2;
- D_P is the population density associated to the cell considered expressed in $people/m^2$;
- $P(fatality|exposure)$ is the probability a person will suffer fatal injuries given exposure to the accident;
- $P(E_{IntFail})$ is the probability that a system internal failure occurs, is obtained considering the product of failure rate and time passed on a cell.

It is important to notice that the probability of mid-air collision is neglected.
The factor $P(fatality|exposure)$ is evaluated as:

$$P(fatality|exposure) = \frac{1-k}{1-2k+\sqrt{\frac{\alpha}{\beta}}[\frac{\beta}{E_{imp}}]^{\frac{3}{p_s}}}$$

Where:

- $k = min[1, (\frac{\beta}{E_{imp}})^{\frac{3}{p_s}}]$ is a correction factor;
- E_{imp} is the kinetic energy at impact, it depends on the characteristics of the UAV (mass, type etc.), and on the variables of the mission (operational altitude and speed);
- p_s is the sheltering factor;

- α is the impact energy required for a fatality probability of 50% when $p_s = 6$, its value is 100 kJ;

- β is the impact energy threshold required to cause a fatality when $p_s = 0$, its value is 34 J.

Summarizing, the risk equation considers critical area, population density and probability of fatal injures to evaluate the frequency of fatality in a cell. The sheltering factor p_s is one principal factor considered. The presence of objects or buildings can shelter a person from the impact reducing the probability of fatal injuries. The sheltering factor is a real number evaluated with a qualitative estimation of the surface in the cell. Some examples of sheltering factors are shown in the table below.

Sheltering Factor value	Type of area
0	No obstacles
2.5	Sparse trees
5	Trees and low buildings
7.5	Tall buildings
10	Industrial area

Table 1.1: Some sheltering factor values

1.3 Path Planner

The path planner computes an optimal path from a starting point to a target point minimizing a cost function, considering the risk map generated by the Map Generation block. The path planner considers the riskA* algorithm. The riskA* algorithm is based on the A* algorithm. Given the risk map, the purpose of the path planner is to minimize the cost function:

$$f(x) = g(x) + k \cdot h(x)$$

Where $g(x)$ is the historical motion cost of the path from the node x to the starting node x_{start}, k is an adjustment variable and $h(x)$ is the heuristic cost. Then the $f(x)$ will express the estimated motion cost of the shortest path starting from x_{start}, reaching x_{goal} and passing through the node x considered. Differently from the original A* algorithm, the RA* algorithm implement the the risk cost in order to evaluate the estimated motion cost $f(x)$. Considering a generic point x_i, the historical motion cost $g(x)$ can be expressed as:

$$g(x_i) = \int_{x_{start}}^{x_i} risk(x)dx$$

where $risk(x)$ is the risk cost function of the path. The heuristic cost function $h(x_i)$ can be expressed as:

$$\int_{x_i}^{x_{goal}} risk(x)dx$$

Then the value of the integral $h(x_i)$ is the risk cost between the point x_i and the final point x_{goal}. The constant k is the adjustment variable of the heuristic cost function and it has to be found trough validation. Common values of k variate between 0.5 and 2. A graphical representation of $f(x)$ is shown in the following picture.

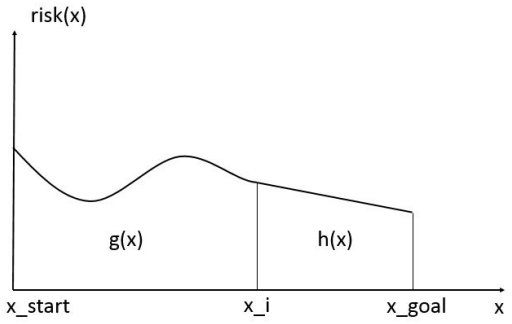


Figure 1.2: The cost function $f(x)$. It is expressed as the sum of the historical motion $g(x)$ and the heuristic cost.

1.4 Map Generation

The map generation block is the first block of the Cloud System. It provides a dynamic risk map where at each area is associated a risk value. With the map is possible to evaluate the risk of flying over a specified area. A risk map is defined as a bidimensional matrix, where each element identifies a portion of the total area. The risk map is generated by the sum of four different layers that describe different characteristic of the area contained, each layer contributes in a different way to the generation of the map.

1.4.1 First Layer: Occupation/Prohibition Layer

This layer contains the information about the areas where is possible to fly. It provides both the position of buildings or obstacles at the altitude considered and the constrains concerning the areas where the flight is prohibited by law. This layer can be considered constant over time.

Figure 1.3: No-fly zone layer

The map shows an example of no-fly zone layer. The occupied cells are represented in red colour, while the cells where is possible to fly are represented in purple.

1.4.2 Second Layer: Population Density

This layer considers the amount of human population that is likely to be found in the corresponding cell. This layer can evolve in real time, as it is generated considering the data related to population monitoring. A high population density lead to an increase in the value of the risk. The ability to update this map in real time is typical of a smart city, because depends on the capacity to detect or estimate the distribution of the population, in order to improve the efficiency of services, or in this case to reduce the risk for its inhabitants.

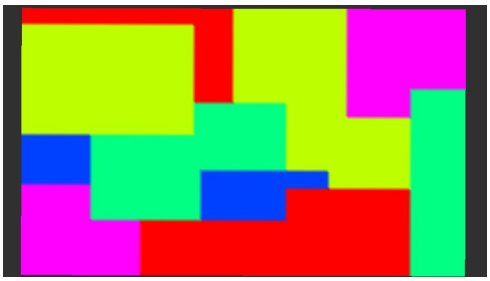


Figure 1.4: Population density layer

The map shows the population density layer. Given the impossibility to have an authentic real time population density layer during the activity carried out in this thesis, this layer is generated randomly only for testing purposes. The population density increases according to this colours: teal, blue, green, purple, red.

1.4.3 Third Layer: 4G/5G Signal Power

This layer contains the information about the quality of the signal 4G/5G. This layer is independent over time but could depend on the altitude considered.

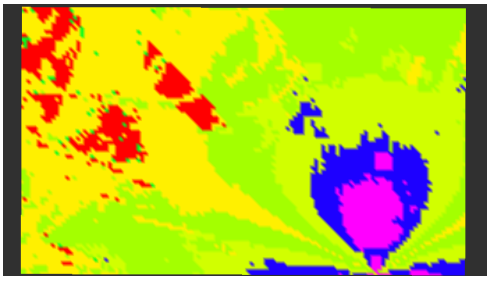


Figure 1.5: Signal power/coverage layer

This layer represents an example of 4G signal power map.

1.4.4 Fourth Layer: Sheltering Factor

The Sheltering Factor layer represents the environmental factors of the map. It describes the type of surface and identifies the presence of objects or obstacles found on the ground in case of crash, for example trees, building, streets, cars etc. Knowing the presence of obstacles is of fundamental importance as they can greatly influence the probability of fatal injuries.

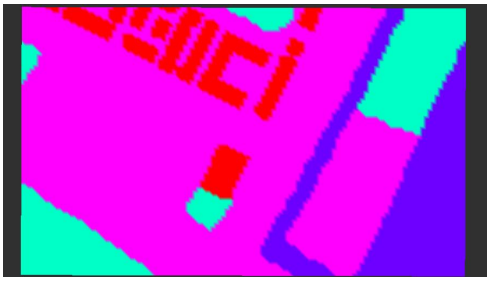


Figure 1.6: Sheltering Factor Layer

This map shows the Sheltering Factor layer of the map used to evaluate the risk maps used as example in this thesis. The sheltering factor varies depending on the ground surface. The purple colour corresponds to a low sheltering factor value (larger risk), the blue colour corresponds to a mid value of sheltering factor (average risk), the teal colour corresponds to a high value of sheltering factor (smaller risk).

An example of final risk map obtained considering the described layers is shown in the following image.

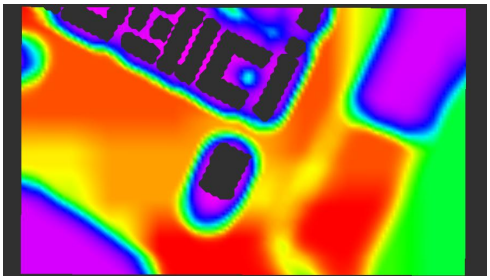


Figure 1.7: Final Risk Map generated by the Map Generation segment considering the described layers.

As expected, the sheltering factor value has a very important role in risk evaluation. Indeed,it is common to find higher risk value in cells where the sheltering factor value is smaller, and lower risk value in cells where the sheltering factor value is larger.

Chapter 2

UAV offensiveness

The flight of UAVs on Italian territory is regulated by the ENAC and EASA. According to regulations, drones are divided into three categories: Open Category, Specific Operation Category and Certified Category.

The UAVs belonging to the Open Category present a low risk related to the flight. They can be piloted by everyone as they do not require the possession of a license and does not require a prior authorization by the competent authority. They can be piloted in visual line of sight operations (VLOS), or in first-person view only if remote pilot is assisted by a visual observer located in his proximity. It is possible to fly over populated areas only if areas with crowds or concentration of people are avoided. The low risk associated with this category depends by many factors. One of the most important factors is the low kinetic energy transmitted in case of impact. Other factors can be the compliance with design features that make it harmless in event of accident, the experience of the remote pilot and so on. An example of UAVs belonging in this category are small UAVs used for photographic inspections or for leisure activities, characterised by a mass smaller than 2 kg.

The UAVs belonging to the Specific Operation Category require an authorization from the competent authority to operate. In this category are usually included UAVs with a mass greater than 2 kg. Considering the higher risk presented, the UAV operator must carry out a risk assessment process, where the risk is identified, analysed and evaluated, and the resulting mitigation measures are taken to obtain an authorisation to fly. An example of this category are flight operations where the remote pilot cannot see the UAV (Beyond Line Of Sight Operations or BVLOS), or flight operations above heavily populated areas.

The UAVs belonging to the Certified Category present a high risk. The vehicle is considered as a civil aviation aircraft and certification is required according to EASA standards. A licensed remote pilot is required and the operator must be approved by the competent authority.

2.1 Quantitative criterion for UAV offensiveness

The quantitative criterion used to evaluate the offensiveness of a UAV is based on its kinetic energy at the impact. Experimental studies have shown that an impact energy of 56 J is can cause fatal injuries. In the following table are presented some values of speed at which the energy of 56J is reached:

Mass [kg]	Speed [m/s]
0.5	15
1	10
2	7.5
4	5

Table 2.1: Some combinations of speed and mass corresponding to 56 J

Where kinetic energy is calculated with:

$$Energy = \frac{1}{2} \cdot Mass \cdot Speed^2$$

The impact energy of a fixed wing UAV is calculated considering the maximum speed reachable by the UAV.

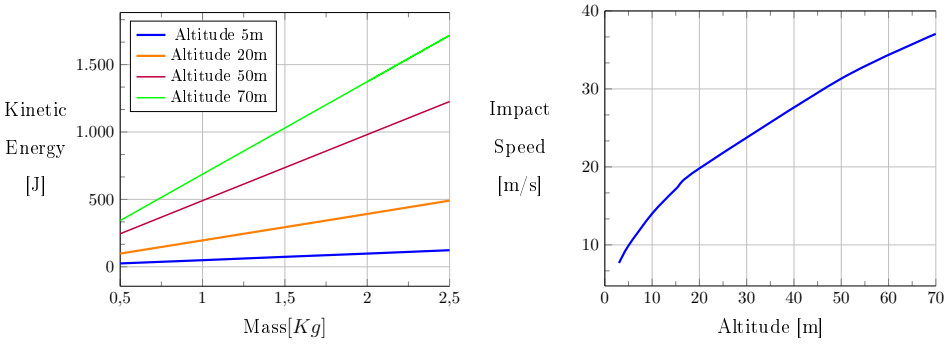
The impact energy of a rotary wing UAV can be calculated following two approaches. The first approach considers the gravitational potential energy, obtained with:

$$Energy = Mass \cdot g \cdot Altitude$$

Consequently, the impact speed is calculated with:

$$Speed = \sqrt{2 \cdot g \cdot Altitude}$$

. This conservative approach leads to an overestimation of the impact speed, especially for flight at higher altitude, because it does not consider the aerodynamic drag. The following pictures show the kinetic impact energy and speed in function of altitude and mass:



(a) (b)

Figure 2.1: The image 2.1a shows the kinetic energy at impact versus mass parameterized in different altitudes.

It is important to notice that for lower altitudes, the impact energy is relatively small, increasing slightly with the mass. Instead, for higher values, the impact energy reaches extremely large values and increases a lot with mass. This happens because, following this approach, the atmospheric aerodynamic drag is neglected. While for smaller altitudes this approach can still be considered valid, for higher altitudes it has to be rejected, because of the unrealistic valued found.

The second approach considers the aerodynamic drag calculated as:

$$D = \frac{1}{2} \cdot \rho \cdot C_D \cdot S \cdot V^2$$

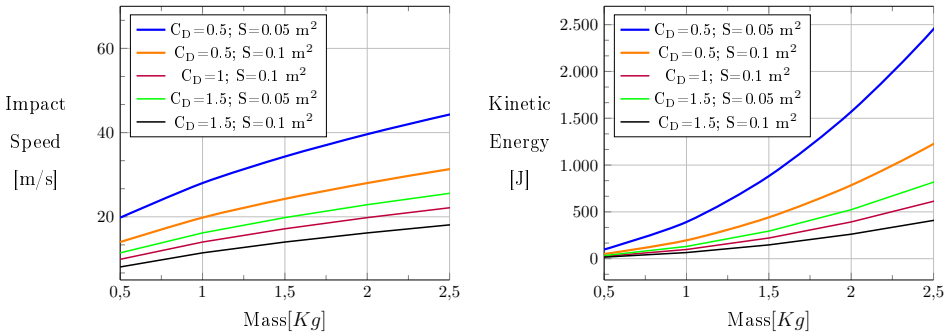
Knowing the values of drag coefficient C_D, reference surface S and mass of the UAV, equalling the aerodynamic drag to the weight:

$$D = \frac{1}{2} \cdot \rho \cdot C_F \cdot S \cdot V^2 = Mass \cdot g$$

the impact speed is obtained with:

$$V = \sqrt{\frac{2 \cdot Mass \cdot g}{\rho \cdot C_D \cdot S}}$$

It is possible to plot the kinetic energy and impact speed in function of mass, depending on reference surface and drag coefficient.



(a) (b)

Figure 2.2: Impact spees and impact energy in functiom of mass and couple of values of CD and reference surface.

The images show the impact speed in function of mass, given a couple of drag coefficient CD and reference area values. The speed values range from the lowest value of 8 m/s to the highest value of 48 m/s. As expected, a higher value of mass leads to a larger value of impact speed and energy, while larger values of drag coefficient, lead to smaller impact speed and energy. The impact energy ranges from a minimum of 16 J to a maximum of 3530 J. This demonstrate the importance of the design characteristics of the UAV. It is possible to notice how easily the impact energy value of 56 J (the value of energy officially considered dangerous) is exceeded, also for a very small value of mass. The most important influence is given by the value of the drag coefficient CD. This demonstrate the importance of equip the UAV with an emergency system capable of increasing the drag coefficient in case of failure.

2.2 Qualitative criteria for UAV offensiveness

There are eight qualitative criteria used to evaluate offensiveness, they involve the UAV design parameters such as geometric and constructive properties, material used, components etc.

- Resistance of envelopes to tearing.

  This applies to UAVs where the lift force is generated by an envelope such as airships or air balloons. The envelope must be able to resist tearing and prevent loss of gas.

- Material used for the main parts.

  Fuselage, wings and other main components of the UAV must be built with low density material with an high deformability capacity.

- Covering materials.

  The covering of fuselage, wings and other main components must be able to transmit the deformation caused by the impact to the underlying material without resisting.

- Sharp components.

  The presence of sharp components on the UAV is not acceptable. This criterion applies mainly to antennas, propellers and rotors.

- Protuberances and components made of hard material.

  The presence of exposed components or protuberances made of hard or high-density material must be avoided. This criterion also applies to non-sharp and non-cutting parts, such as propeller and rotor protections, cameras, batteries and so on.

- Rotating components.

  It is necessary to prevent the contact of propellers and rotors against people in event of a collision.

- Fire protection.

  The design of the UAV must consider measures to minimize the risk of fire, for example in case of battery damage or short circuits of the electrical system after a collision.

- Connections of the components.

  All the components of the UAV must be assembled in order to prevent their detachment during flight operations.

Chapter 3

Crash distance analysis

The study of the crash distance of the UAV after a failure that totally compromises its operativity is of fundamental importance during the risk evaluation. If the crash distance is considered, the risk function will not depend only on areas overflown by the UAV during nominal conditions but also adjacent areas where there is the possibility to crash. The approach will be different in case of fixed wing or rotary wing UAVs.

3.1 Fixed wing

In this section is described the procedure followed to obtain the equation used to evaluate the crash distance for a fixed wing UAV.

It is assumed that the UAV, after a failure that totally prevents its operativity, glides at constant speed and in stationary and stable conditions until the impact to the ground. It is also assumed the absence of external disturbance forces. The transient to reach the gliding condition from the starting cruise conditions after the failure is neglected.

The hypotheses are:

- Not propelled flight
- Absence of external disturbance forces
- Instantaneous failure of all components
- Gliding in stationary and stable conditions
- Transient between cruise conditions and gliding conditions neglectable

3.1.1 Analytical formulation

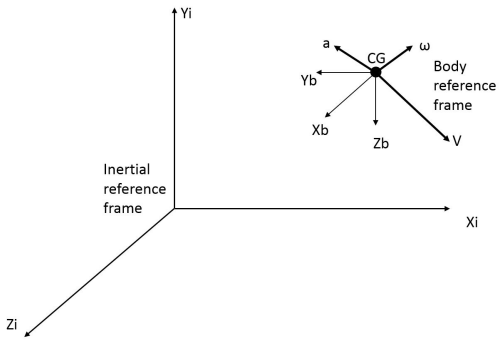


Figure 3.1: Velocities in inertial an body reference frames

It is considered the centre of gravity of an aircraft with velocity $\overrightarrow{V}$, acceleration is $\overrightarrow{a}$ and angular velocity $\overrightarrow{\omega}$. Given an inertial reference frame, the equilibrium is expressed as:

$$\overrightarrow{F}^A + \overrightarrow{F}^C + \overrightarrow{F}^I = 0$$

and:

$$\overrightarrow{M}^A + \overrightarrow{M}^C + \overrightarrow{M}^I = 0$$

Where:

- A indicates the applied external forces and moments
- C indicates the constraint forces and moments
- I indicates the inertial forces and moments

For a flying aircraft the constraint forces and moments are null, then $\overrightarrow{F}^C = O$, $\overrightarrow{M}^C = 0$.

The expressions of velocity, momentum and forces along the body axes reference frame are considered (axis $\overrightarrow{i}$, $\overrightarrow{j}$, $\overrightarrow{k}$):

$$\overrightarrow{V} = u \cdot \overrightarrow{i} + v \cdot \overrightarrow{j} + w \cdot \overrightarrow{k}$$

$$\overrightarrow{\omega} = p \cdot \overrightarrow{i} + q \cdot \overrightarrow{j} + r \cdot \overrightarrow{k}$$

$$\overrightarrow{Q} = m \cdot u \cdot \overrightarrow{i} + m \cdot v \cdot \overrightarrow{j} + m \cdot w \cdot \overrightarrow{k}$$

$$\overrightarrow{K} = K_x \cdot \overrightarrow{i} + K_y \cdot \overrightarrow{j} + K_z \cdot \overrightarrow{k}$$

$$\overrightarrow{F}^A = F_x \cdot \overrightarrow{i} + F_y \cdot \overrightarrow{j} + F_z \cdot \overrightarrow{k}$$

$$\overrightarrow{M}^A = M_x \cdot \overrightarrow{i} + M_y \cdot \overrightarrow{j} + M_z \cdot \overrightarrow{k}$$

Where:

- u, v, w are the components of velocity vector $\overrightarrow{V}$ along the body axes

- p, q, r are the components of angular velocity vector $\overrightarrow{\omega}$ along the body axes (roll, pitch and yaw angular velocities)
- m is the mass of the aircraft, supposed constant
- $\overrightarrow{Q}$ is the momentum vector
- $\overrightarrow{K}$ is the angular momentum vector

The external forces and moments components F_x, F_y, F_z, M_x, M_y, M_z contain the terms depending on aerodynamic forces, gravitational forces and propulsive forces. Considering the hypothesis of stationary flight and neglecting the lateral-directional plane, three of the previous set of equations are equal to zero:

$$\overrightarrow{\omega} = 0, \overrightarrow{K} = 0, \overrightarrow{M} = 0$$

The inertial forces are evaluated as:

$$\overrightarrow{F}^I = -\frac{d\overrightarrow{Q}}{dt}; \overrightarrow{M}^I = -\frac{d\overrightarrow{K}}{dt} - \overrightarrow{V} \times \overrightarrow{Q}$$

If the moments are expressed in axes applied on the centre of gravity results that $\overrightarrow{V} \times \overrightarrow{Q} = 0$. Then:

$$\overrightarrow{F}^A = \frac{d\overrightarrow{Q}}{dt}$$

$$\overrightarrow{M}^A = \frac{d\overrightarrow{K}}{dt}$$

Deriving the vector $\overrightarrow{Q}$ with respect to time (the mass has beeen assumed constant):

$$\frac{d\overrightarrow{Q}}{dt} = [\dot{\overrightarrow{Q}} + \overrightarrow{\omega} \times \overrightarrow{Q}] = [m \cdot (\dot{u} \cdot \overrightarrow{i} + \dot{v} \cdot \overrightarrow{j} + \dot{w} \cdot \overrightarrow{k}) + \overrightarrow{\omega} \times \overrightarrow{Q})]$$

Evaluating the term $\overrightarrow{\omega} \times \overrightarrow{Q}$ is obtained:

$$\overrightarrow{\omega} \times \overrightarrow{Q} = m \cdot [(q \cdot w - r \cdot v) \cdot \overrightarrow{i} + (r \cdot u - p \cdot w) \cdot \overrightarrow{j} + (p \cdot v - q \cdot u) \cdot \overrightarrow{k}]$$

This term is null because of the previous consideration.

Substituting and separating the components along the three axes (remembering that has been assumed $v = 0, \dot{v} = 0$:

$$F_x = m(\dot{u} + q \cdot v - r \cdot v) = m \cdot \dot{u}$$

$$F_y = m(\dot{v} + r \cdot u - p \cdot w) = m \cdot \dot{v}$$

$$F_z = m(\dot{w} + p \cdot v - q \cdot u) = m \cdot \dot{w}$$

These equations are set equal to zero because of the hypothesis of stationary flight.

