POLITECNICO DI TORINO

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Autonomous Burnt Area Detection Model

A Machine Learning Approach

Relatori:
Prof. Paolo Garza
Dott. Alessandro Farasin

Candidato:
Giovanni Nini

Anno accademico 2017-2018

Ai miei genitori, che hanno sacrificato tanto per darmi tutto.

Acknowledgements

Un grazie infinito alle due persone che hanno reso possibile tutto questo: i miei genitori. Per l'educazione ed i valori che mi hanno tramandato. Un grazie ad Elisa ed Eleonora, con cui ho condiviso un percorso di vita. A Simone, che fa parte della famiglia. Ai miei amici e colleghi, con cui ho sofferto e gioito. Ringrazio, in special modo, Alberto, con il quale ho condiviso buona parte di questi anni. Giovanni, esempio di amicizia vera e indissolubile. Vittorio, fratello vero e autentico. Giulia, la metá che mi migliora e completa.

Contents

Chapter 1

Introduction

Global climate emergencies have been representing a critical topic during the last decades. Fires, floodings and other events have been hitting the entire world with very high frequency, determing very difficult situations and, in worst cases, catastrophes. Since Earth is experiencing higher temperatures with respect to the past, fires represent a huge issue to deal with. Europe and, especially, southern Europe, thus, the Mediterranean Area, have been going through this kind of emergencies. Here it is shown the incidence of forest fires in Europe, in a temporal range up to 2010:

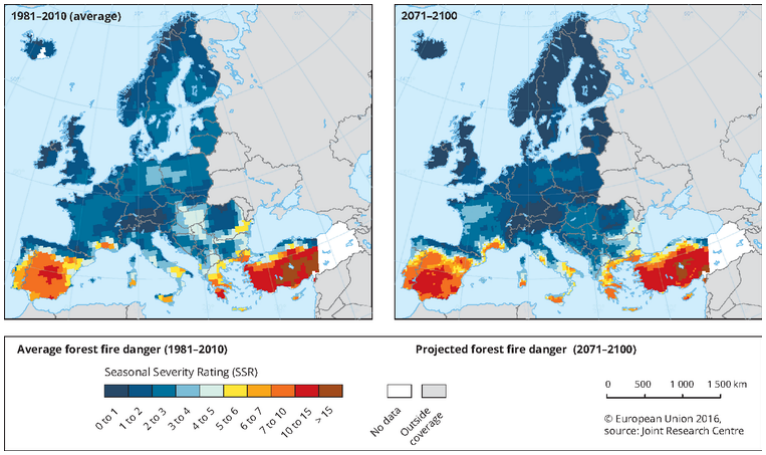


Figure 1.1: - Forest fires- Europe and Mediterranean area [32]

In 1.1 it is showed the forest fires incidence in the european area by means of the **SSR**: Seasonal Severity index, which is a weighted indicator for defining the average fire severity of a specific area. As it is clearly showed, the Mediterranean area and, especially, Portugal, Spain and Turkey, have been hit by a severe fire season. In this

particular scenario, it becomes crucial defining some strategies, exploiting all the possible technologies developed in the last years, in order to prevent these events from happening or, at least, to build tools and knowledge to reduce the damages produced by them. Nowadays, there are powerful instruments that must be put together so that it will be possible to handle emergency situations in a better way. Data are one of those instruments. Sensors, data stations and IoT technologies have made available a huge quantity of information in any format and of every nature. This important amount of structured and unstructured data, if handled wisely, hides knowledge. During the last years, satellites have been collecting enormous quantities of data for different purposes, starting from the American missions called Landsat, to the new European programme called Copernicus. The latter is introducing hour by hour, day by day, fresh information about lands, seas, air quality, climate analysis and so on. This is possible thanks to the missions launched a couple of years ago, which will be functioning for the next years, collecting more and more information about our Planet. Being able to collect this information and to properly work on it, thus, becomes critical. The goal of this project was tracing a path among the variety of services providing the information mentioned above; and defining a way through all the possible formats available, to put in order the provided information and to properly manipulate those formats so that it was possible to define a model for detecting burnt areas, thus area hit by fires. The state of art in this field is based on a manual annotation for each single event, slowing down significantly the mapping process and, sometimes, the capability of quick intervention during natural catastrophes. Even though there were a lot of steps to achieve interesting results, it is possible to define a few major steps, which will be illustrated in more details in the next chapters.