Considering wind axes is possible to write the forces as:

$$F_x = T \cdot \cos\alpha - D - W \cdot \sin\gamma \cong T - D - W \cdot \sin\gamma$$

$$F_z = T \cdot \sin\alpha + L - W \cdot \cos\gamma \cong L - W \cdot \cos\gamma$$

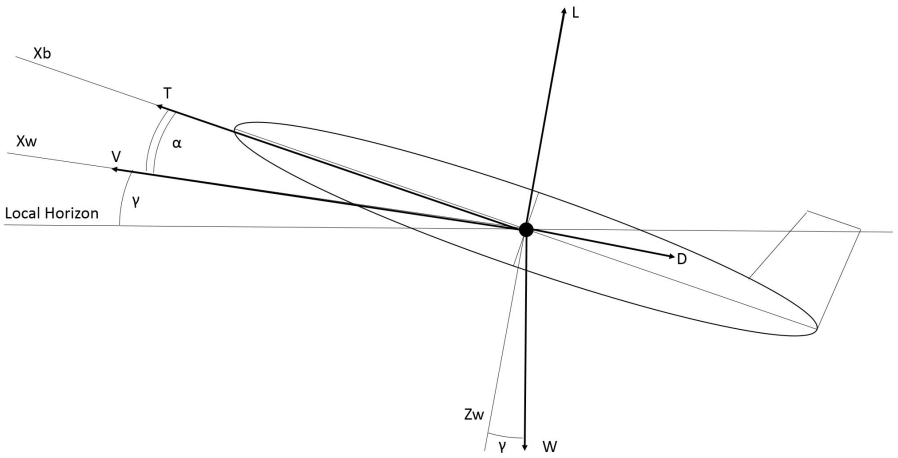


Figure 3.2: Forces in wind axes

Given the assumptions of stationary ($F_x = 0, \;\; F_z = 0$), and non-propelled flight ($T = 0$), these equilibrium equations are obtained:

$$D = -W \cdot \sin\gamma$$

$$L = W \cdot \cos\gamma$$

The aerodynamic efficiency is defined as $E = \frac{L}{D} = \frac{C_L}{C_D}$. It is possible to combine the equilibrium equations to calculate the glide angle:

$$\frac{D}{L} = -\frac{W \cdot \sin\gamma}{W \cdot \cos\gamma} = -tan\gamma$$

It also possible to write it as:

$$tan\gamma = -\frac{1}{E}$$

Considering h the altitude and X the horizontal distance flown, it is possible to write:

$$\frac{dh}{dX} = tan\gamma = -\frac{1}{E}$$

The final expression of the distance is:

$$X = -E \cdot \Delta h = E \cdot h$$

It is useful to remember that Δh is negative being calculates as $\Delta h = (0 - h)$. It is also important to notice that the velocity does not directly affects the value of the distance flown, but it affects it indirectly because the aerodynamic efficiency E also depends on it.

3.1.2 Results: crash distance

Considering as initial conditions an altitude of 35 meters and an efficiency of 7, the crash distance obtained it 245 meters. The trajectory is linear as shown in the following graph.

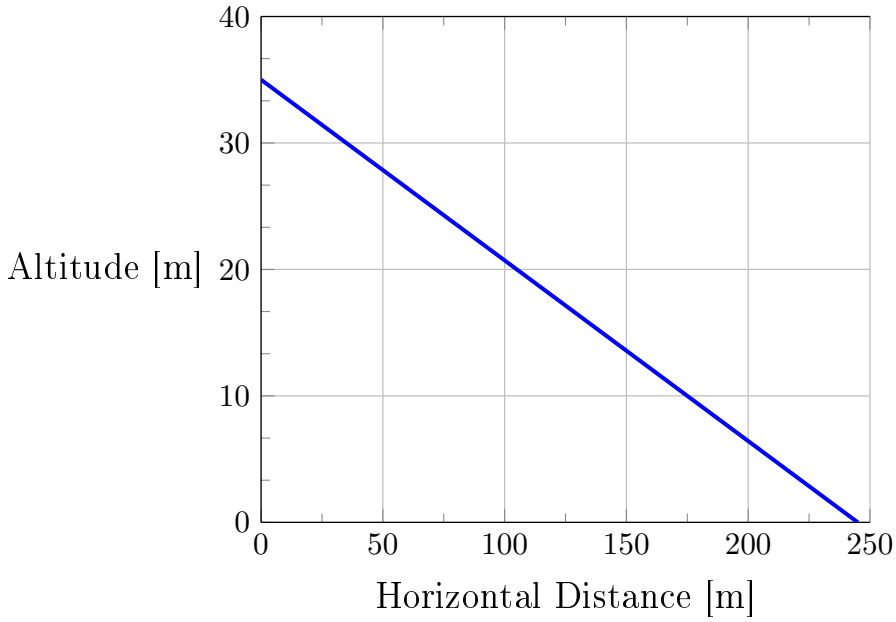


Figure 3.3: Trajectory, fixed wing UAV

To better understand the hypothesis of neglecting the transient it is sufficient to look at the complete trajectory of the UAV before and after the failure.

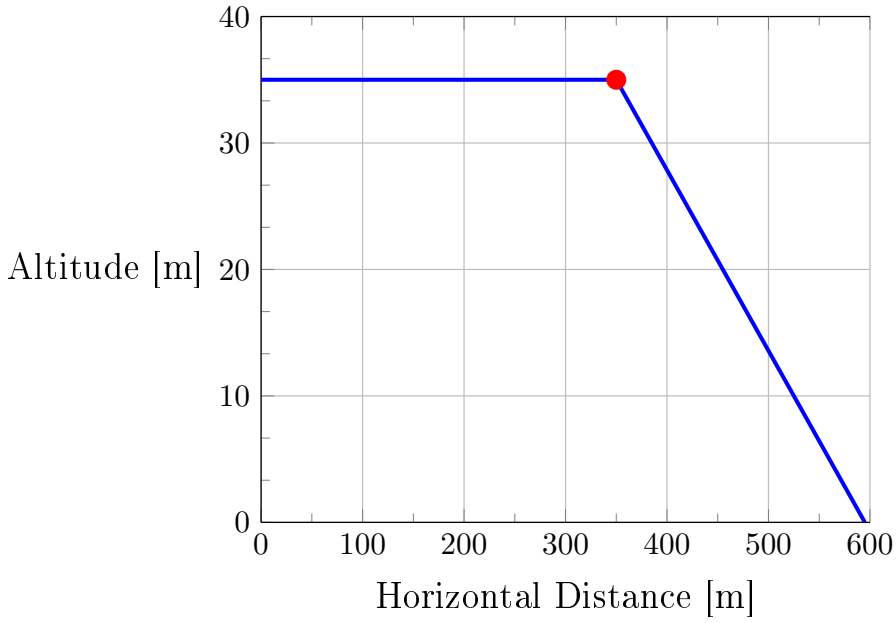


Figure 3.4: Trajectory before and after failure, fixed wing UAV

The failure corresponds to the red point, before it the UAV is nominal conditions at the established altitude, then it instantly passes to the gliding conditions described before. In case of a very high cruising speed this approach could lead to an underestimation of the crash distance, but this is compensated by considering a higher value of aerodynamic efficiency than the one expected.

3.1.3 Results: effects on risk map

The effects of the crash distance on the risk map are presented in this paragraph. It is expected that a smaller crash distance, involving a less number of cells, gives the effect of a risk map with higher risk difference between different regions in the map. A larger crash distance, instead, gives the effect of a more uniform risk map, with a higher risk.

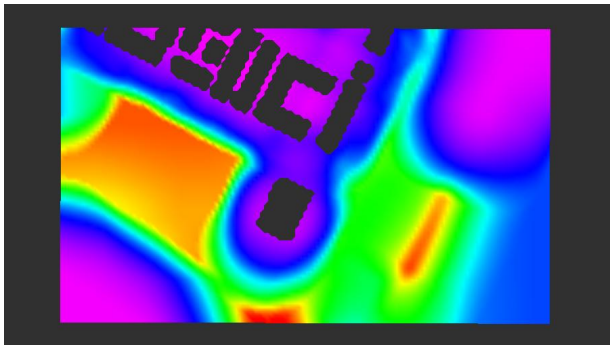


Figure 3.5: Risk map generated considering a crash distance of 60m

To purple and blue areas correspond smaller risk, to green and yellow areas correspond average risk, to red areas correspond the highest risk value. In this case, has been considered a crash distance of 60 meters. As expected, the risk map generated present very small and very large values of risk. The number of cells that contribute to the risk value of a sigle cell is small. Therefore the effect given by different values of sheltering factor or population density is stronger. Comparing this risk map to the sheltering factor layer corresponding to figure 1.6, it is possible to see that in cells where the sheltering factor is smaller, the risk evaluated is much larger. In cells where the sheltering factor is higher, the risk evaluated is much smaller.

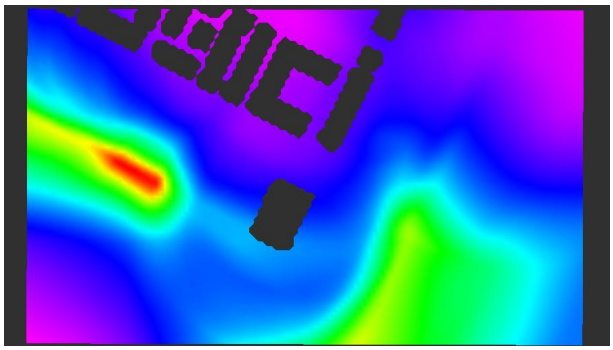


Figure 3.6: Risk map generated considering a crash distance of 120m

This map shows the risk value distribution obtained considering a crash distance of 120 meters. As expected the risk distribution is more uniform. It is important to notice that a more uniform risk does not correspond to a small risk value. Indeed, the total risk value is larger. A consideration that can be done is that, the highest risk value can be found where not only the cells have a small sheltering factor associated, but there is also a large presence of other neighbouring cells characterized by a small value of sheltering factor.

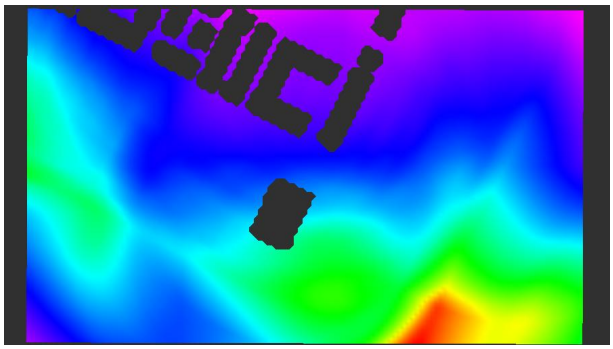


Figure 3.7: Risk map generated considering a crash distance of 180m

Last map shows the distribution of risk values for a crash distance of 180 meters. This distance is a very large gliding distance for a UAV. This condition can occur when flying at very high altitude. The map, as expected, present a high uniformity. The risk value differences are smaller than the previous cases. Moreover, in this case, the risk values found in cells close to the map borders, may be subjected to a bad evaluation, because the distribution of sheltering factor and population density in cells outside the borders may highly influence the risk value found. Also in this case, the uniformity of the risk map does not imply a smaller value of risk.

3.2 Rotary wing

In order to obtain the crash distance for rotary wing UAVs, have been numerically integrated the ballistic flight equations. As above described for the fixed wing UAVs, is supposed an instantaneous failure that prevent the complete operativity and control. In rotary wing aircrafts the lift is generated by several propellers (or rotors). It is reasonable to suppose that after a failure all the propellers stop rotating. Without any lift generated by propellers or any other surfaces, the aircraft is expected to follow a ballistic trajectory until the impact to the ground. The autorotation maneuver is excluded for two main reasons. The first reason in to find in the architecture of the UAV propulsion system, the propellers used to generate lift are directly connected to the electric motors without the presence of a freewheeling unit. Without the possibility of decoupling the rotations of the two components, the propeller rotation speed is proportional to the rotation speed of the electric motor, that in case of failure would be still. The second reason involves the ability to control the aircraft, the autorotation is a state reachable with a continuous control of the aircraft, which is missing for the hypothesis of a total failure of the system.

The hypotheses are summarized below:

- Instantaneous failure of all components
- Absence of external disturbances
- No autorotation
- No lifting surfaces other than propellers

3.2.1 Analytical formulation

In this section is explained the analytical approach followed to obtain the equation of motions that have been integrated numerically in order to evaluate the crash distance for rotary wing UAVs. Because of the hypotheses described above (in particular the absence of lifting surfaces) it was considered appropriate to use the ballistic flight equations, treating the UAV as a point-like particle located at the centre of mass, given its value of mass and drag coefficient C_D.

The flight is clearly in low subsonic regime, the Mach number is approximate to zero, therefore the compressibility effects have been neglected.

The forces acting on the body are described in the figure below:

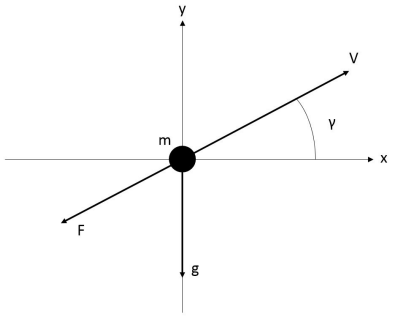


Figure 3.8: Forces on body

Considering a fixed reference frame of coordinates x y, with y in a direction parallel and opposite to gravity and x in the direction of the projection of the velocity on an axis perpendicular to y. The parameters shown in figure are:

- $\overrightarrow{V}$ is the velocity vector
- $\overrightarrow{g}$ is the gravity acceleration vector
- γ is the flight path angle or glide angle
- $\overrightarrow{F}$ is the drag force evaluated as $\frac{\rho \cdot V^2}{2} \cdot S \cdot C_D$

The equilibrium equation can be written as:

$$m \cdot \frac{d\overrightarrow{V}}{dt} = m \cdot \overrightarrow{g} - \frac{\rho \cdot V^2}{2} \cdot S \cdot C_D \cdot \frac{\overrightarrow{V}}{V}$$

Where:

- S is the reference surface
- $\frac{\overrightarrow{V}}{V}$ represents the direction of the velocity vector

The velocity vector by definition is:

$$\frac{d\overrightarrow{r}}{dt} = \overrightarrow{V}$$

With $\overrightarrow{r}$ position of the body in the reference frame (x, y).

The equation 3.2.1 is decomposed along tangential and normal directions:

$$\frac{dV}{dt} = -g \cdot \sin\gamma - \frac{\rho \cdot V^2}{2 \cdot m} \cdot S \cdot C_D$$

$$\frac{d\gamma}{dt} = -\frac{g \cdot \cos\gamma}{V}$$

It is important to remember that the velocity vector is expessed with $\overrightarrow{V}$, while its module or speed is expressed with V.

The equation 3.2.1 is projected along the directions x and y:

$$\frac{dx}{dt} = V \cdot \cos\gamma$$

$$\frac{dy}{dt} = V \cdot \sin\gamma$$

Adopting the forward finite differences method, the derivative are expressed as:

$$\frac{dV}{dt} = \frac{V_{i+1} - V_i}{\Delta t} + O(\Delta t)$$

$$\frac{d\gamma}{dt} = \frac{\gamma_{i+1} - \gamma_i}{\Delta t} + O(\Delta t)$$

$$\frac{dx}{dt} = \frac{x_{i+1} - x_i}{\Delta t} + O(\Delta t)$$

$$\frac{dy}{dt} = \frac{y_{i+1} - y_i}{\Delta t} + O(\Delta t)$$

The numerical approximation is $\epsilon = \Delta t$.

Substituting in the original equations, the desired expressions are obtained:

$$V_{i+1} = V_i - (g \cdot \sin\gamma_i + \frac{\rho \cdot S \cdot C_D}{2m} \cdot V_i^2) \cdot \Delta t$$

$$\gamma_{i+1} = \gamma_i - \frac{g}{V_i} \cdot \cos\gamma_i \cdot \Delta t$$

$$x_{i+1} = x_i + V_i \cdot \cos\gamma_i \cdot \Delta t$$

$$i_{i+1} = y_i + V_i \cdot \sin\gamma_i \cdot \Delta t$$

3.2.2 Results

In this section the results obtained by the numerical integration of the previous equations are presented.

Initially the trajectories depending on initial values of velocity and altitude have been studied. The integration step used to evaluate all solutions is $\Delta t = 10^{-4}$. Considering a UAV characterized by diameter of 0.25 m and mass 0.5 kg, the following trajectories are shown considering variation of velocity and altitude.

As first is convenient to study how the crash distance varies depending on the glide angle. Condidering an initial velocity module of 20, altitude 35, it is possible to study the trajectory and crash distance in function of the initial glide angle.The red trajectory corresponds to the ideal ballistic trajectory obtained without considering the effects of drag resistance. The blue trajectories correspond to drag coefficients of 0.5, 1.0 and 1.5.

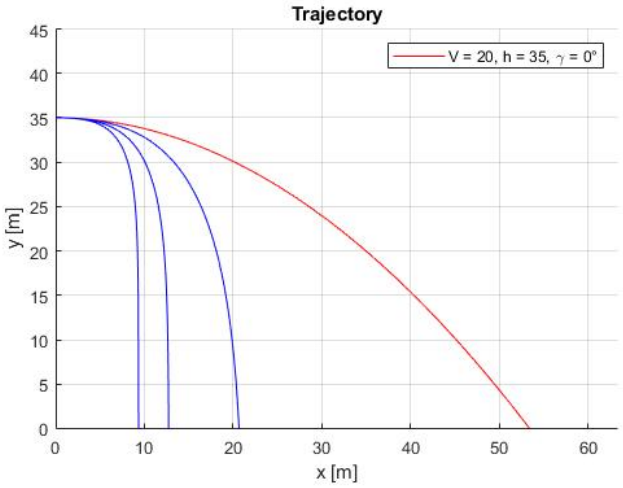


Figure 3.9: Trajectories for different values of drag coefficient for initial speed 20 m/s, altitude 35 m, diameter 0.25 m, mass 0.5 kg, gliding angle of 0 degrees.

For gliding angle of 0 degrees, the crash distances found are: 20.6 m (CD = 0.5), 12.7 m (CD = 1.0), 9.4 (CD = 1.5).

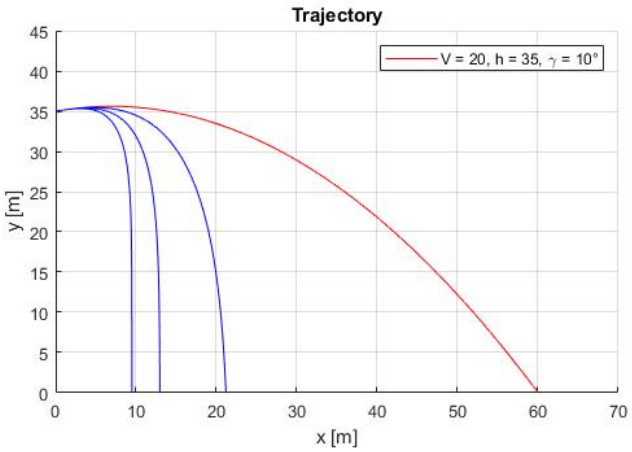


Figure 3.10: Trajectories for different values of drag coefficient for initial speed 20 m/s, altitude 35 m, diameter 0.25 m, mass 0.5 kg, gliding angle of 10 degrees.

For gliding angle of 10 degrees, the crash distances found are: 21 m (CD = 0.5), 12.9 m (CD = 1.0), 9.5 (CD = 1.5).

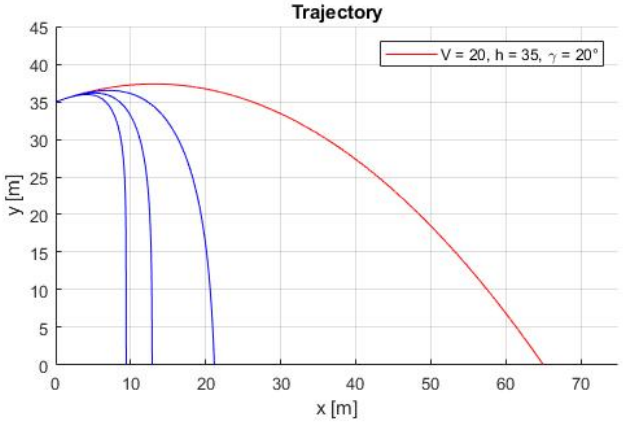


Figure 3.11: Trajectories for different values of drag coefficient for initial speed 20 m/s, altitude 35 m, diameter 0.25 m, mass 0.5 kg, gliding angle of 20 degrees.

For gliding angle of 20 degrees, the crash distances found are: 21 m (CD = 0.5), 12.9 m (CD = 1.0), 9.4 (CD = 1.5).

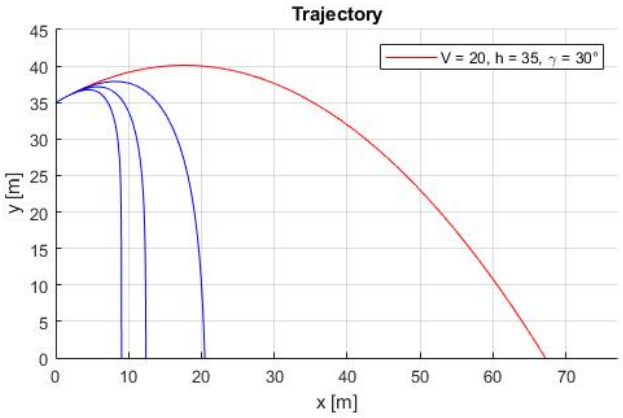


Figure 3.12: Trajectories for different values of drag coefficient for initial speed 20 m/s, altitude 35 m, diameter 0.25 m, mass 0.5 kg, gliding angle of 30 degrees.

For gliding angle of 30 degrees, the crash distances found are: 20.5 m (CD = 0.5), 12.4 m (CD = 1.0), 9.0 (CD = 1.5).

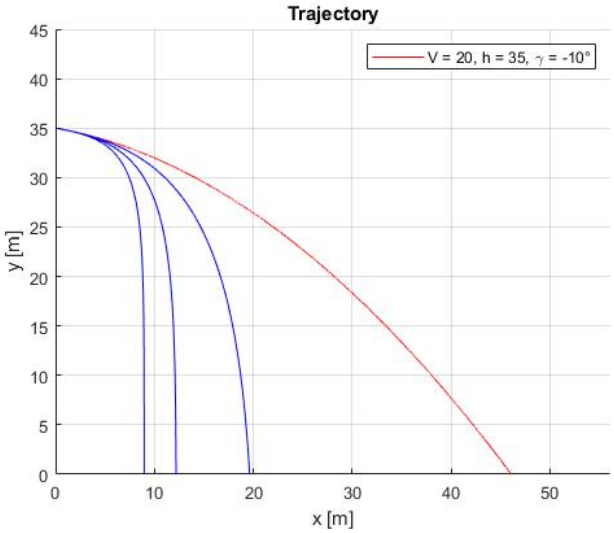


Figure 3.13: Trajectories for different values of drag coefficient for initial speed 20 m/s, altitude 35 m, diameter 0.25 m, mass 0.5 kg, gliding angle of -10 degrees.

For gliding angle of -10 degrees, the crash distances found are: 19.6 m (CD = 0.5), 12.2 m (CD = 1.0), 8.9 (CD = 1.5).

The results show that the difference between small differences of gliding angle are very small and can be neglected. Therefore a gliding angle of $\gamma = 0$ is adopted for the future results described. The table below shows many values of the crash distance in function of diameter, mass, drag coefficient and initial speed. The altitude considered is 20 meters.

	Diameter 0.15 m								
Speed	Mass 0.3 kg			Mass 0.5 m			Mass 1 kg		
[m/s]	CD 0.5	CD 1.0	CD 1.5	CD 0.5	CD 1.0	CD 1.5	CD 0.5	CD 1.0	CD 1.5
0	0	0	0	0	0	0	0	0	0
5	8	6.5	5.5	8.8	7.7	6.8	9.4	8.8	8.2
10	14.6	11.2	9.0	16.5	13.8	11.8	18.2	16.5	15.0
15	19.9	14.7	11.5	23.1	18.6	15.5	26.3	23.1	20.6
20	24.3	17.3	13.5	29.0	22.5	18.4	33.8	28.9	25.3
25	28.1	19.5	15.0	34.1	25.8	20.8	40.7	34.1	29.4
30	31.3	21.4	16.2	38.7	28.6	22.8	47.15	38.7	32.9

Table 3.1: Crash distances found for reference diameter of 0.15 meters.

	Diameter 0.25 m								
Speed	Mass 0.3 kg			Mass 0.5 m			Mass 1 kg		
[m/s]	CD 0.5	CD 1.0	CD 1.5	CD 0.5	CD 1.0	CD 1.5	CD 0.5	CD 1.0	CD 1.5
0	0	0	0	0	0	0	0	0	0
5	5.7	3.9	3.0	7	5.2	4.1	8.3	7.0	6.0
10	9.5	6.1	4.5	12.2	8.5	6.5	15.3	12.2	10.0
15	12.1	7.5	5.5	16.1	10.8	8.1	21.2	16.1	13.0
20	15.2	8.6	6.3	19.2	12.5	9.3	26.1	19.2	15.2
25	15.8	9.4	6.8	21.7	13.9	10.2	30.3	21.7	16.9
30	17.1	10.1	7.3	23.9	15.0	11.0	34.0	23.9	18.4

Table 3.2: Crash distances found for reference diamenter of 0.25 meters.

It is be possible to plot the crash distance in function of the initial speed. For example, considering a mass of 1.5 kg, drag coefficient of 1.5, diameter 0.45 m, altitude of 50 m, the crash distance in function of the initial speed is shown in the following image:

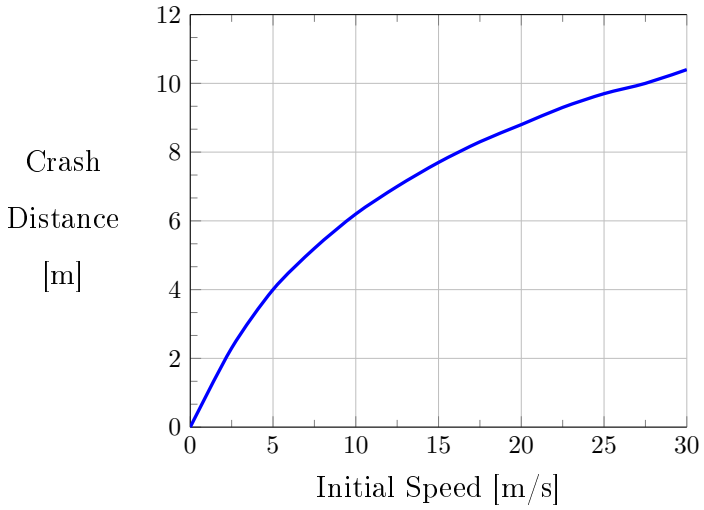


Figure 3.14: Crash distance as function of initial speed

In this way it is also possible to plot a trend line. In this case the trend line found corresponds to the equation: $X_C = -0.0111V^2 + 0.6456V + 0.6085$. This equation can be useful if the UAVs parameters considered in the path planning algorithm are well-known and do not change. Moreover, the flight altitude must remain constant and equal to the one considered to obtain the equation. If the UAVs considered by the path planner are many, or their characteristics may change, it is not useful to use this approach.

In order to have a more generic result, a different approach is more convenient. The best way considered to have the most generic results that can be used for all UAVs is to generate a table, where all values are already available, in function of all parameters of mass, altitude, initial speed, drag coefficient, diameter or reference surface. In this way, knowing the parameters and flight conditions of the UAV, the path planner may easily access the table and read the corresponding crash distance. A script that generates the table has been written using c++ language. An example of table generated is shown in the picture below.

Speed	Mass	CD	Altitude	Diameter	Crash Distamce
[m/s]	[kg]	[-]	[m]	[m]	[m]
0.5	0.1	0	10	0.1	0.7
0.5	0.1	0	10	0.3	0.7
0.5	0.1	0	10	0.5	0.7
0.5	0.5	0.5	10	0.1	0.7
3	0.3	0.25	20	0.3	4.2
5.5	1.7	1	30	0.1	12.0
18	0.7	1	45	0.1	39.0
23	0.7	0.5	55	0.3	22.0
25.5	0.9	0.25	50	0.3	42.1

Table 3.3: Example values from the table generated.

It is important to notice that the table contains the crash distance values evaluated with discrete steps of the five parameters described before. Is is possible to change these steps, using larger values to generate a smaller table, or using smaller values to generate more accurated results.

Chapter 4

Sheltering Factor Classification

As described above the sheltering factor is a real number that depends on the type of surface on which the UAV impacts after a failure. The value of the sheltering factor ranges theoretically from zero (no shelter) to infinity (best shelter). In case of land without any shelter, such as a field, an empty ground or a square, the sheltering factor value is zero. Then it increases with the presence of objects or obstacles, from a smaller value for sparse trees to an increasing value for dense vegetation, small buildings, tall buildings and industrial zones. Practically is more convenient to assume a limit value of ten, with the value of ten corresponding to an industrial zone. The value of sheltering factor is not just a pure number, but it is related to the impact energy required to have a fatal injury. The impact energy varies from tens Joule for no shelter to hundreds of thousand Joule for good shelter. The layer describing the sheltering factor is a matrix containing all the values of the cells in which the original map has been divided.

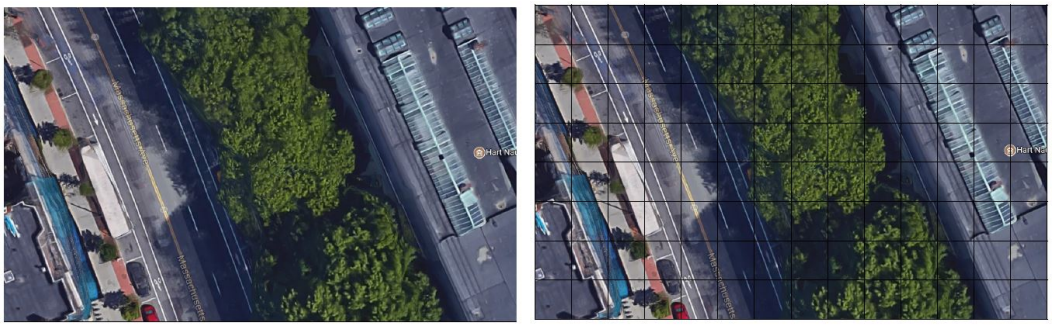
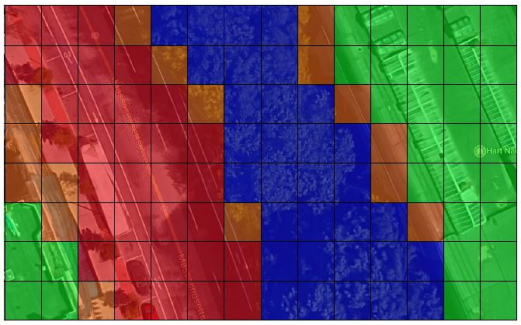


Figure 4.1: Map division into cells. On the left the original map, on the right the cells generated.

This matrix is then read by the risk algorithm that combines the layers and generates the risk map.



1 6 1 1 1 1 1 1 1 1 6 6
6 6 1 1 1 1 1 1 1 1 6 6
1 1 1 1 1 1 1 1 6 1 6 1
6 1 1 1 1 1 1 1 1 1 3 1
6 6 1 0 1 1 1 3 1 1 5 1
5 0 1 1 1 1 1 0 1 1 1 1
0 0 0 1 1 1 1 0 1 5 1 1
5 0 0 5 1 5 1 6 6 1 1 1

Figure 4.2: Sheltering factor layer. On the left the graphic representation, on the right an example of matrix with sheltering factor values.

This layer is not constant but slowly evolves over time, for example it has different values depending on the season, as is changes with the presence of foliage on the trees. The need to periodically update the layer suggests that an automation of the process is required. The advantage that, given a map, the values of sheltering factor are automatically assigned is clearly to replace the human work in favour of a software that can perform the same task in seconds or minutes. Briefly the process consists in: read the map, divide into smaller areas and assign to each of them a value, generating the sheltering factor matrix that will constitute the sheltering factor layer. While for a human the process is relatively easy, as it requires only a little knowledge, for a machine the process of classify images requires much more complex and elaborate capacities.

In recent years, has been spreading the use of artificial neural networks to solve problems of image classification. An artificial neural network (ANN) is a mathematical model that was made inspired by the behaviour of the neurons in the animal brain. The ANNs have been theorized for the first time in 1943 and have been slowly theoretically developed until the 2000s. Only in last decade ANNs have been used in practical applications. The main problem that has slowed down the development of this method in the last century is the high computational capacity they need for their training, available only recently with the introduction of parallel computing with GPU.

Therefore, in order to automate our task of classifying the maps, it was decided to use the artificial neural network method. The gain in terms of time necessary for the classification of a map is considerable, in fact the order of magnitude of the duration of the classification process goes from several minutes or hours for a classification made by a human, to seconds or minutes for a classification made by a trained neural network. However, the computational cost in terms of time that requires the training of the network should also be taken into consideration. In the best case a training can take a few hours, but it is more likely to take lots of hours, days or months. In fact, is probable that a training does not generate a sufficiently accurate network and it becomes necessary to change some of the learning parameters and re-train the network.

Two conditions have to be considered for which the automation of the process using neural networks can be convenient. The first condition is that it must be repeated for a high number of maps. The second condition is that it must be repeated frequently. Both these conditions are verified in smart cities, where detailed maps of the entire city are available and can be updated frequently.

Two strategies involving the artificial neural network method have been followed.

4.1 Phase 1: Simple Network

The first strategy is to build a small and light neural network. The benefits of this strategy are the small amount of time required to train the network and to classify the input data:

1. Task Identification: identify the task or the problem that has to be solved by the neural network. In this case it is a classification problem: we want to associate a category to an image;
2. Training Database Generation: generation of a database with a high number of data where there are both the inputs (in this case the images), and the corresponding outputs that are expected to be obtained (the categories corresponding to the images);
3. Model Architecture Definition: the architecture of the neural network is decided (type, quantity and position of layers in the network);
4. Model Production: following the determined architecture the model is written;
5. Training and Optimization: the model is trained and optimized until the accuracy reaches an acceptable level.

4.2 Phase 2: Transfer Learning

The second strategy involves using an already existing ANN and re-train it. The transfer learning is a machine learning method where a model developed for a task is used as starting point to generate a second model in order to perform another task. The approach used in this case is:

1. Task Identification: this point is the same already described for the first strategy.
2. Training Database Generation: database generation as already described.
3. Model selection: a pre-trained model is chosen among those available. Many universities or research institutions give open source models. A good criterion is to choose a model that performs a task similar to the one that has to be solved after the transfer learning.
4. Network Customization: is often necessary to modify the network to make it compatible with the database.
5. Transfer Learning and Optimization: the model is retrained and optimized.

Chapter 5

Machine Learning and Artificial Neural Networks

Machine learning is a modelling technique that gives the computer systems the ability to learn using data. Machine learning uses training data as input and it figures out a model. In most applications the input data are images, documents or audio.

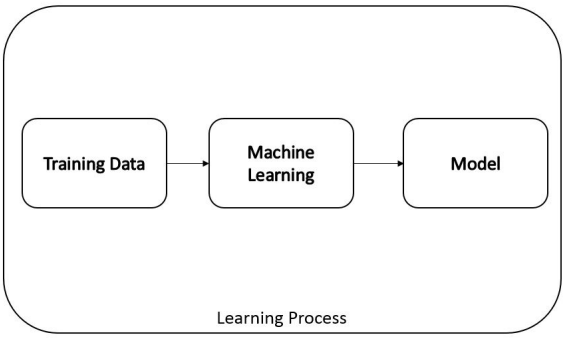


Figure 5.1: Learning Process

Once the model has been found it can be applied with the data we want to study, called field input data, giving the output. The field data is distinct from the training data used for the generation of the model.

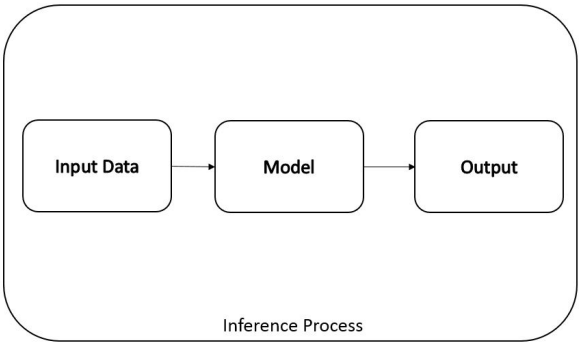


Figure 5.2: Inference Process

5.1 Types of Machine Learning

Machine learning techniques can be classified into three types depending on the training method: supervised learning, unsupervised learning, reinforcement learning. In supervised learning the training data is composed of input and correct output, the learning consists of modifying the model until it produces an output that corresponds to the correct output given with the input. The most common application are classification and regression, in classification the problem consists of finding the classes to which the data belongs (the outputs are discrete classes), in regression the problem consists of estimate a value from the input data (the output are values).

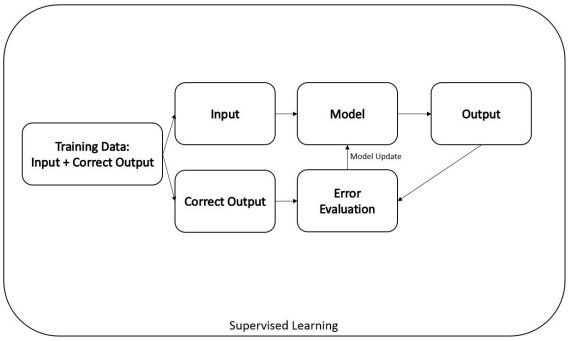


Figure 5.3: Supervised Learning

In the unsupervised learning, the training data contains only the input data, this method is used for investigating the characteristics of the data or for preprocessing. Reinforcement learning is usually applied for researching an optimal value, the training data consists in input, output and grade for the output. In this work the supervised learning technique has been implemented using the artificial neural network method.

5.2 Artificial Neural Networks

The neural network imitates the mechanism of the brain, the mathematical modelization of the neuron is called node or artificial neuron. Each node receives the signal from the previous nodes, process it and send it to the following nodes. The signal in an artificial neural network is a real number, the node receives the sum of the inputs and calculate the output with a non-linear function. Each connection has a weight that increases or decreases the strength of the signal at a connection and is updated as the learning process proceeds. It is possible that neurons have a threshold so that the signal is sent only if the threshold is reached.

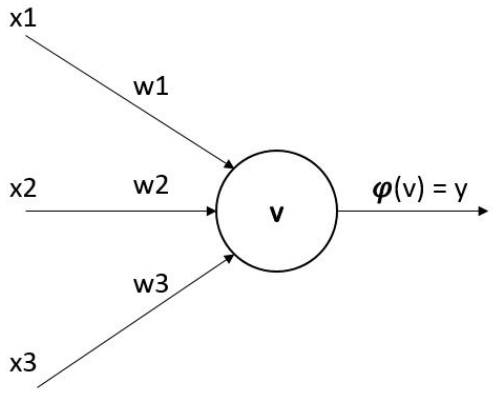


Figure 5.4: Single node with three connections

Considering a node that receives inputs and give an output y, the input signals are x_1, x_2, x_3; the weights are w_1, w_2, w_3. The signal that reaches the node is called weighted sum, and is calculated as: $v = w_1 * x_1 + w_2 * x_2 + w_3 * x_3$. The equation can be written in matricial form as: $v = W * X$. Where $W = [w_1\ w_2\ w_3]$, $x = [x_1\ x_2\ x_3]^T$. The node then processes the signal, its behavior is determined by the activation function φ. Then the output signal can be written as $y = \varphi(v) = \varphi(W * X)$. In most cases, a constant is be added to the weighted sum.

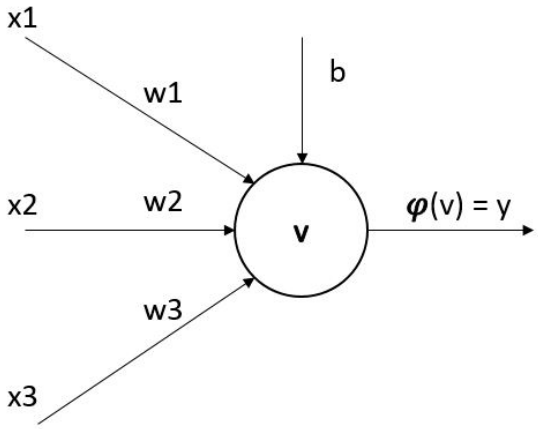


Figure 5.5: Single node with three connections and bias

The constant b is called bias, in this case the weighted sum is calculated as: $v = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + b$ and the output is $y = \varphi(v) = \varphi(W * X + b)$. The informations of the neural network is stored in the form of weights and bias. In other words, the training process consists of changing the values of W and b to obtain the outputs expected. Typically, artificial neurons are organized in layers, signals travel from the first layer to the last one. The layers between the input and output layers are called hidden layers. They are given this name because they are not accessible from the outside of the neural network.

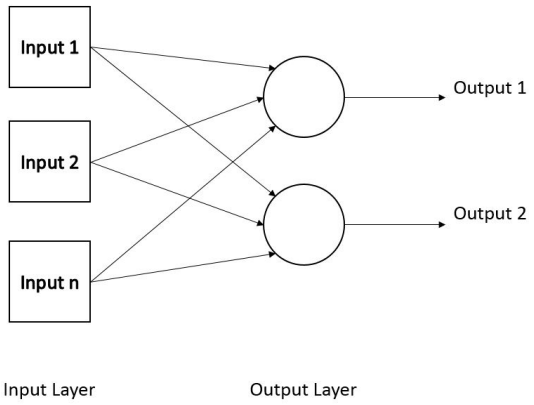


Figure 5.6: Single-layer Artificial Neural Network

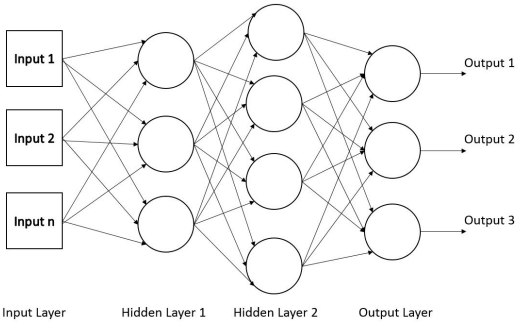


Figure 5.7: Deep Artificial Neural Network

The supervised learning process can be summarized with: the weights are initialized with adequate values, using the input data the output is evaluated, the error is calculated comparing the output with the correct output, the weights are updated to reduce the error. The algorithm which modifies the weights of the neural network is called learning rule, it can be expressed in equation as: $w_{ij} = w_{ij} + \Delta w_{ij}$. The official notation considers the weight w_{ij} as the one of the connection between the output node i and the input node j. An example of learning rule is the Delta rule. The Delta rule update the weights following this equation: $w_{ij} = w_{ij} + \alpha * \delta_i * x_j$, where :

- δ_i is defined as: $\delta_i = \varphi_i' * (v_i) * e_i$;
- α is the learning rate;
- φ_i' is the derivative of the activation function φ of the node i;

- e_i is the error of the output node defined as the difference between the correct output and the output of the network $e_i = d_i - y_i$ (with d_i known correct output).

The learning rate determines how much the weight is updated. This value is extremely important for the convergence of the learning process, if the value is too high the algorithm fails to converge, if the value is too low the calculation is too slow and requires an excessive amount of time to reach the solution.

Considering a single-layer network and a linear activation function $\varphi(x) = x$. the derivative of the activation function is $\varphi_i'(x) = 1$, then the learning rule becomes: $w_{ij} = w_{ij} + \alpha * e_i * x_j$.

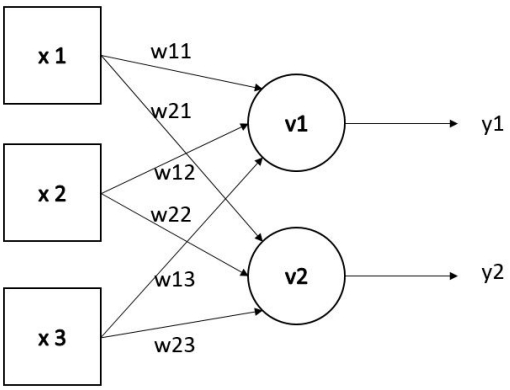


Figure 5.8: Single-layer Artificial Neural Network with weights

This network shown in the picture is composed of three input nodes that give the outputs x_j, six connections between the input layer and output layer (each connection has a weight w_{ij}) and two output nodes that give the outputs y_i. The Delta rule initially calculates the error as the difference between the correct output and the network output: $e_i = d_i - y_i$. Then it updates the weights with: $w_{ij} = w_{ij} + \alpha * e_i * x_j$. The following picture shows the steps above described:

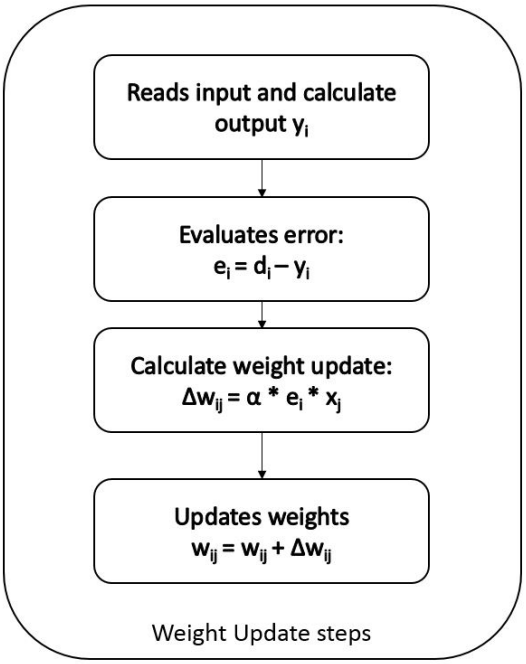


Figure 5.9: Weight update steps

These steps are repeated for all training data available. The execution of these steps for all the training dataset is called an epoch. The picture shows an epoch of the learning process:

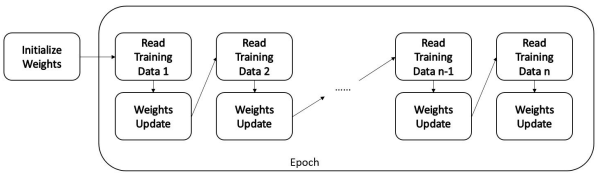


Figure 5.10: One Epoch

Eventually all this process must be repeated until the error reaches an acceptable value, or the maximum number of epochs has been reached.

Instead of using a linear activation function it is often convenient to use the sigmoid function:

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

.

The derivative of the Sigmoid function is: $\varphi'(x) = \varphi(x) * (1 - \varphi(x))$, then the delta is expressed with $\delta_i = \varphi'(v_i) * e_i = (1 - \varphi(v_i)) * e_i$. The learning rule becomes: $w_i = w_i + \alpha * \varphi(v_i) * (1 - \varphi(v_i)) * e_i * x_i$.

5.2.1 SGD, Batch and Mini-Batch

It is possible to use three different ways for the weight update: the Stochastic Gradient Descent method, the Batch method and Mini-Batch method. The Stochastic Gradient Descent (SGD) method calculates the error and updates the weights for each training data.

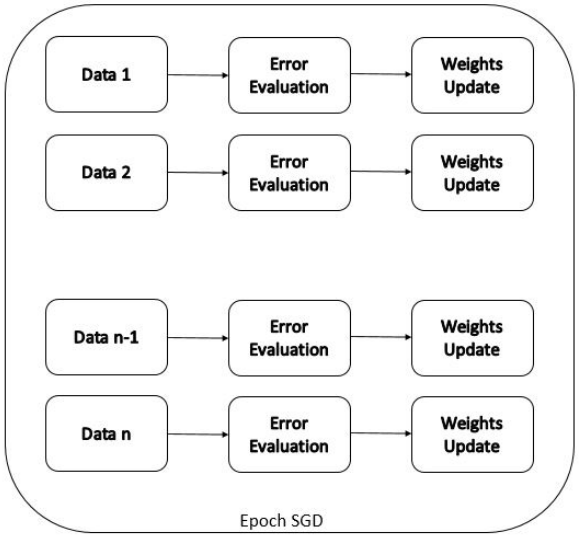


Figure 5.11: Stochastic Gradient Descent method

This method is the simplest to implement and, in most cases, also the faster (less epochs are required to reach the error required) but is more computationally expensive (the computation requires more time) and tends to be less stable.

The Batch Gradient Descent method calculates the error for each training data but updates the weights after all the errors have been evaluated.

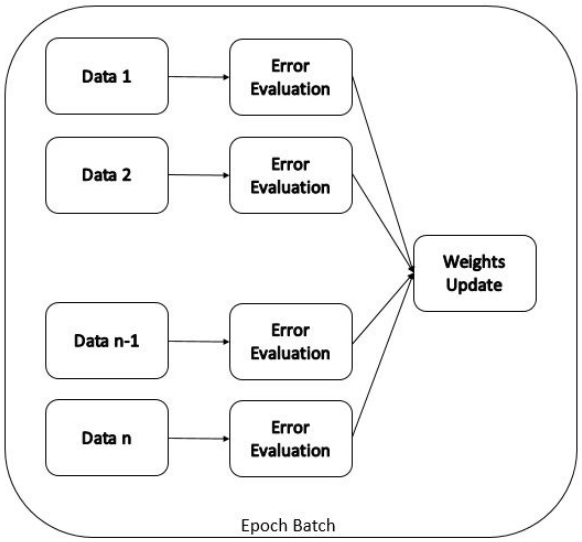


Figure 5.12: Batch method

The Batch method is more computationally efficient (less time required) than SGD, the algorithm is more stable due to the lower frequency of the updates. On the other hand, this

method is usually more complex to implement, and the convergence could lead to a less optimal values of weights than the SGD. All dataset must be in memory and available to the algorithm at the same time, this could be critical when using a large amount of data.

The Mini-Batch method is a combination of SGD and Batch. It splits the dataset into small batches, calculate the error for each training data and update the weights after the errors of all the data in the mini-batch have been evaluated.

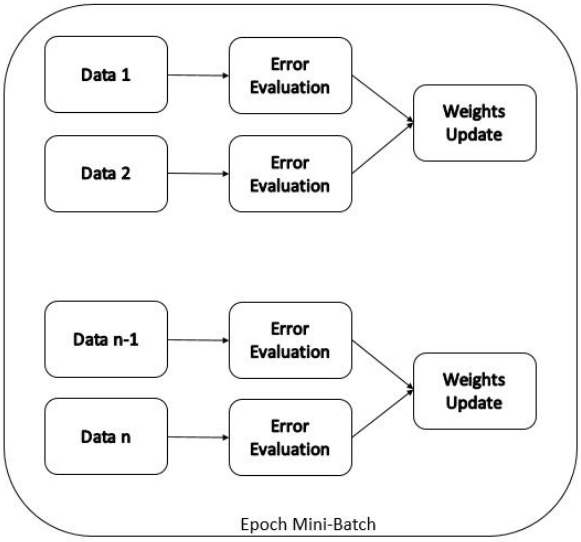


Figure 5.13: Mini-Batch method

This method is the most used in machine learning applications, because is a balance between the SGD and the Batch methods, it is more computationally efficient than SGD and does not require all training data to be available at the same time.

5.3 Multi-Layer Networks

Considering a multi-layer network made of an input layer, an output layer and one or more hidden layers. The Delta rule previously described is not valid because while the error of the output node is still defined as the difference between the correct output and the output of the network, it is not defined in the hidden layers. This happens because the correct outputs are given only for the last layer (output node) and not for the hidden ones. The problem of defining the error in the hidden layers is solved with the introduction of the back-propagation algorithm. The back propagation algorithm consist of, evaluated the error at the output node with the delta rule, propagating the errors backwards using the same procedure applied to calculate the output.

The steps followed by this method are:

1. Initialize weight values (for example with random values);
2. Calculate the weighted sum $V_1 = W_1 * X$ (V_1 is a vector with n_{hidden} elements, W_1 is a matrix $n_{hidden} * n_{input}$, X is a vector with the n_{input} inputs);
3. Calculate the output of the hidden layer $Y_1 = \varphi(V_1)$ (phi is the activation function, Y_1 vector with n_{hidden} elements;
4. Calculate the weighted sum $V_2 = W_2 * Y_1$;

5. Calculate the output Y_2;

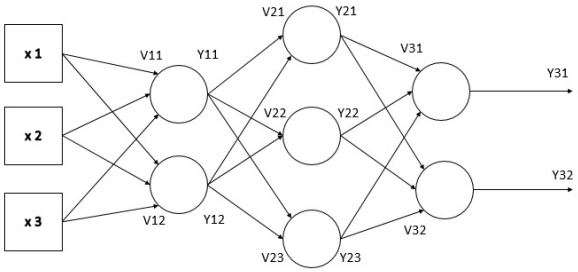


Figure 5.14: Forward pass in multi-layer networks.

6. Calculate the error $E_2 = D - Y_2$;
7. Use the delta rule $\delta_2 = \varphi(V_2) * E_2$;
8. Update the weights $W_2 = \alpha * \delta_2 * Y1$;
9. Calculate the error $E_1 = W_2 * \delta_2$;
10. Use the delta rule $\delta_1 = \varphi(V_1) * E_1$;
11. Update the weights $W = \alpha * \delta_1 * X$.

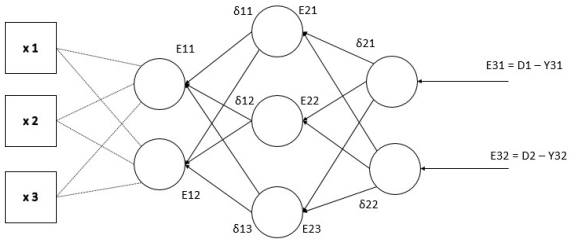


Figure 5.15: Back-propagation in multi-layer networks.

These steps are then repeated for every training data in the dataset, and until the error reaches an acceptable value or the maximum number of epochs has been reached.

5.4 Typical Neural Networks issues

5.4.1 Vanishing Gradient

The vanishing gradient problem is typical of neural networks based on the back-propagation algorithm, it occurs when, propagating backward, the error fails to reach the further nodes. This means that when updating the weights with $w_{ij} = w_{ij} + \Delta w_{ij}$, the value Δw_{ij} is so small that prevents the weights from changing. In the worst case this could stop the training of the neural network. One way to reduce the vanishing gradient problem is the use of the ReLU activation function and dropout layers.

5.4.2 Overfitting

The definition of overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably". This happens when a neural network is trained to classify in an extremely precise way all the training dataset, but fails to classify an other dataset. The overfitting of a network has two main different causes:

- the use of too many learnable parameters (a too high number of nodes);
- a bad training dataset.

Particularly for the CNNs a proper choice of the training dataset is of fundamental importance to reduce the possibility of overfitting. Another way to reduce it is using dropout layers.

5.4.3 Computational Cost

The computational cost is the time required to complete the training. It is proportional to the number of weights, then more nodes and layer there are, more time is needed. It is possible to decrease the computation cost using a high performance hardware (parallel computing with GPU) or using techniques known to reduce it (such as batch normalization or numerical optimization algorithms).

Chapter 6

Convolutional Neural Networks

An Artificial Neural Network that uses convolutional layers is called also Convolutional Neural Network (CNN). The main difference between the CNNs and other neural networks is that the CNN layers are conceptually two-dimensional or three-dimensional. The CNNs are specialized in image recognition, to do it they imitate how the visual cortex of the brain processes and recognizes images. Image recognition is in essence a classification problem, therefore the CNNs are a type of classification networks. The CNNs typical architecture can be divided into two segments: the Feature Extraction Network and the Classifier Network. The FEN consists of series of convolution and pooling layers. The convolution layer is similar to a group of digital filters, the pooling layers combine the neighbouring pixels into a single pixel (downsampling). While the convolution layers are learning layers (they contain weights and bias that are updated during the learning process), the pooling layers are used only to reduce the dimension of the image and do not contain weights. The Classifier Network usually consists in an "ordinary" fully connected neural network.

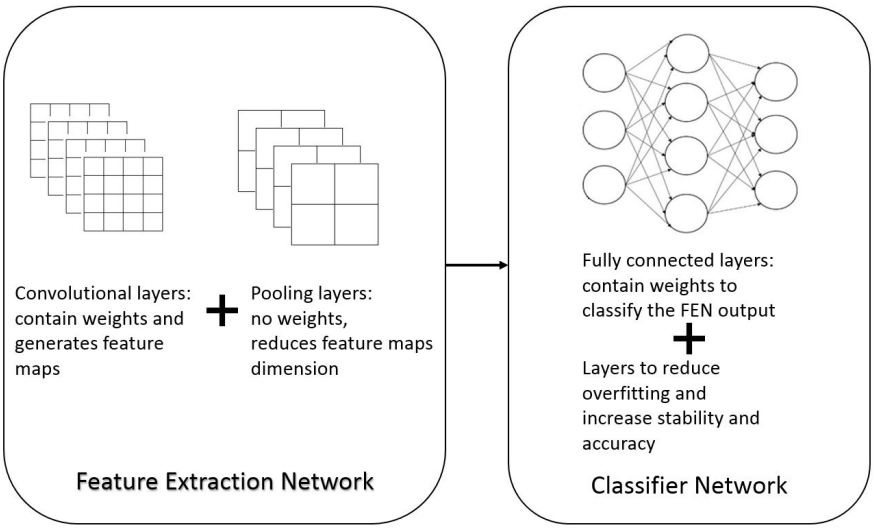


Figure 6.1: Main layers of Feature Extraction Network and Classifier Network

In this chapter all layers present in both Feature Extraction Network and Classifier Network are described.

6.1 Convolutional Layer

The convolutional layer or convolution layer generates new images called feature maps, it does not contain "traditional" weights but two-dimensional or three-dimensional filters called convolution filters. The number of feature maps generated are equal to the number of filters used. As general rule is possible to say that using a higher number of filters contribute to a better training. Considering an $n * n$ pixels image and some $m * m$ convolution filters ($m < n$), all the feature maps generated will have $(n - m + 1) * (n - m + 1)$ pixels. The filters are initialized at the beginning of the training (for example randomly) and are updated during the process; the final value of the filters is in effect the result of the training.

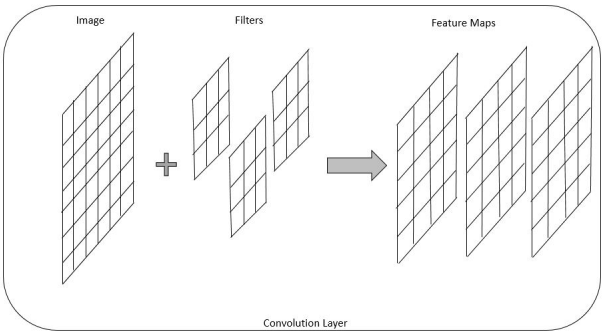


Figure 6.2: Convolution Layer: input image, filters and feature maps

The convolution operation is the sum of the products of the elements that are located on the same positions of the sub-matrix considered of the image and the filter.

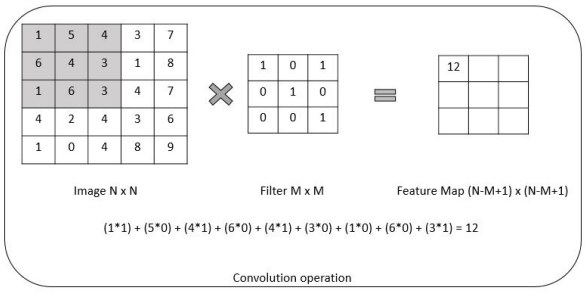


Figure 6.3: Convolution Operation

6.2 Pooling Layer

The pooling layer reduces the size of the image combining the neighbouring pixels, this operation has only the purpose to downscale the image. A pooling layer is not a learning layer, it does not contribute to the learning but has two very important purposes: it is essential for decreasing the time required by the process and increases the regularization of the model, preventing overfitting. The pooling layer uses a non-linear function, the most common function is the Max Pooling. The Max Pooling function cut the input image into a set of squares or rectangles and, for each one, gives as output the maximum value. A less used function is the Average Pooling function, which gives as output the average value instead of the maximum.

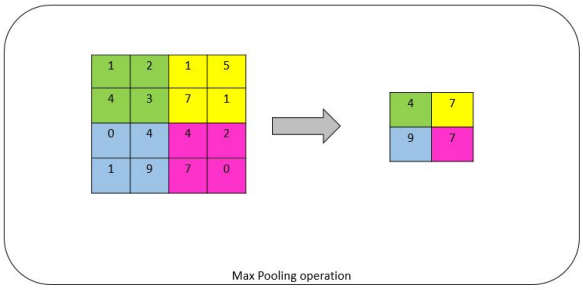


Figure 6.4: Max Pooling Operation

In CNN it is common to insert a pooling layer between two convolution layers. The most common filters used in pooling layers are composed of 2x2 or 3x3 pixels.

6.3 Rectifying Linear Unit (ReLU) Layer

The ReLU layers implements the Rectified Linear Unit function as activation function. This layer has been developed in order to avoid the vanishing gradient problem. It is often more used than the sigmoid function for the reason that it better transmit the error (during the back-propagation process). The ReLU function is defined as:

$$\varphi(x) = \begin{cases} \text{x}, & \text{if x} > 0 \\ 0, & \text{if x} < 0 \end{cases}$$

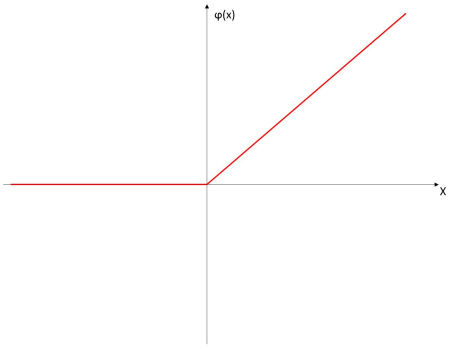


Figure 6.5: Rectifying Linear Unit function

It is intuitive to understand that also the implementation of this function is very easy.

6.4 Sigmoid Layer

The Sigmoid layer implements the Sigmoid function. This function is one of the most used as activation function in artificial neural networks. It this work this function has been implemented in chapter 7. The Sigmoid function is defined as:

$$\varphi(x) = S(x) = \frac{1}{1 + e^{-x}}$$

Sometimes it can be possible to find the Sigmoid function expressed as:

$$\varphi(x) = S(x) = \frac{e^x}{e^x + 1}$$

The two different expression are equivalent.

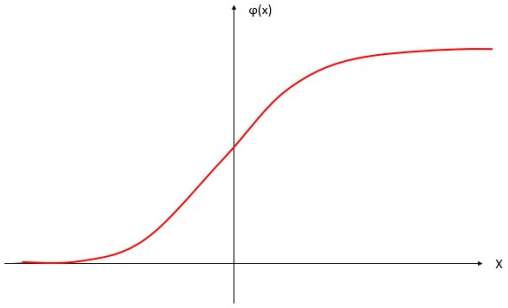


Figure 6.6: Sigmoid function

6.5 Normalization Layer

This layer reproduces the natural phenomenon of lateral inhibition. The lateral inhibition is the capacity of an excited neuron to reduce the activity of its neighbours increasing sensory perception (in the dark the contrast between the light and dark creates sharper shapes). For example, in optical illusions where different shades of grey are used, lateral inhibition makes the darker areas appear darker and the lighter areas lighter.

In a convolutional neural network the normalization layer change the output value of each node considering the outputs of adjacent channels. The expression used is the following:

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k+\alpha \sum\limits_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2\right)^\beta}$$

Where:

- n is the number of adjacent channels considered for the normalization;
- N is the total number of channels;
- k, α, β are constants called hyper-parameters.

It is important to notice that in this case the normalization occurs across channels, considering the value of the same pixel in different channels. It would be possible (but it is not used in this work) to normalize within the same channel (within channel normalization), considering the neighbourhood of a pixel in a single channel.

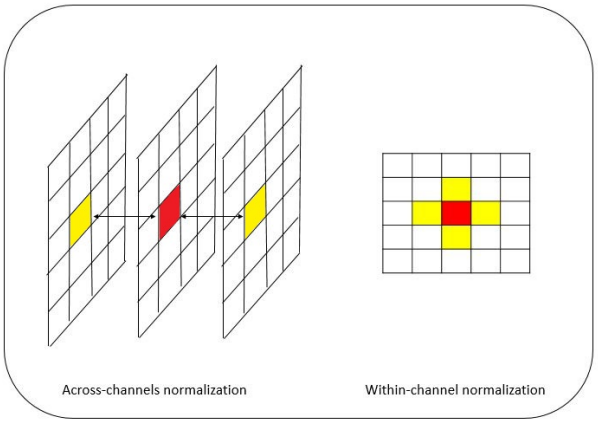


Figure 6.7: Cross channel and within channel normalization

6.6 Fully Connected Layer

This layer correspond to the "traditional" layer of a artificial neural network. In a Fully Connected (FC) layer, each node is connected to all the nodes of the previous layer. Each node receives as input the weighted sum of all artificial neurons in the previous layer and evaluate its output through the activation function. Its output is then used as input for all artificial neurons in the following Fully Connected layer. Given the large number of connections, this type of layer contains a large number of weights. The only parameter that can be changed in these layers is the number of artificial neurons. The optimal number of nodes has to be found with validation. It is not true that larger number of nodes, give better results. Instead it is true that lager number of nodes, will require a larger computational time to train. It is possible to represent a Fully Connected layer as a vector of K (number of nodes) elements. If K is the number of nodes of the FC layer considered and J is the number of nodes of the previous Fully Connected layer, the weight matrix associated to the considered layer have dimension of KxJ. FC layers are used in convolutional neural networks in order to classify the data from the Feature Extraction Network. The final output is a vector of dimension K, if K is the number of nodes of the final FC layer (the activation functions do not change the dimension of the output vector). The best representation of a Fully Connected layer is shown in the picture below.

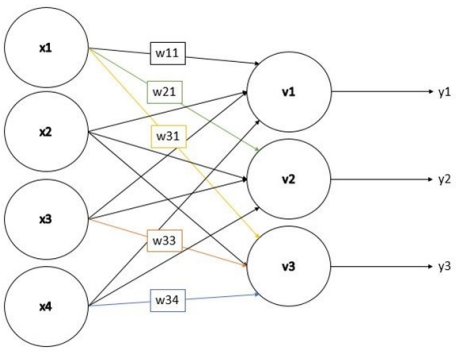


Figure 6.8: Fully Connected layer connected with the previous layer.

6.7 Dropout Layer

This layer uses the dropout technique to train only a partial number of nodes every iteration, in order to prevent overfitting. The technique consists of setting to zero the output of each hidden nodes with a probability of 0.5. In this way only the 50% of the nodes contribute to the calculation of the final output or the back-propagation. So, for every different input used, the network presents always a different architecture.

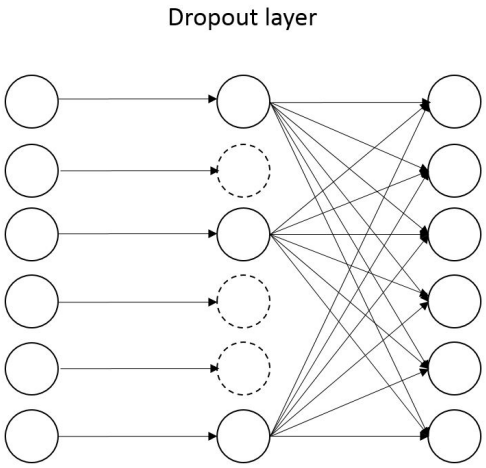


Figure 6.9: Dropout layer. A random group of artificial neurons is "switched off"

Chapter 7

Phase 1: Simple Network

This chapter describes the work done for the first strategy, involving the creation of an artificial neural network from scratch, in order to automatically evaluate the sheltering factor values of a given map. The firsts steps are the identification of the task (already extensively described) and the generation of the database. Then the network architecture has been defined. The criteria followed in order to choose the network architecture are:

- The network must be new: This is the important criterion characterizing this part of the work made for this thesis. For mainly didactic reasons it was decided to code the ANN without using already existing networks or parts of them. Obviously this choice implied a consistent amount of time in order to strictly follow this criterion, but with the advantage of putting into practice all the theoretical studies that preceded this section, with enormous personal satisfaction.
- The network must be light: The ANN written must be light in order to potentially work on any device. This peculiarity must concern the memory occupied in the mass storage (typically Hard Disk Drive or Solid-State Drive), the memory required in RAM, and the computational speed required by the Central Processing Unit. Following this criterion involves the possibility of using the network in any device and probably also on mobile devices. This criterion has been chosen with the idea of being able to classify a map directly on board the UAV. It is assumed that the hardware of a UAV does not allow to perform expensive operations. Using a light ANN with low requirements in terms of computational resources and time, could become possible to classify the maps in real time without sending data to the cloud system.
- Low accuracy of the results expected: As a consequence of the criteria described above, high accuracy in results is not required. Considering the advantages of lightness and speed of execution, it was not considered particularly restrictive to have a high accuracy of the results of the ANN.

It is important to remember that all criteria above listed, are used only for the strategy presented in this chapter.

7.1 Artificial Neural Network architecture definition

Adopting the most used configuration for image classification neural networks, it was decided to compose the main part of the network in two sections: the Feature Extraction Network (FEN) and the Classifier Network (CN). The final network will be made of:

- Input
  This section consists of one layer containing only the input data (the image considered).

- Feature Extraction Network (FEN)
  This section contains one or more two-dimensional layers in order to create feature maps with the purpose of emphasize the characteristics of the original image.
- Classifier Network (CN)
  This section contains one or more layers in order to classify the data coming from the FEN.
- Output
  This section consists of one layer containing the final output of the network. During the training the output stored in this layer is used to evaluate the error, while during the classification the output is the result of the classification process.

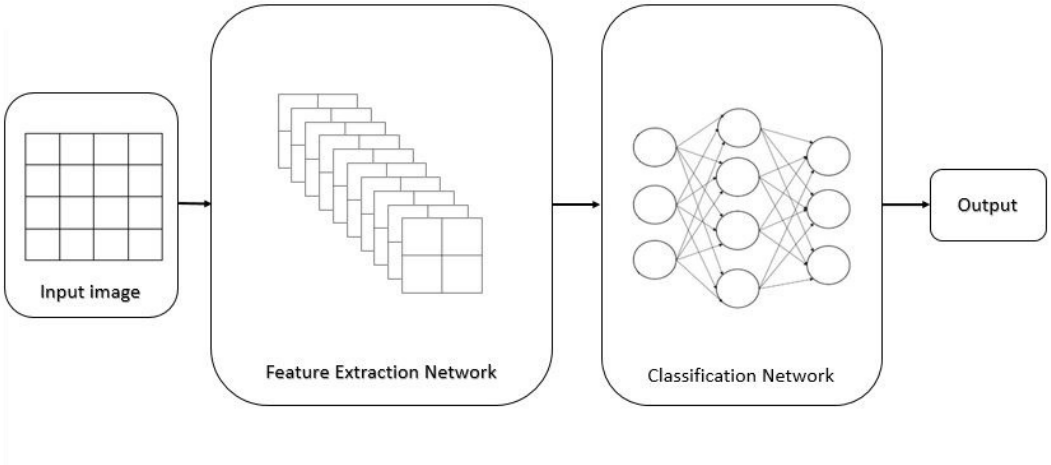


Figure 7.1: The Network Architecture

In the Feature Extraction Network it was decided to use convolutional layers, being the most used type of layers to treat the problem of image classification. To reduce the data size and decrease the training time, pooling layers has been used. The use of normalization layers has been avoided. This choice has been made considering that normalization layers contain hyperparameters that have to be determined by validation requiring a large amount of time to perform. In the Classifier Network it was decided to use fully connected layers. To reduce the problem of overfitting, it was decided to adopt the use of dropout layers. Two types of activation functions have been used: the ReLU and the Sigmoid activation functions. The Softmax activation function is adopted for the last layer as this function is the most suitable and most used in classification problems.

The detailed architecture of the network is described below:

- Layer 1: Input Layer

  This layer contains the input data of the network. The image has a size of N x N pixels. This value (N) is not fixed but can be chosen and can also be different from the size of the original image. The processing of the input data takes place in the pre-processing phase described later.

- Layer 2: Convolutional Layer (FEN)

  This layer performs the convolution operations on the image using k filters of M x M pixels. The filter size (M) and their number (k) are not fixed values but can be changed by the user.
  Downstream of this layer, k feature maps are generated with size $(N-M+1)$ x $(N-M+1)$.

- Layer 3: Activation Function (FEN)

  An activation function is used in this layer. It is possible to choose between the ReLU and the Sigmoid activation functions, both implemented (only one of the two function can be used, not both of them simultaneously).

- Layer 4: Pooling Layer (FEN)

  In this layer max pooling is performed using pooling matrices of dimension 2 x 2. The size of this matrices cannot be changed. This layer reduces the dimension of the k feature maps to a quarter of the number of pixels. The total number of the feature maps remains unchanged.

- Layer 5: Reshape Layer

  This layer allows the connection between the Feature Extraction Network and the Classifier Network. The task carried out by this layer is to transform the feature maps (treated as matrices) into a single vector. The size of the vector depends on the image size (N) and filters size (M) and amount (k). it is possible to obtain the vector length with the equation:

$$L = \frac{(N - M + 1)^2}{4} \cdot k$$

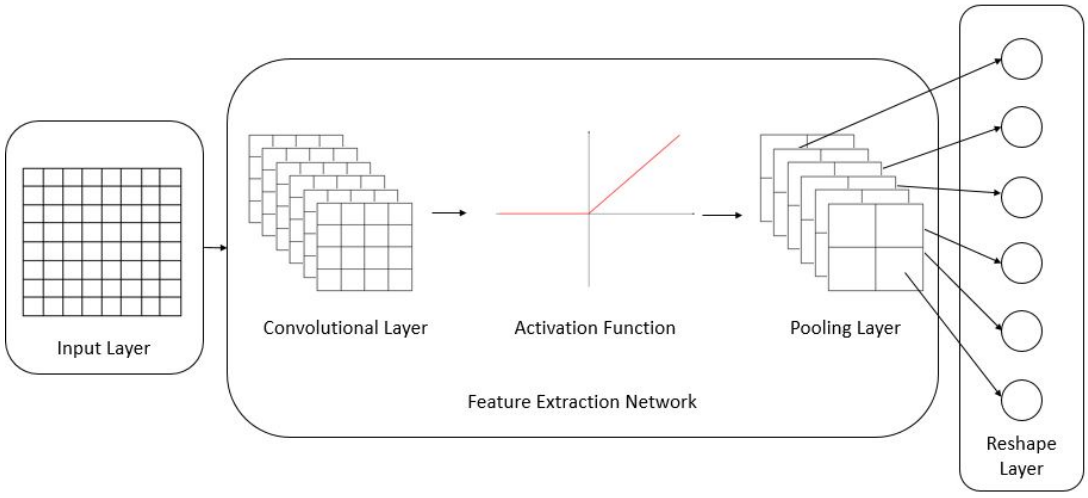


Figure 7.2: Input, FEN and Reshape Layer

- Layer 6: Fully Connected Layer (CN)

  This is the first fully connected hidden layer. It is composed of 100 artificial neurons. The weight matrix dimension is 100 x L. The layer receives L input values and generates 100 outputs. (L is the number obtained in the fifth layer)

- Layer 7: Activation Function (CN)

  The activation function is implemented in this layer. It is possible to choose between the ReLU activation function and the Sigmoid activation function, both implemented. This layer is identical to the third layer already described. It s important to notice that the activation functions can also be of two different types and not necessarily the same in both third and seventh layer.

- Layer 8: Dropout Layer (CN)

  This is a dropout layer, the output of a part of the artificial neurons of the previous layer will be neglected (forcing it to zero).
  The most common value used for dropout probability is 0.5. As initial value for has been considered a value of 0.1. Because of the small number of artificial neurons composing the layer, the best value of dropout was expected to be smaller than 0.5. This consideration has been refused by the results obtained in paragraph 7.3.2.

- Layer 9: Fully Connected layer (CN)

  This is the second fully connected hidden layer. It is composed of 100 artificial neurons. The weight matrix dimension is 100 x 100. The layer receives 100 inputs and generates 100 outputs

- Layer 10: ReLU Activation Function (CN)

  The ReLU activation function is implemented in this layer. Unlike third and seventh layers, it is not possible to choose the Sigmoid activation function. It has been chosen to use only the activation function to avoid the problem of the vanishing gradient.

- Layer 11: Layer Fully connected (CN)

  This is the third and last fully connected layer. It is composed of 8 artificial neurons. The number of artificial neurons in this layer has to be equal to the number of categories in which the images are to be classified. The weight matrix has dimension 8 x 100. The layer receives 100 inputs and generates 8 outputs.

- Layer 12: Softmax Activation Function (CN)

  The Softmax activation function is implemented in this layer, as it is the most suitable considering the output expected. The sum of the output given by this function is unitary ($\sum_i y_i = 1$, where i are the categories from "sheltering factor 0" to "sheltering factor 7").
  In this way it is possible to treat the output y_i of each category as the probability of the input to belong to it.

- Layer 13: Output Layer

  This layer contains as final output the result of the classification.

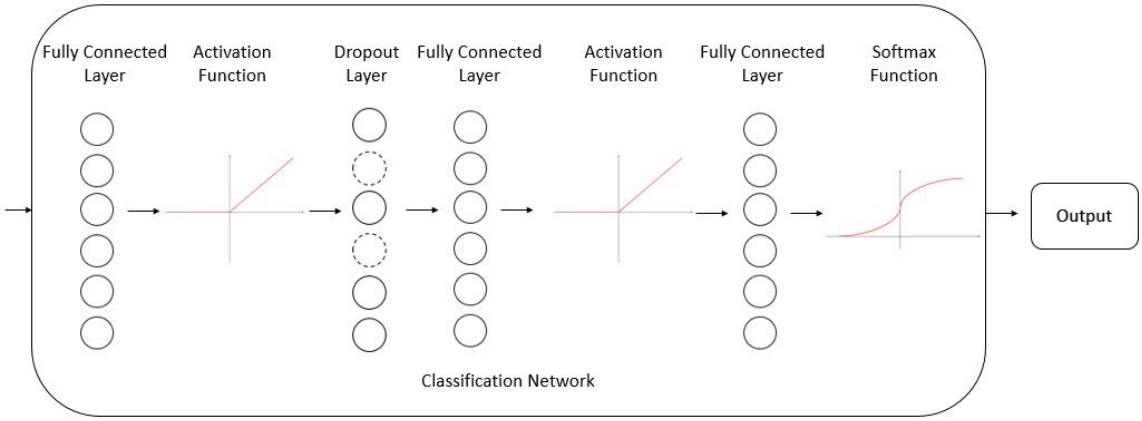


Figure 7.3: CN and output layer

7.2 The training algorithm

In this section are described all the steps followed by the script in order to generate and train the network. It is possible to identify three distinct parts of the training algorithm: pre-processing, training and post processing.

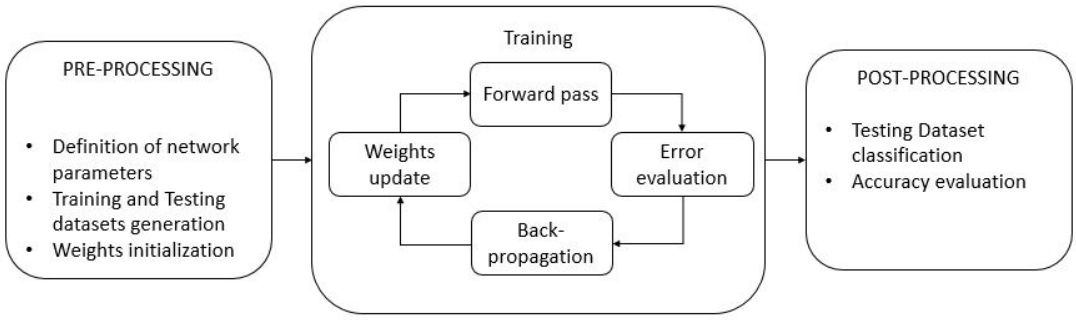


Figure 7.4: The training algorithm

7.2.1 Pre-processing

In the pre-processing section are defined the network parameters, the image are loaded and the weight matrices of the convolutional layer (the filters) and the fully connected layers are initialized.

Network and training parameters definition

First of all, the dimensional characteristics of the network are defined, including the size of the images, the size of the convolutional filters, the number of filters, the number of database images to be used for training and the number of database images to be used for testing. Secondly, all the training parameters are defined, including: number of epoch, mini-batch size, learning rate, momentum. The values of all these parameters are already predefined automatically, but can be changed by the user. Below are listed all the parameters defined with their preselected values.

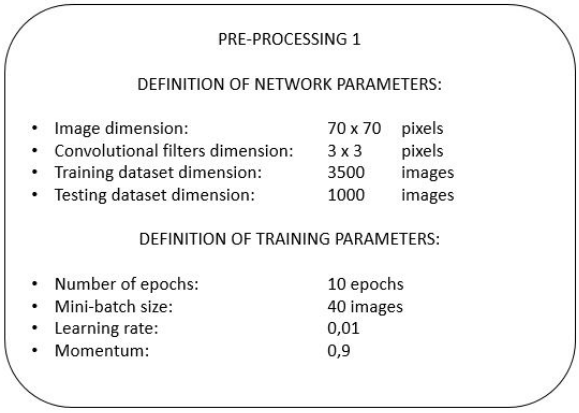


Figure 7.5: Network and training parameters definition with example values

Training and Testing datasets generation

All images of the database found in the specific folder are loaded into RAM memory. The images of the database used have original dimension of 80x80. They are then resized according to the previously defined value. The images are then shuffled and divided into the training dataset and the testing dataset. The number of elements in each dataset has been defined in the previous step. If the number of pixels of the images and the filters is not compatible, an error will be provided, automatically terminating the algorithm.

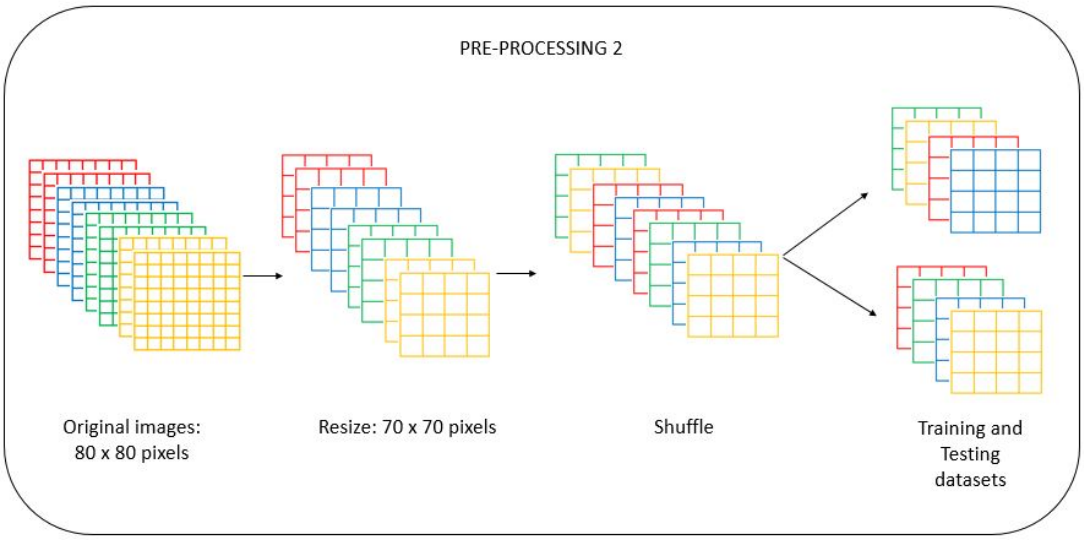


Figure 7.6: Training and Testing datasets. 1) Database loading. 2) Image resizing. 3) Shuffling in order to generate always different datasets for different training. 4)Split into training dataset and testing dataset.

Weights initialization

In this step the weight matrices of the convolutional and the fully connected layers are initialized. The convolutional layer matrix weights are initialized generating a random number between -1 and 1, which is divided by a factor of 10 (for convolutional weight) and 60 (fully connected weights), in order to have a small value of initial weights but different from zero.
This operation concludes the pre-processing phase.

7.2.2 Training

In this phase the actual training is performed, considering all the parameters chosen. As already described, the training consists in, given the input x, generating a vector of output y. The error is then evaluated comparing the output y with the result expected d (the value of d is present in the training database). Then the back-propagation of the error is performed in order to obtain the correction of the weight values (Δw). Considering this correction, the weight matrices are updated and the process repeated. This process is finished when the defined number of epochs is reached. A convergence check is implemented in this phase, in case of divergence an error will be provided and the training terminated automatically. It is important to notice that the convergence of the algorithm does not mean that the training has been successful (the final network may leads to a wrong classification), but only that a solution has been found.

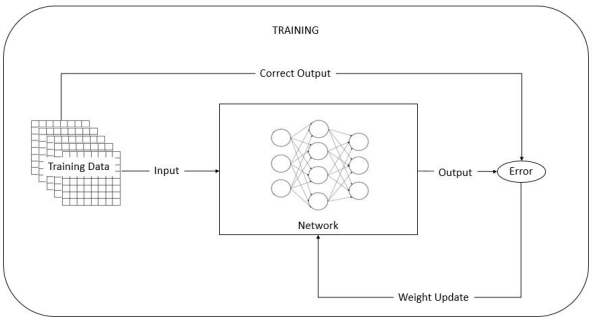


Figure 7.7: Training

7.2.3 Postprocessing

The postprocessing phase consists of evaluating the accuracy of the trained net. Considering the testing dataset, the network is used to classify each of the images contained in the dataset. Knowing the true category to which each of the images belongs, it is compared to the output category provided by the network. Considering the number of images for which the category provided by the network corresponds to the real category, and the total number of images, it is possible to calculate the accuracy of the training from the equation:

$$\text{Accuracy} = \frac{\text{Number of right predictions}}{\text{Total number of predictions}}$$

In this phase, the predicted classification output is considered correct if the input image belongs to the same category of the two neighboring categories of the output. For example, if an image belongs to "Sheltering Factor 5" category, the classification output will be considered right if the predicted category belongs to "Sheltering Factor 4" or "Sheltering Factor 5" or "Sheltering Factor 6". This approximation is possible because, two similar sheltering factor values, do not cause a high difference in the risk map. Indeed, attributing to the cells different but similar values of sheltering factor, affects the final result, in terms of risk map generated and best path evaluated by the path planning algorithm, very slightly. To better distinguish the two methods of calculating accuracy, the accuracy evaluated with this method is referred as approximate accuracy. When the accuracy is calculated considering a prediction right only if it coincides with the category to which the input belongs, the accuracy is referred as exact accuracy.

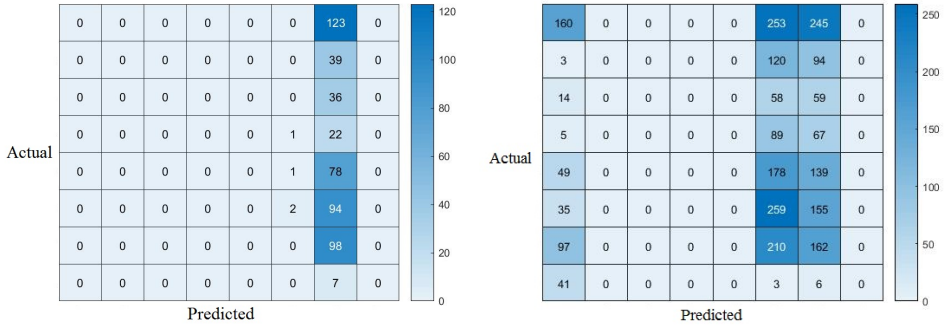
7.3 Results

In this section are presented the results obtained for the first neural network. Has been made a parametric study in order to choose the best parameters for the final version of the network. The parameters studied are shown in the list below:

- Learning Rate α
- Momentum β
- Filters Dimension M
- Number of filters k
- Mini-batch size
- Image dimension
- Dropout value
- Number of Epochs
- Number of images in the training dataset

7.3.1 Overfitting

Initially it was used only a fraction of the database, with the advantage to speed up the parametric study for choosing the best parameters. In all the following phases, a training dataset of one thousand images was adopted. Only during the last phase of the parametric study, the training dataset size has been changed, using the entire database. A condition that occurred in all the results is the presence of the overfitting problem. In the case of this network, overfitting occurs by generating a network that always provides the same results (the output of the network is always the same category) for any data present in the testing dataset. At first it would be possible to think that this problem depends on the stability and convergence of the algorithm. This assumption is not true, in fact it could happen also when a convergence with a very low error has been reached. The real cause of this problem is the randomness of the training and testing datasets. It is possible that in the training dataset generated, is presented a disproportion of data corresponding to a certain category. The network then will be trained in such a way to always give as output the category to which most of the data corresponds. If a similar data distribution is also present in the testing dataset, the accuracy value eventually found could also be very high. In case of presence of high overfitting, regardless of the accuracy value, the training has to be considered unsuccessful.
The best way to notice the presence of overfitting is inspecting the confusion matrix. A confusion matrix is a table that allows the visualization of the performance of a training algorithm. Each row represents the correct category in which an input data belongs, each coloums represent the predicted category. Some example of confusion matrices are shown below.



(a) (b)

Figure 7.8: High and Medium overfitting.

The 7.8a matrix shows a results that gives as output only one category. In this case the training process fails to obtain the optimal solution, reaching a local minimum different from the one searched. The 7.8b matrix shows a result that gives three category as output, in this case the solution found is better, but it cannot be considered acceptable.

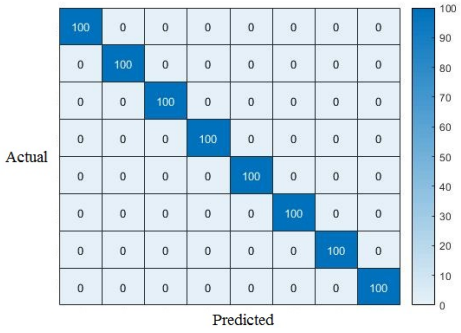


Figure 7.9: Best theoretical case.

This matrix shows the best theoretical result corresponding to a classification accuracy of 1. The confusion matrix generated in the testing process has been inspected to evaluate overfitting. In the results there is a qualitative indicator corresponding to a certain training. To High Overfitting, corresponds an output consisting of only one or two categories. This means that the confusion matrix will be made of one or two columns. To an Medium Overfitting, corresponds an output consisting of two or three categories, or a confusion matrix made of two or three columns. To Low Overfitting, corresponds an output consisting of more than three categories, or a confusion matrix made of more than three columns.

7.3.2 Parametric Study

Learning Rate α

The influence on the results of the learning rate value has been tested. From this parameter depends the weight update value. Also, the stability of the algorithm is heavily influenced by the learning rate. It is important to remember that the parametric study of alpha was performed keeping constant all the other parameters, including also the number of epochs. It is expected that for very high values, the algorithm fails to find a solution. Decreasing α, the algorithm is expected to become stable and the accuracy to be higher. From a certain value to lower, the accuracy is expected to decrease, because the learning rate is so small that it should be necessary to increase the number of epochs to reach the best solution. The optimum point refers to a configuration of weights that generates the minimum error value evaluated during the network training. The values of learning rate considered range from very high values (1.0) to very small values (0.0005). At first it can seen reasonable to assume that the best value of α is the highest value for which the algorithm is stable, as it leads to the solution faster. Actually, considering the highest stable value could lead to approach the optimum point of the solution without reaching it. A smaller learning rate could also lead to greater accuracy. However, this may require more epochs (and therefore computational time) to reach the solution. By adopting a very low alpha value it is also possible that the solution does not converge to an absolute optimal value, but to a local optimum.

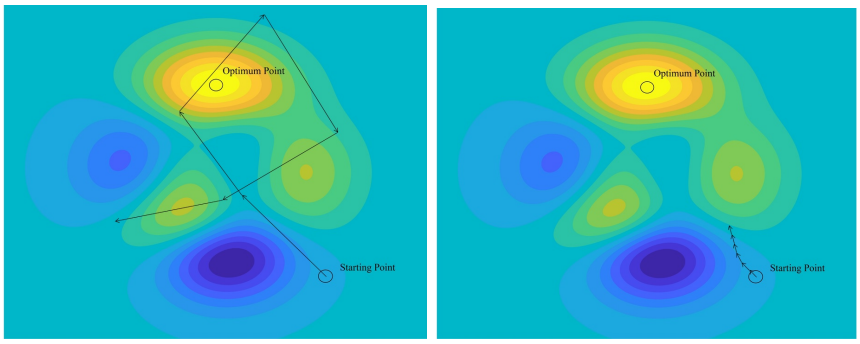


Figure 7.10: Too large and too small values of α.

These images show a qualitative representation of what happens for too high or to small values of α. In the first case the algorithm fail to find the solution after six iterations, in the second case the weight update is so small that after six iterations, the solution found is still too far from the optimum point. While in the first case is impossible to reach the optimum point, in the second case it could be possible to reach it by increasing the number of epochs. To better understand the effect of the learning rate it is possible to consider the following images:

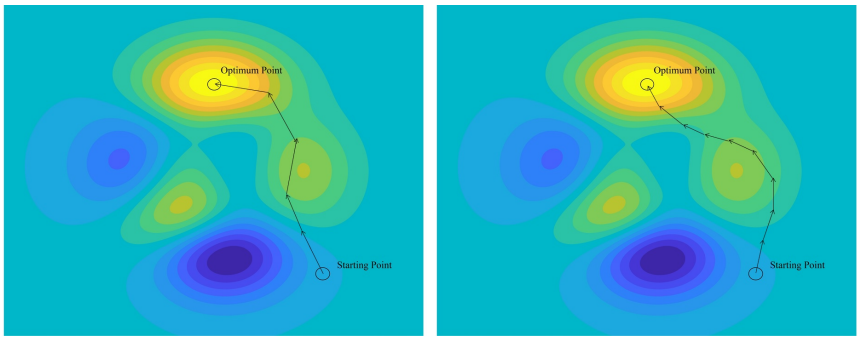


Figure 7.11: Large and small values of α that succeed to reach the optimum point.

The images show two training characterized by relatively large and small value of learning rate that succeed to reach the solution. It is easy to see that the training with larger learning rate, reaches the solution with a smaller number of iterations. On the other hand, the training with smaller learning rate needs more iterations and time to reach the optimum point. It can be said that, only comparing two equally successful trainings, the one characterized by higher value of learning rate is surely more convenient.

Considering higher values of α allows to quickly reach the solution with a smaller number of epochs. In this case a problem that can occur is that, even in case of stable algorithm, after a certain number of iterations the solution jumps around the optimal value without reaching it.

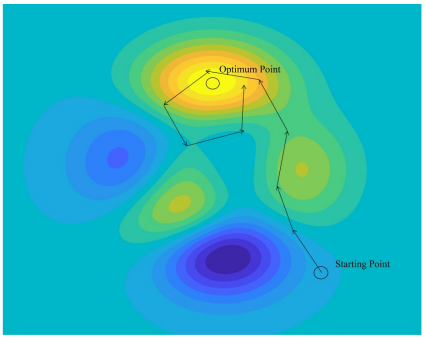


Figure 7.12: Training that jumps around the solution.

This image shows what could happen when a large learning rate is chosen and an iteration overshoot the optimum value. It is extremely important to notice that in this case the solution found may also be acceptable. Indeed, the training may jump around the optimum point without reaching it but still giving a sufficient accuracy. In this case, even increasing the number of epochs, the accuracy won't reach the optimum value as expected theoretically.

A very low value of α, leads to slowly reaching the optimum point, but higher precision may be achieved. There are two problems that may occur adopting a very low learning rate. The first is that the number of epochs (or the number of iterations) is not sufficient to reach the optimum solution.

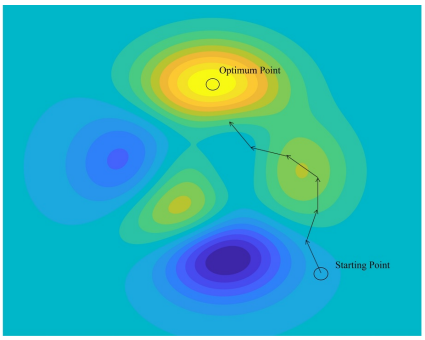


Figure 7.13: Training with small learning rate and insufficient number of epochs.

This image shows a training that fails to reach the optimum point because of a too small learning rate (or too small number of epochs). It important to notice that increasing the number of epochs can't be considered always a valid option to achieve better results. Indeed, by increasing the number of epochs, can be possible that the training time required becomes excessively large.

The second problem that may occur is that the point reached is not the one corresponding to the best solution, but to a local minimum point.

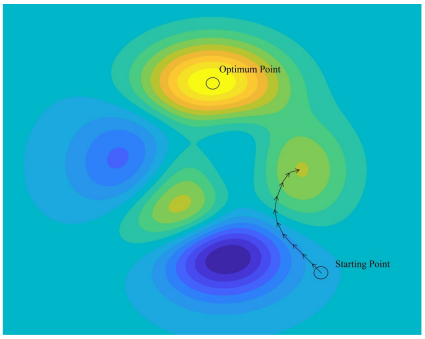


Figure 7.14: Training with small learning rate that reaches a local minimum point.

The image shows a training where a too small value of learning rate has been chosen. The solution found does not correspond to the optimal value but a local minimum. This result can be avoided by choosing a higher value of learning rate, or possibly by changing the momentum value (this parameter will be described in the next paragraph)

The results confirm what was expected. For alpha values equal to 1 and 0.5 (extremely high values for a training), the algorithm is unstable and the training fails. For lower values the algorithm becomes stable, as described before. The maximum value of the accuracy is reached for an α value of 0.005. To this value corresponds both the highest accuracy and lowest overfitting found in these results. This value was therefore considered the best value and chosen as final parameter.

The table shows the results obtained by changing the value of α. It is important to notice that all other training parameters haven't been studied yet. Depending also on all parameters studied in the following steps, in these results the overfitting is very high.

Learning rate	Accuracy	Time [s]	Overfitting
1	Unstable	-	-
0.5	Unstable	-	-
0.1	0.35	99	high
0.05	0.36	97	high
0.01	0.35	101	high
0.005	0.48	101	mid
0.001	0.35	102	high
0.0005	0.40	99	mid

Table 7.1: Parametric study results changing α value

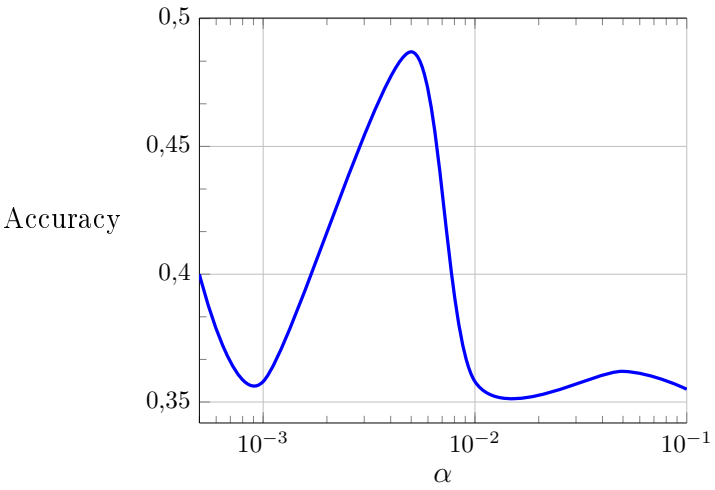


Figure 7.15: Accuracy in function of the learning rate α

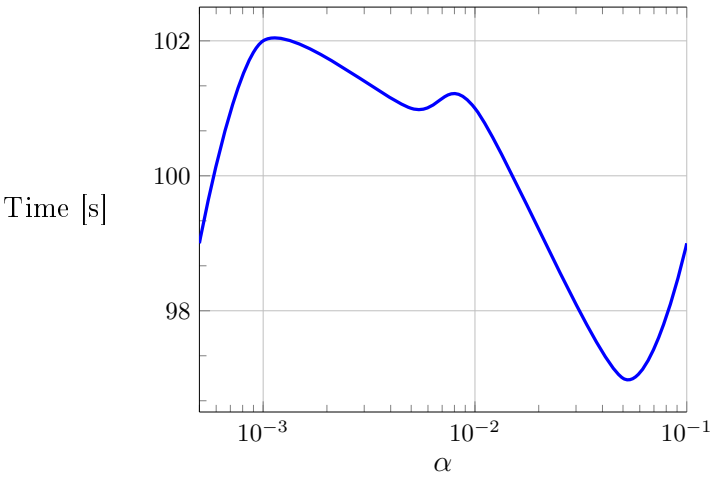


Figure 7.16: Computational time in function of the learning rate α

Momentum β

The value of β also affects the capacity of the network to reach higher accuracies. The momentum parameter is used to avoid the situation of reaching a local minimum point. Theoretically, choosing a high value of momentum may increase the stability of the algorithm. Moreover, having a large value of momentum means that the convergence may be be faster. Instead, a small value of momentum, could make the training not able to avoid local minima or slower. As general rule, if the momentum is large, the learning rate should be kept small. It is expected then that too small or too large beta values may lead to low accuracy, with instability in case of very small values. Two cases with large and small β are shown in the following qualitative images:

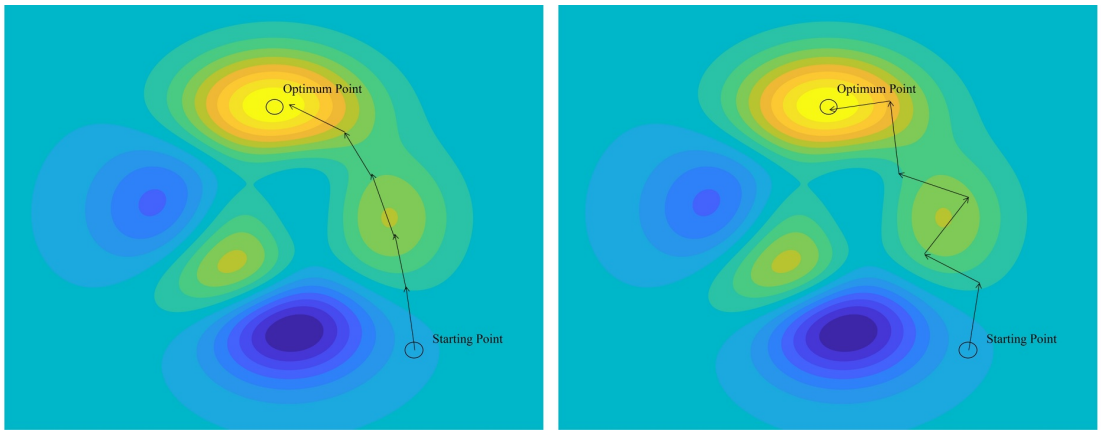


Figure 7.17: Successful trainings with large and small momentum values.

The first picture shows a training with larger momentum value, it can be seen that the training is more stable and it successfully avoid local minima. The second picture shows a training with smaller momentum value, in this case the algorithm is less stable, the training is slower and local minima are not avoided reliably. Considering a larger momentum value, the direction of the gradient at the i-th step, differs less from the direction of the i-1-th step. Instead, considering a smaller value, the direction of the gradient of the new iteration may differ more. However, choosing a too large value of momentum, may lead to overshooting the optimal solution. Instead choosing a value too small, may lead to reach a point of local minimum or instability.

Learning rate	Accuracy	Time [s]	Overfitting
0.95	0.34	97	high
0.9	0.36	98	high
0.85	0.45	99	high
0.8	0.36	98	high
0.75	0.37	100	mid
0.7	0.37	99	high
0.65	0.35	100	high

Table 7.2: Parametric study results changing β value

Results confirm in part what expected. High β values seems to lead to a better solution with high accuracy. In fact these solution have to discarded because of the high overfitting of the network generated. For lower β values, accuracy does not vary much as expected. The reason is

that, having chosen a low alpha value, the value of momentum (unless an extremely high or low value is used) do not influences highly the results of the network. Excessively low momentum values were not considered. The best value of β was considered that for which the overfitting phenomenon was smaller.

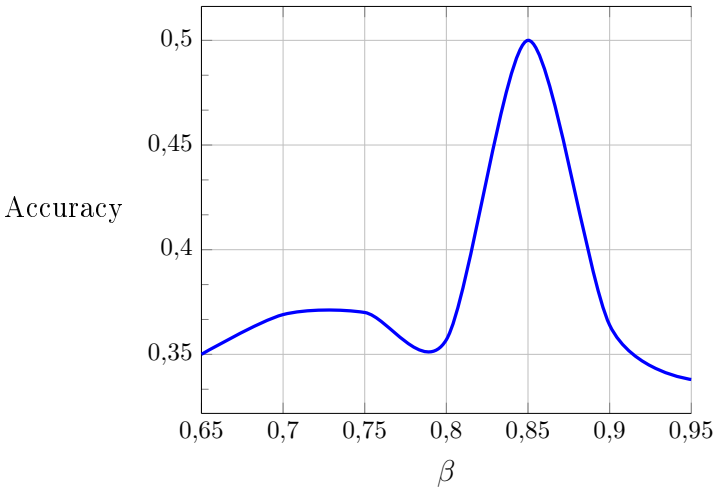


Figure 7.18: Accuracy in function of momentum β

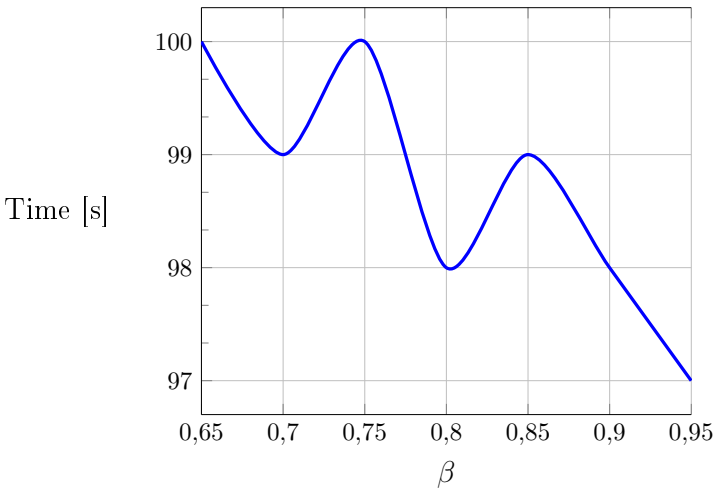


Figure 7.19: Computational time in function of momentum β

Filters Dimension M

The filters dimension influences two factors: the size of the feature maps and the size of the weight matrices associated with the convolutional layer. By increasing the dimension of the filters, the dimension of the feature maps decreases. It would seem then that the time needed for the training will also decrease. This is not true because to a higher dimension of the filters, corresponds a higher dimension of the convolutional layer weight matrix. In fact, the computational time seems to be constant or increasing slightly with the increase of the filters dimension. Using a higher value of filters dimension is expected to increase the accuracy of network constantly. The following table lists some examples of dimension of feature maps and number of weights depending on the size of the filters.

Filters Dimension	Accuracy	Time [s]	Overfitting
3	0.38	100	mid
5	0.36	104	mid
7	0.38	104	mid
9	0.51	94	mid
11	0.51	114	high
13	0.37	115	mid
15	0.39	117	mid

Table 7.3: Parametric study results changing filters dimension

Values are obtained with image dimension of 80 x 80 pixels (N x N), and 9 filters (k). As the values of N and k change, the previous table will obviously have different values. The computational cost in terms of time, does not have a significant variation, varying slightly with the filters dimension. Unlike from what expected, increasing filters dimension (and then number of weights) does not lead to a constant increase in the accuracy. Indeed the maximum value of accuracy found corresponds to a filters dimension M of nine or eleven. It was therefore chosen the value of 9 because the results presented less overfitting.

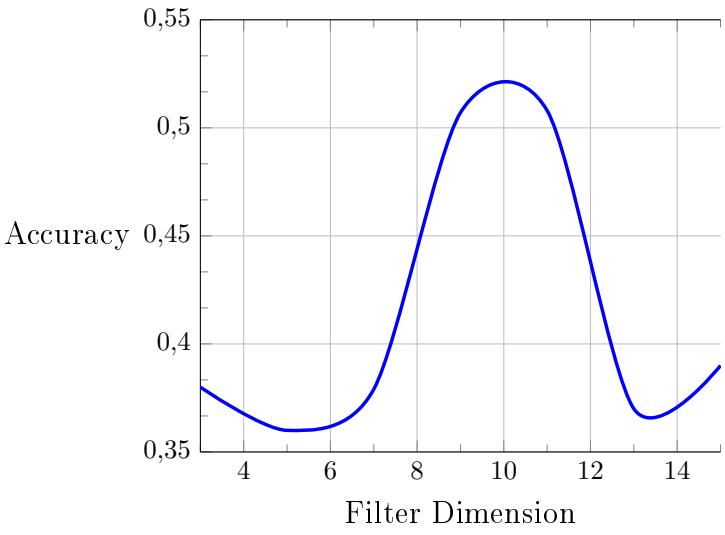


Figure 7.20: Accuracy in function of filters dimension

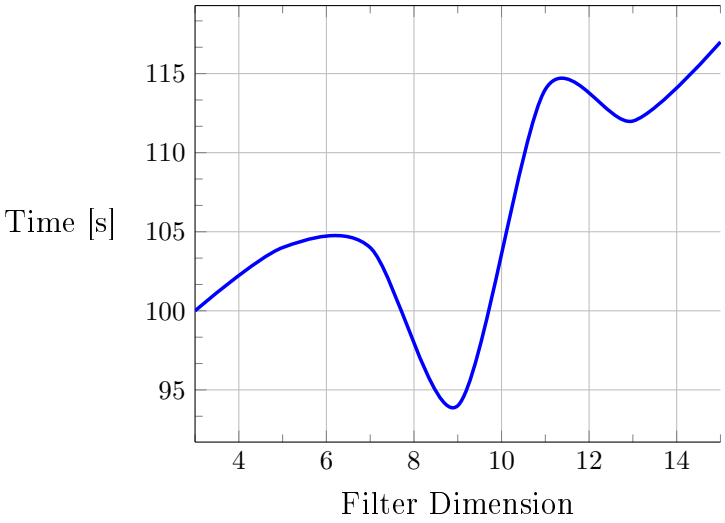


Figure 7.21: Computational time in function of filters dimension

Number of Filters k

The number of filters of the convolutional layer (k) influence proportionally the number of feature maps generated downwards of the FEN (Feature Extraction Network). In this network the proportion constant is equal to one, then the number of feature maps generates is exactly equal to the number of filters. It is expected that both the accuracy and the computational time increase with a higher number of filters. More precisely, is expected a slight increase of the accuracy and a significant increase of the computational time.

Filters	Accuracy	Time [s]	Overfitting
1	0.36	7	high
5	0.33	36	high
9	0.41	63	mid
11	0.38	75	mid
15	0.47	100	mid/low
19	0.42	131	mid
21	0.40	154	mid
25	0.35	174	high
29	0.33	199	high

Table 7.4: Parametric study results changing number of filters k

The results show that, as expected, the computation time increases with the number of filters. Instead, the accuracy of the network initially increases to a highest value for k equal to 15, then decreases for higher values of k. This trend of accuracy, characterized by reaching a peak for a given value, is a result that often occurs in this study. This trend is very useful because, with a parametric study, it is easy to find the best value of the parameters considered. In addition, also the overfitting is heavily influenced by the number of filters k. In particular, for low values ($k = 1$, $k = 5$), the overfitting is vey high (the confusion matrix is made of only 1 column). The same considerations have to be done for high values of k ($k = 25$, $k = 29$). For intermediate values, which correspond also to a higher accuracy, the overfitting is reduced.

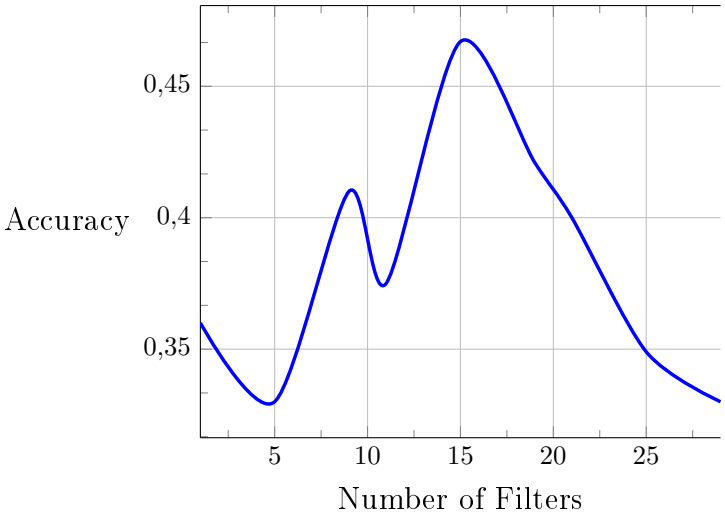


Figure 7.22: Accuracy in function of number of filters

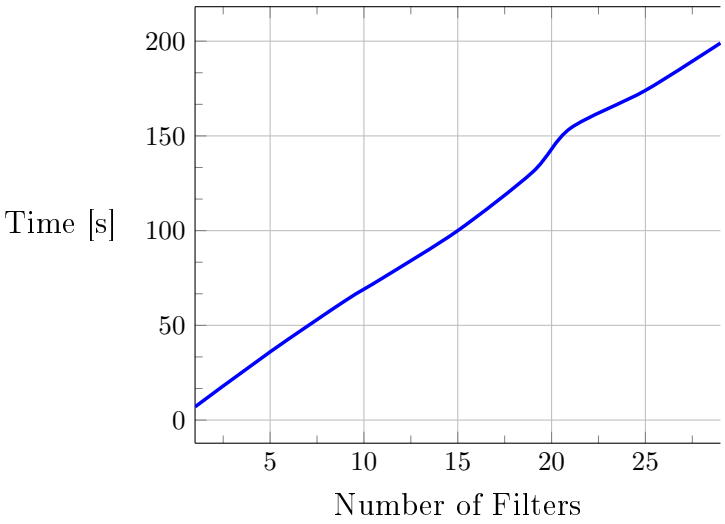


Figure 7.23: Accuracy in function of number of filters

Mini-Batch Size

The Mini-Batch Size value indicates the number of input data (and errors) evaluated before a weight update is performed. Theoretically, a low number of Mini-Batch Size corresponds to a less stable algorithm with a high computational cost but may lead to a better result in case of convergence. It is expected that a higher value tends to stabilize the algorithm and reduce the computational time, but may lead to lower accuracy.

Mini-batch size	Accuracy	Time [s]	Overfitting
1	0.34	175	high
10	0.42	114	low
20	0.39	101	mid
30	0.34	99	high
50	0.35	108	high
75	0.50	101	high
100	0.38	114	high
125	0.39	111	mid
150	0.34	92	high
250	0.37	108	mid

Table 7.5: Parametric study results changing Mini-batch size

The results show that the trends of accuracy and computational costs do not vary as much as expected. Instead the overfitting is highly influenced. In particular, for higher values the overfitting tends to be higher (therefore worse results), while for low values it tends to be lower (therefore better results). Different considerations have to be done for the case of Mini-Batch Size equal to one (this corresponds to a weight update every input evaluated). In fact, a much higher calculation time was required with high overfitting. This is due to the tendency of the algorithm to be unstable for very low values of Mini-Batch Size. Other than in case of Mini-Batch Size equal to 1, instability has not been observed. Therefore the value of 10 has been considered as the best, because of its high accuracy and low overfitting.

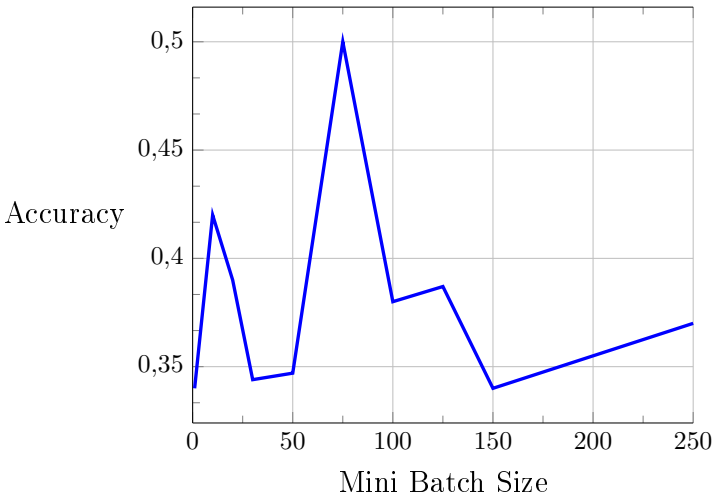


Figure 7.24: Accuracy in function of Mini-Batch size

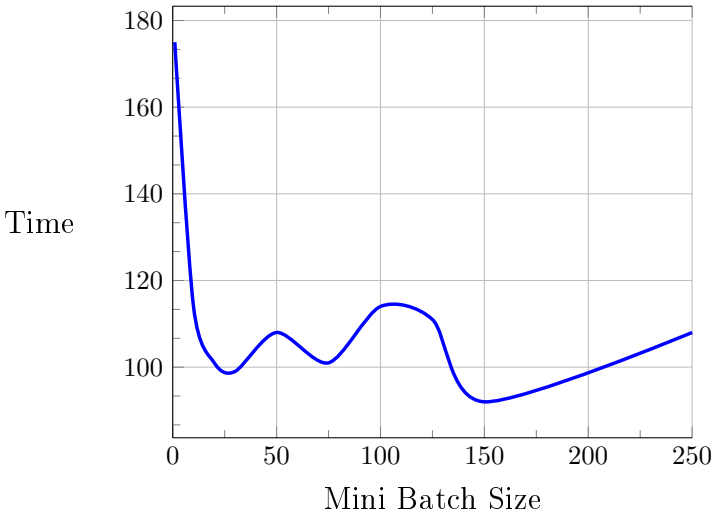


Figure 7.25: Computational time in function of Mini-Batch size

Image Dimension N

In this section the variation of accuracy and computational time in function of the input image resolution is studied. It is important to understand that the native resolution of the images in the database is always 80 x 80 pixels. The value of the parameter N considered in this paragraph is related to the size of the input image after the original image has been resized. It is also important to remember that the Matlab function "resize" allows to downscale and upscale images using bicubic interpolation. It is expected that the accuracy increases slightly while the computational time heavily increases with the upscale of the original images.

Image Dimension	Accuracy	Time [s]	Overfitting
40	0.37	24	high
60	0.42	38	high
80	0.44	64	mid
100	0.33	99	mid
120	0.37	144	mid
140	0.34	199	high

Table 7.6: Parametric study results changing image dimension N

The results show a different behaviour. Indeed, the best resolution is achieved using the native resolution of the database images. Both downscale and upscale decrease accuracy. The computational cost follows the trend expected, increasing heavily with the dimension of the dataset image. For very high value of image dimension, also little instability has been observed.

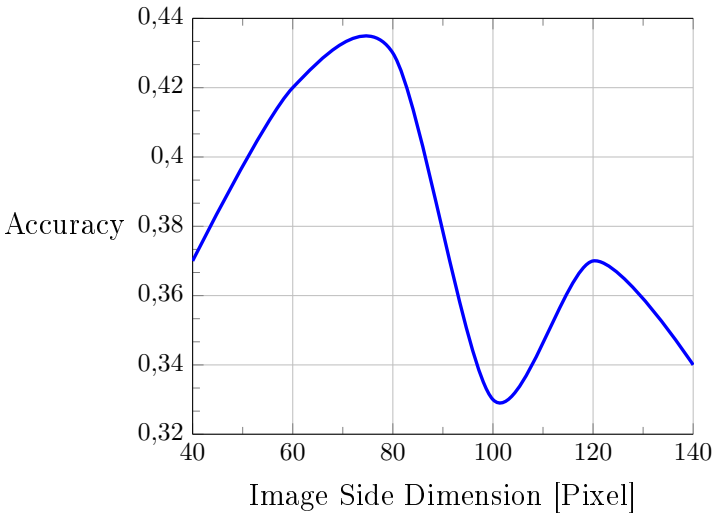


Figure 7.26: Accuracy in function of image dimension

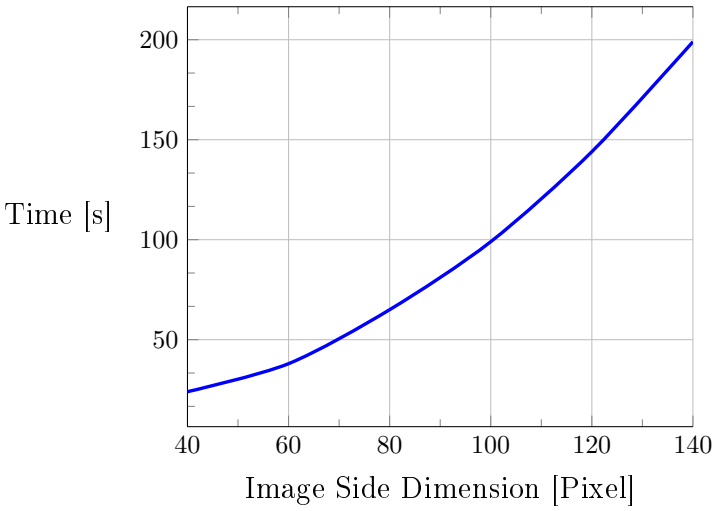


Figure 7.27: Computational time in function of image dimension

Dropout

The dropout number represents the number of artificial neurons of the previous layer "switched off" by the dropout layer. For example, to a value of 0.1 corresponds 10% of artificial neurons switched off, to a value of 0.5 corresponds the 50% of artificial neurons switched off and so on. The presence of the dropout layer reduces the overfitting and may also affects accuracy. In most studies found, the most used value of dropout is 0.5. In this work, since the number of artificial neurons used is small, it was expected the optimal value of dropout to be smaller than 0.5. The computational time was expected to remain constant. The most important result expected is a big difference in accuracy and overfitting of the results, with high accuracy and low overfitting for an optimal value, and low accuracy and high overfitting for other values.

Dropout value	Accuracy	Time [s]	Overfitting
0	0.36	66	high
0.1	0.39	66	high
0.2	0.36	66	mid
0.4	0.37	66	mid
0.5	0.41	66	mid
0.6	0.48	66	mid/low
0.8	0.38	66	mid

Table 7.7: Parametric study results changing dropout value

The results confirm partially what expected. The overfitting is high for low or high dropout values, and mostly for extremely low dropout values (0 or 0.1). Differently from what expected, the optimal value is not smaller but slightly greater than 0.5. Indeed, the best result corresponds to a dropout value of 0.6. To this value corresponds not only the highest value of accuracy, but also shows less overfitting than other cases. As expected, the computational time is not affected by the dropout value.

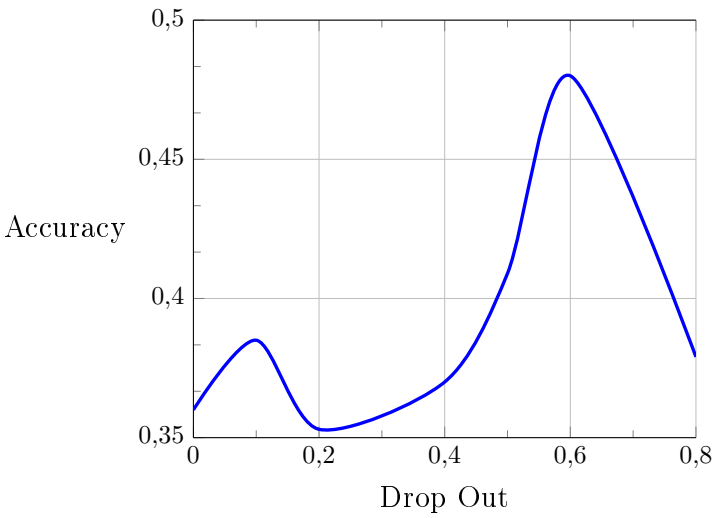


Figure 7.28: Accuracy in function of dropout value

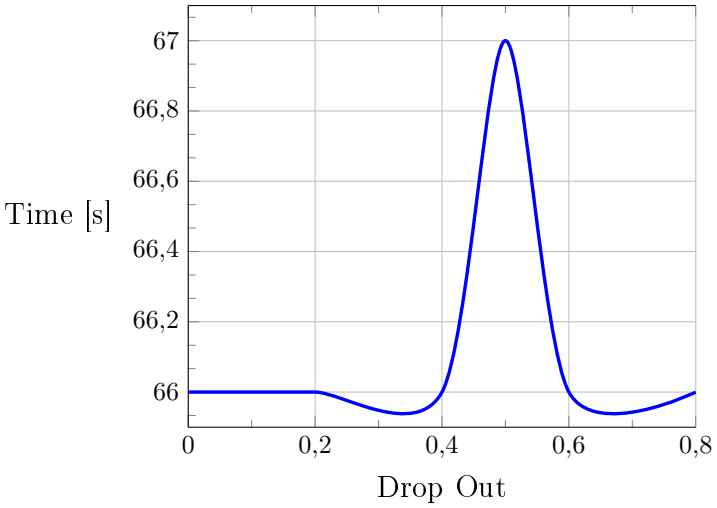


Figure 7.29: Computational time in function of dropout value

Total Number of Epochs

The number of epochs represents how many times the training process evaluates all the input data in the training dataset. Theoretically a higher number of epochs leads to a better solution, with higher accuracy. This number also influences heavily the computational cost of the training,

Number of epochs	Accuracy	Time [s]	Overfitting
1	0.49	22	mid/low
2	0.35	45	high
3	0.38	66	mid
4	0.41	90	mid
5	0.39	112	mid
6	0.43	132	mid
7	0.38	154	mid
8	0.38	175	mid
9	0.45	198	mid/low
10	0.4	221	mid

Table 7.8: Parametric study results changing total number of epochs

At first, it seems that the solution does not follow a well-defined trend, in fact for some number of epoch the accuracy is higher, while for others it is lower. It may be mistakenly thought that the results do not depend on the number of epochs at all. With a more accurate analysis, it is possible to understand the reason of this phenomenon. The first consideration to make is that, since the network is designed to be very light, even very low numbers of epoch (for example 1 epoch) may be sufficient to reach a relatively high accuracy. A second important consideration is to be found in the learning process. The learning rate is constant, this means that at each weight update corresponds a constant step towards the solution. In this case, after a few weight updates, the result is quite close to the optimal one. This makes the next weight update to overshoot the optimum point (it gets away instead of getting closer). What happens then is that the result jumps around the optimal value getting each time closer and further. It is possible to avoid this phenomenon and improve the result adopting a variable learning rate. It consists in using initially a learning rate α of higher value (the considerations made previously on

the stability of the algorithm are still valid), and then decreasing the α value when approaching the solution, to avoid overshooting the optimum point. Because of these results, it has been decided to consider three different values of number of epochs for the following phase. The value chosen are: 1 epoch, 4 epochs and 9 epochs. This values are characterized by greater accuracy than the other values in the table and are a good combination to compare the results generated by a low, medium and high number of epochs.

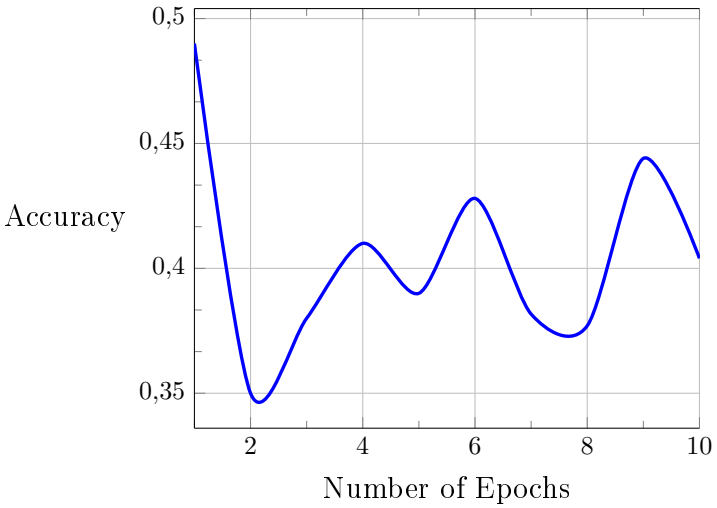


Figure 7.30: Accuracy in function of total number of epochs

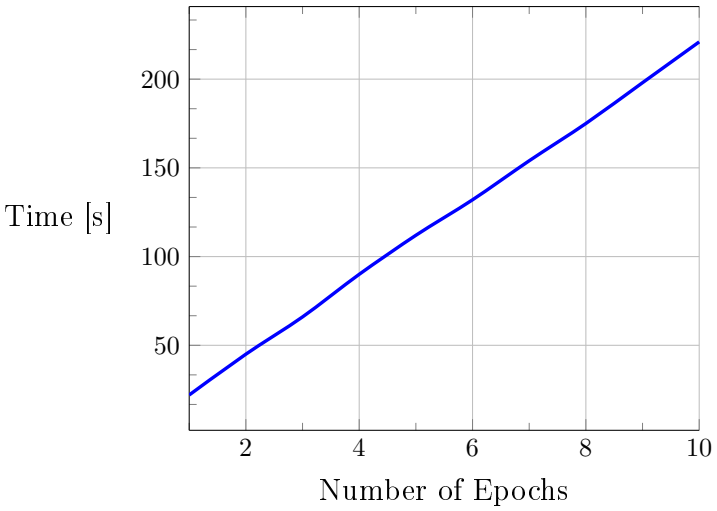


Figure 7.31: Computational time in function of total number of epochs

Training Dataset Dimension

The results generated by the three chosen values of number of epochs are compared in function of the training dataset dimension. For a number of epochs of 1, the results are shown in the following table.

Dataset dimension	Accuracy	Time [s]	Overfitting
1000	0.45	23	mid
2000	0.45	44	low
3000	0.35	67	mid
3500	0.40	79	mid

Table 7.9: Parametric study results changing the training dataset dimension with a total number of epochs of 1

The table shows relatively high accuracy values, with very low training times. However, the overfitting is quite high. Increasing the dataset dimension, overfitting tends to increase slightly. This is caused mainly by some characteristics of the database, such as relatively low image quality and different numbers of data present in each category.

For a number of epochs of 4, the results are shown in the following table.

Dataset dimension	Accuracy	Time [s]	Overfitting
1000	0.49	88	low
2000	0.49	177	mid/low
3000	0.41	266	mid/low
3500	0.42	323	low

Table 7.10: Parametric study results changing the training dataset dimension with a total number of epochs of 4

The table shows relatively high accuracy, with moderate training times. This result corresponds to the optimal solution. The overfitting is lower than the previous case, but it is still present. It would seem that by increasing the number of epochs, the results tend to improve.

For a number of epochs of 9, the results are shown in the following table.

Dataset dimension	Accuracy	Time [s]	Overfitting
1000	0.40	203	mid
2000	0.40	404	high
3000	0.38	605	high
3500	0.36	680	high

Table 7.11: Parametric study results changing the training dataset dimension with a total number of epochs of 9

The table shows lower value of accuracy than the previous cases. The training time is also definitely high and also overfitting is extremely high. The network generated by this configuration is therefore to be discarded.

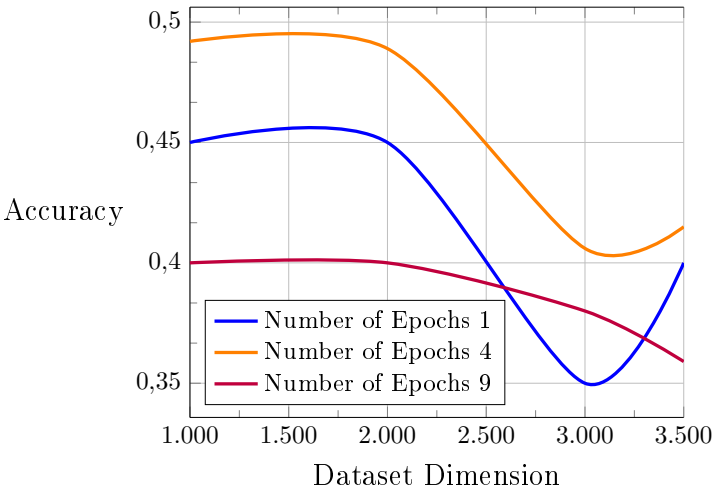


Figure 7.32: Accuracy in function of dataset dimension for different values of total number of epochs

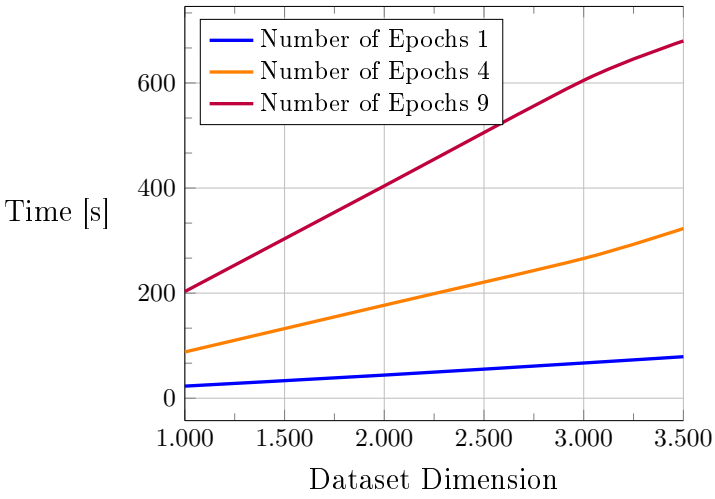


Figure 7.33: Computational time in function of dataset dimension for different values of total number of epochs

7.3.3 Future developments

The developments needed to improve the network are listed below:

- Variable learning rate.

  Adopting a variable learning rate, the accuracy may increase, because the risk of overshooting the optimum point will be reduced or eliminated. Then it will be possible to generate a result closer to the optimal solution. An example of training with variable learning rate is shown in the following image:

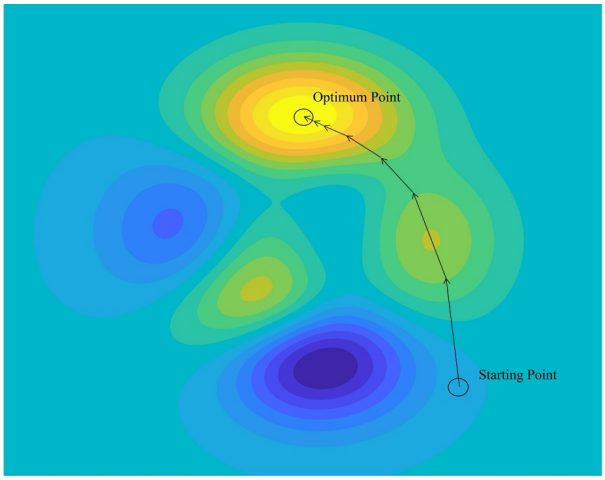


Figure 7.34: Successful trainings with variable learning rate α.

  However, it is not expected that adopting a variable learning rate, could reduce overfitting.

- Database with higher resolution images and/or bigger cells.

  Using a database with images with higher resolution is most likely to generate results with higher accuracy and may also lead to reduce the overfitting. Another way is to divide the original map in bigger cells. The cells now considered have a dimension of more or less 5 x 5 meters on the map. Using cells of dimension 10 x 10 meters may lead to a better accuracy, at the cost of having the original map divided into a smaller number of cells.

- Database with more data.

  Theoretically, the bigger the database, the better the results. The database used is composed of about 4800 images. The number of images in each category is also different. Extending the database with more images may increase the accuracy of the network and reduce overfitting. Another improvement of the database is to make sure that all the categories have an equal number of images.

- Three channels images.

  The last improvement that can be done is also the most important one. The network made is very light and fast but gives results with an accuracy that is not sufficiently high. A certain way to increase accuracy and reduce overfitting, is to consider all three channels of the images (RGB), differently from what made in this work (only one channel has been used). Implementing the use of all three channels certainly will imply making the network more complex and less light. Also, the computational time of the training will be much higher.

Chapter 8

Phase 2: Transfer Learning

The second phase, involving the implementation of transfer learning from an already existing network is discussed in this chapter. The network used has been chosen after a search of the CNNs findable online. Is has been chosen to use AlexNet, because it was considered most suitable network written in Matlab programming language. AlexNet is a CNN (Convolutional Neural Network) available in Neural Network Toolbox in Matlab. It was developed for the Image Net Large-Scale Visual Recognition 2010 Challenge to classify 1.2 million images into 1000 unique categories. The purpose of the original network is very similar to what this thesis work is trying to achieve. Adopting the transfer learning from a CNN developed to classify images could be one of the best ways to generate a final network characterized by high accuracy and low overfitting. The architecture of the network is similar to the architecture adopted for the CNN described in chapter 7. AlexNet is composed of an initial input layer, a Feature Extraction Network, a Classifier Network and an output layer. Differently from the network described in the previous chapter, it uses all three channels RGB of the input images, then a higher value of accuracy and really low overfitting is expected. The database used for the training is the one already described, then the consideration made about it are the same. AlexNet is composed of 25 layers listed in the following page:

Layer Number	Layer Name	Layer Type
1	Data	Input Data
2	Conv1	Convolutional Layer
3	Relu1	ReLU Activation Function
4	Norm1	Cross Channel Normalization
5	Pool1	Max Pooling Function
6	Conv2	Convolutional Layer
7	Relu2	ReLU Activation Function
8	Norm2	Cross Channel Normalization
9	Pool2	Max Pooling Function
10	Conv3	Convolutional Layer
11	Relu3	ReLU Activation Function
12	Conv4	Convolutional Layer
13	Relu4	ReLU Activation Function
14	Conv5	Convolutional Layer
15	Relu5	ReLU Activation Function
16	Pool5	Max Pooling Function

Table 8.1: AlexNet: Feature Extraction Network

Layer Number	Layer Name	Layer Type
17	Fc6	Fully Connected Layer
18	Relu6	ReLU Activation Function
19	Drop6	Dropout Layer
20	Fc7	Fully Connected Layer
21	Relu7	ReLU Activation Function
22	Drop7	Dropout Layern
23	Fc8	Fully Connected Layer
24	Prob	Softmax Activation Function
25	Output	Classifier Output

Table 8.2: AlexNet: Classifier Network

It can be seen that this network is a lot more complex than the one previously described. In particular, there is a huge difference between the FEN of the two networks. In this case multiple convolutional layers have been used. Moreover, are also used two normalization layers. It is important to remember that a normalization layer needs a validation process in order to properly work, making the study of the best parameters of the network more complex to perform.

Employing a larger number of layers and weights, it is expected that the training computational time of this network is also much larger. In fact, while the training time of other network was of the order of magnitude of minutes, for this network the computational time is of order of hours.

8.0.1 Layers description

In this paragraph are briefly described all the layers of the network.

Layer 1 "Data": this is the input layer, it contains one image from the training dataset. The size of the image is 227*227*3, that is an image of 227*227 pixels and 3 colours channels (RGB)

Layer 2 "Conv1": this is the first convolution layer, it is a learning layer (a layer that contains weights and bias that are updated during the learning process). It uses 96 filters of size 11*11 pixels and 3 channels. The total number of learnable parameters in this layer is 11*11*3*96 (weights) + 96 (bias) = 34944 parameters.

Layer 3 "Relu1": in this layer the ReLU activation function is used.

Layer 4 "Norm1": this is a cross-channel normalization layer with hyper-parameters: $n = 5$, $\alpha = 1 * 10^-4$, $\beta = 0.75$, $k = 1$.

Layer 5 "Pool1": this is the first pooling layer, the pool size applied is 3*3 pixels.

Layer 6 "Conv2": this is the second convolution layer, it is a learning layer. It uses 256 filters of size 5*5 pixels and 48 channels. The total number of learnable parameters in this layer is 5*5*48*256 (weights) + 256 (bias) = 307456 parameters.

Layer 7 "Relu2": in this layer the ReLU activation function is used.

Layer 8 "Norm2": this is a cross-channel normalization layer with hyper-parameters: $n = 5$, $\alpha = 1 * 10^-4$, $\beta = 0.75$, $k = 1$.

Layer 9 "Pool2": this is the second pooling layer, the pool size applied is 3*3 pixels.

Layer 10 "Conv3": this is the third convolution layer, it is a learning layer. It uses 384 filters of size 3*3 pixels and 256 channels. The total number of learnable parameters in this layer is 3*3*256*384 (weights) + 384 (bias) = 885120 parameters.

Layer 11 "Relu3": in this layer the ReLU activation function is used.

Layer 12 "Conv4": this is the fourth convolution layer, it is a learning layer. It uses 384 filters of size 3*3 pixels and 192 channels. The total number of learnable parameters in this layer is 3*3*192*384 (weights) + 384 (bias) = 663936 parameters.

Layer 13 "Relu4": in this layer the ReLU activation function is used.

Layer 14 "Conv5": this is the fifth convolution layer, it is a learning layer. It uses 256 filters of size 3*3 and 192 channels. The total number of learnable parameters in this layer is 3*3*192*256 (weights) + 256 (bias) = 442624 parameters.

Layer 15 "Relu5": in this layer the ReLU activation function is used.

Layer 16 "Pool5": this is a pooling layer; the pool size applied is 3*3 pixels.

Layer 17 "Fc6": this is the first fully connected layer, it is a learning layer. It is composed of 4096 nodes (then gives 4096 outputs), and receives 9216 inputs. The total number of learnable parameters is 4096*9216 (weights) + 4096 (bias) = 37752832 parameters.

Layer 18 "Relu6": in this layer the ReLU activation function is used.

Layer 19 "Drop6": this is a dropout layer with dropout probability of 0.5 (50%).

Layer 20 "Fc7": this is the second fully connected layer, it is a learning layer. Is is composed of 4096 nodes, and receives 4096 inputs. The total number of learnable parameters is 4096*4096 (weights) + 4096 (bias) = 16781312 parameters.

Layer 21 "Relu7": in this layer the ReLU activation function is used.

Layer 22 "Drop7": this is a dropout layer with dropout probability of 0.5 (50%).

Layer 23 "Fc8": this is the third fully connected layer, it is a learning layer. It is composed of 1000 nodes and receives 4096 inputs. The total number of learnable parameters is 1000*4096 (weights) + 1000 (bias) = 4097000 parameters.

This layer has been modified to adapt the net for our case. The number of nodes has been set equal to the number of categories needed for classification (from sheltering factor 0, to sheltering factor 7; in total 8 categories). The new layer created is composed of 11 nodes and still receives 4096 inputs. The total number of learnable parameters is 11*4096 (weights) + 11 (bias) = 45067 parameters.

Layer 24 "Prob": in this layer the Softmax activation function is used.

Layer 25 "Output": this is the classification layer, it contains 1000 nodes. Each node represents a class in which the input data belongs. The output given by a node is the probability of the data to belong in the corresponding class or category.

This layer has been modified to adapt the net to our case. The new layer created is composed of 11 nodes representing the classes from sheltering factor zero to sheltering factor ten.

Also in this phase, a parametric study has been made in order to find the best values that lead to higher accuracy.

8.1 Results

The results obtained with AlexNet are described in this chapter. For this network, not all parameters have been studied, due to the large amount of time needed for a single training. In effect, being an already existing and already trained network, not all parameters need to be studied. There is an important consideration to make regarding accuracy. Differently from the accuracy considered for the network described on the first phase, in this phase the exact accuracy is considered. The classification is considered right if the category obtained as output corresponds exactly to the correct category. In the other network, instead, the output was considered accurate if the category predictes belongs to the correct category or to one of the two neighboring ones. The accuracy is evaluated as:

$$\text{Accuracy} = \frac{\text{Number of right predictions}}{\text{Total number of predictions}}$$

8.2 Training Options Parametric Study

The total options of a neural network training are numerous. A parametric study has been done to know which ones influences most the final results. The two indicators considered for this study are the accuracy of the network after the training and the computational cost in terms of time. In most of the cases, the overfitting of the trained network was very low. Three training options has been considered:

- Total number of epochs
- Learning rate α
- Mini-Batch size
- Dataset dimension

8.2.1 Total Number of Epochs

The number of epochs greatly affects the accuracy of the network. In particular, for a very low number of epochs, the final value of accuracy is lower, as the number of epochs increases, the accuracy increases until it reaches a maximum value. Once this value is reached, increasing the number of epochs, the accuracy value is considerable constant.

Number of epochs	Accuracy	Time [h]
10	0.26	0.77
20	0.5	1.53
50	0.51	3.87
100	0.52	7.77

Table 8.3: Parametric study results changing total number of epochs

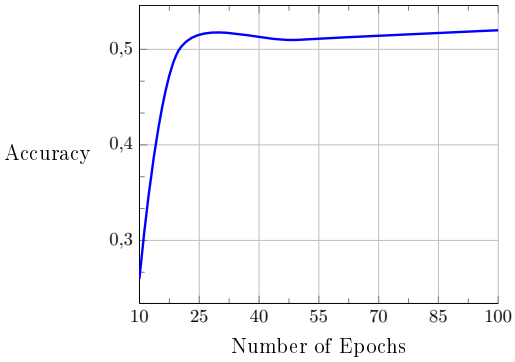


Figure 8.1: Accuracy in function of number of epochs

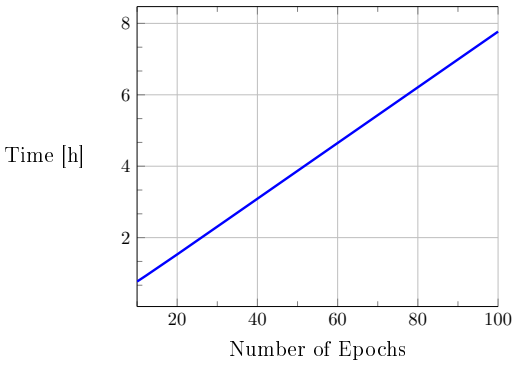


Figure 8.2: Computational time in function of number of epochs

The trend obtained in the results reflects what expected. The accuracy tends to increase with a higher number of epochs, tending towards a limit value. Differently from the network described in phase 1, the number of epochs necessary to have a higher value of accuracy is greater.

The number of epochs also greatly affects the training time. The correlation between time and number of epochs can be considered linear. A double value of number of epochs, corresponds to twice the time required for training. The training time trend is shown in the following picture.

8.2.2 Learning Rate

As already described, the learning rate of a neural network represents the rate at which weights are updated. A high learning rate α means that the weight update Δw is higher. By decreasing the learning rate, the algorithm becomes more stable. Theoretically, for large value of α, the algorithm should be unstable, from a certain value to lower, the algorithm should become completely stable.

Learning Rate	Accuracy	Time [h]
1	Unstable	-
0.5	0.26	0.75
0.1	0.26	0.75
0.05	0.26	0.75
0.01	0.26	0.75
0.001	0.50	0.75
0.0001	0.44	0.75

Table 8.4: Parametric study results changing learning rate α

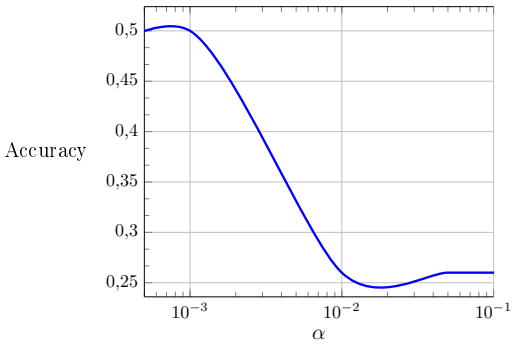


Figure 8.3: Accuracy in function of learning rate

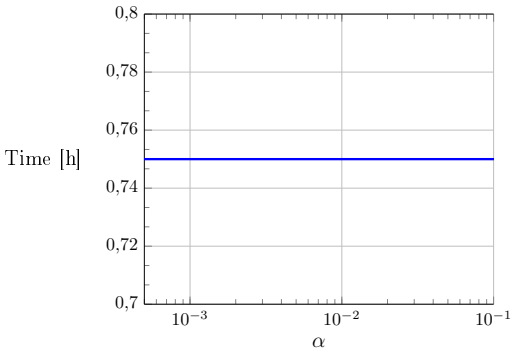


Figure 8.4: Computational time in function of learning rate

As expected, it has been found that a extremely large value of learning rate leads the training algorithm to instability, failing to find a solution. Decreasing the learning rate value, the training converge to a solution. Even in case of convergence reached, the training may be not successful because the trained network may be characterized by a small value of accuracy or affected by

high overfitting. This condition has been observed for learning rate values between 0.5 and 0.01. For lower values this problem has not occurred. The best result found corresponds to a learning rate of 0.001 with high accuracy. Also as expected, a smaller learning rate, decreases the speed of the algorithm. Indeed for a value lower than 0.001, the accuracy decreases, this happens not because the algorithm fails to reach the optimal solution but because more number of epochs would be needed to reach the optimum point (the number of epochs considered in this study is constant). No considerations are needed about the computational time, indeed, depending mainly from the number of epochs and not from the learning rate, all the trainings needed the same amount of time.

8.2.3 Mini-Batch size

The minibatch size corresponds to the number of images evaluated before a weight update. For example, a minibatch size of ten means that during the training, ten images will be considered, ten errors will be evaluated and ten Δw values obtained, and then the weight will be updated with the mean value of the Δw found. With a minibatch size value of twenty, there will be twenty images, twenty errors and twenty Δw considered before a weight update. Theoretically it was expected that by decreasing the minibatch size the algorithm was less stable (or the possibility of instability higher) but, if able to converge, the accuracy was higher. The computational cost in terms of time also was expected to be slightly higher. Vice versa, it was expected that a higher value of minibatch size would lead to stabilization of the algorithm, lower accuracy and lower training time.

Mini-Batch size	Accuracy	Time [h]
7	0.53	0.86
15	0.55	0.5
30	0.54	0.35
60	0.53	0.26

Table 8.5: Parametric study results changing Mini-Batch size

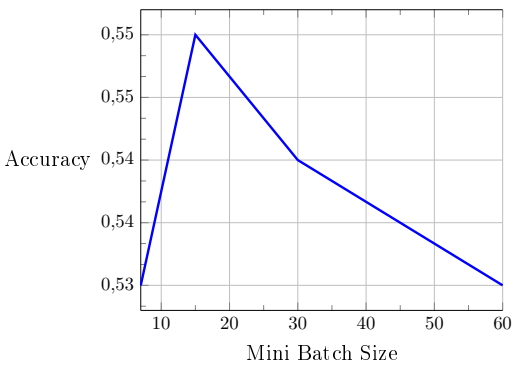


Figure 8.5: Accuracy in function of Mini-Batch size

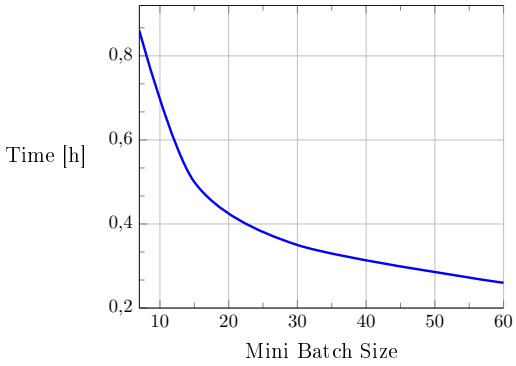


Figure 8.6: Computational time in function of Mini-Batch size

The results show a trend different from what expected. Considering the stability, the algorithm has always been stable, regardless of having assumed small or large value of minibatch size. Considering accuracy, it resulted also independent from the selected value, oscillating around a constant value. The training time was instead highly dependent on the minibatch size value. Increasing it reduces a lot the training time. In conclusion a high value of minibatch size is certainly more convenient.

8.2.4 Dataset dimension

Is this chapter the training dataset dimension is expressed as the number of maps considered for training. Theoretically, a small number of maps, should result in a smaller accuracy. The accuracy should increase as the number of maps increases (and therefore the dataset dimension increases). The computational time is expected to be extremely dependent on the database dimension, requiring small computational time for a smaller database, and much larger computational time for a larger database.

Number of maps	Accuracy	Time [h]
3	0.54	0.35
9	0.60	1.33
19	0.55	2.46
23	0.51	2.65
30	0.49	3.5

Table 8.6: Parametric study results changing training dataset dimension

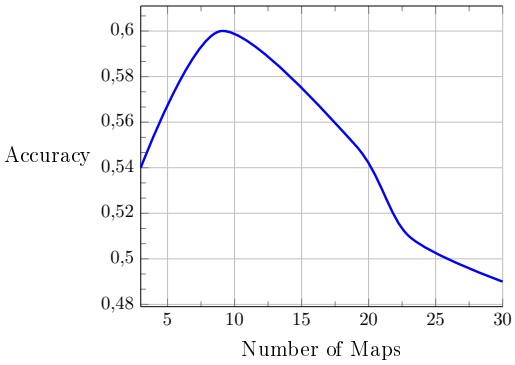


Figure 8.7: Accuracy in function of number of maps

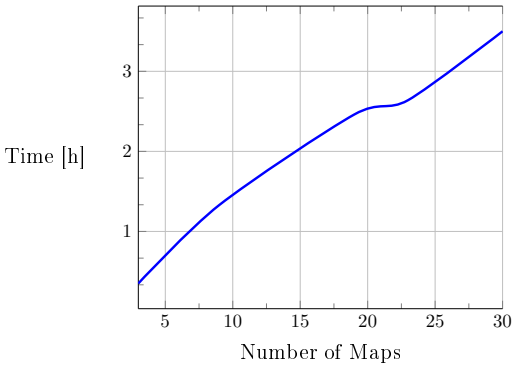


Figure 8.8: Computational time in function of number of maps

The results in this case are completely different from what expected. There are two consideration that can be made. The first is the quality of the network: the CNN used is excellent for solving image classification problems. Indeed, even considering a small training dataset, the accuracy of the trained network is high. The second consideration regards the database. As already explained, the quality of the database used is rather low. A weak point in the database

is the low resolution of the images. Having a database composed by higher resolution and better quality images, it will certainly improve the performance of the trained network. Another weak point, but less important, is the composition of the images. The amount of data present in each category is not uniform (some categories have more data than the others). This means that the trained network will tend to better predict imaged of categories for which there is more data available.

8.3 Considerations on accuracy

As described before, for this network has been considered the exact accuracy. This is because, being the confusion matrix of most results composed of many elements close to the main diagonal, it could be possible to obtain an approximate accuracy similar for all the results given by different training parameters. Considering the approximate accuracy it would not have been possible to recognize the configurations that give better results. Taking into account the results obtained with the last configuration evaluated, with all the database and the final values, the approximate accuracy obtained would be 67.5%. This can be an acceptable accuracy value for a neural network.

8.4 Future Developments

For this network, in order to improve the results, the only development that can be done concern the database. All the consideration already made remain valid, especially those on the use of higher resolution images. The training time of the network is very large. For each training several hours were required. This can be considered acceptable only if the training is to be performed only once, with a small database like the one used in this work. In the case of a larger database, or if the training has to be repeated several times, the training time may become excessive. One method to resolve this problem is using parallel computing with high-performance GPUs. In this way, the computation time could be reduced by one or two orders of magnitude, making possible to carry out heavier trainings and obtaining better results achieving higher accuracy.

Bibliography

[1] Carlo Casarosa. *Meccanica del volo*. Pisa University Press, 2013.

[2] European Aviation Safety Agency EASA. "Concept of Operations for Drones, A risk based approach to regulation of unmanned aircraft". In: (2015). URL: https://www.easa.europa.eu/.

[3] Ente Nazionale Aviazione Civile ENAC. "Aeromobili a pilotaggio remoto con caratteristiche di inoffensività". In: *Linee Guida LG 2016/003 NAV Ed. n. 1 del 1° giugno 2016* (2016). URL: https://www.enac.gov.it.

[4] Gianluca Ristorto Giorgio Guglieri. "Safety Assessment for Light Remotely Piloted Aircraft Systems". In: *International conference on Air Transport INAIR 2016* (2016).

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

[6] Giorgio Guglieri. *Introduzione alla meccanica del volo*. Celid, 2005.

[7] Geoffrey E. Hinton Alex rizhevsky Ilya Sutskever. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (2012).

[8] Rudolf Scitovski Josip Cumin Branko Grielj. "Numeral solving of ballistic flight equations for big bore air rifle". In: *Tehnički vjesnik – Technical Gazette 16, 1(2009), ISSN 1330-3651* (2008). URL: http://www.tehnicki-vjesnik.com.

[9] Fulvia Quagliotti Luca De Filippis Giorgio Guglieri. "Path Planning Strategies for UAVS in 3D Environments". In: *Journal of Intelligent and Robotic Systems, January 2012, Volume 65* (2012).

[10] Hermant Sharma et al. "Design of a High Altitude Fixed Wing Mini UAV – Aerodynamic Challenges". In: *International conference on Intelligent Unmanned Systems, ICIUS-2013-129* (2013).

[11] Kimon P. Valavanis and George J. Vachtsevanos. *Handbook of Unmanned Aerial Vehicles*. Springer Publishing Company, Incorporated, 2014. ISBN: 9048197066, 9789048197064.