Chapter 2

State of the art and General Overview

This chapter describes some works that have been done in a field of application similar to the one of this project. It is also shown a high level description of the context of the thesis, in terms of the services used and currently available and, additionally, the tools exploited to achieve results.

2.1 Related Works

Burnt area detection systems exploiting machine learning techniques have not been widely covered in past researches. As things stands at the moment, very basic fire annotations are refined manually in order to get polygons surronding burnt areas in a more precise way. Some solutions have been developed during the last years, such as the one presented in [33]. Its main goal is mapping burnt areas over tropical forests of South America and South-East Asia. There are existing satellite-based techniques that map active hotspot fires by analysing the temperature of the given hotspot and compare it with the temperature of the surrounding areas. If this difference is greater than a given threshold, the analysed area is considered as an on-going fire.

2.2 Collecting burnt areas information

This section covers the main methods and data sources that have been exploited in order to retrieve all the information related to specific geographic areas affected by fires. There are several softwares and services that provide graphic or application programming interfaces that let the user properly interact with this good amount of data.

2.2.1 Data origin - Copernicus and Sentinel Satellites

The entire work is based on the data collected by Sentinel satellites [4] . These satellites have been specifically developed to support the Copernicus programme. In detail, Copernicus is a european programme, whose goal is understanding the environment we live in, in order to manage its resources in a sustainable manner and to provide precious information for emergencies. Its services provide near real-time data at a global level, thanks to the already mentioned satellites and in-situ ground stations. As things stand at the moment, there are six different Sentinel missions [5]. Some of them have been launched and are performing different kind of analyses, for different purposes; while some others will be launched soon. The Sentinel-1 family is composed of two polar-orbiting satellites, whose acquisitions are based on a C-band synthetic aperture radar imaging, in order to retrieve images regardless of weather conditions. The Sentinel-2 family, as the first one, relies on two polar-orbiting satellites, but its goal is monitoring land surface conditions and changes to vegetation. Its satellites have a single payload: the multi-spectral instrument, which provides thirteen different spectral bands for each single acquisition. The Sentinel-3 mission goal is measuring sea surface topography, sea and land surface temperatures, land surface and ocean color, in order to support ocean forecasting systems and monitoring changes in both climate and environment. It has several payloads, in order to measure all those different parameters. Sentinel-4 intent is the continuous monitoring of the atmospheric composition. Its main objective is analysing air quality exploiting an ultraviolet visible near-infrared spectrometer and an infrared sounder. Sentinel-5 mission will be launched in 2021 timeframe. It will also deal with atmospheric composition, by accurately measuring its main constituents such as ozone, dioxide, methane, nitrogen and so on. Recently, there has been the launch of the Sentinel-5 precursor mission, that's why it is known as Sentinel-5 P. It is providing new data and measures until Sentinel-5 mission will be operative. The last mission, Sentinel-6, will be launched in 2020. It will provide high accuracy altimetry for measuring global surface height. It is a cooperative mission, devoloped in partnership between Europe and U.S. .

2.2.2 EFFIS

EFFIS [7] stands for "European Forest Fire Information System". Its main goal is supporting all the services in charge of protecting forests against fires in european countries. It also provides the European Commission services and the European Parliament with up-to-date and reliable information regarding wildland fires in Europe. It has been supported, for more than a decade, by a network of experts from

forty european, middle east and north-african countries. From 2015 it is a part of the Copernicus programme and, specifically, it is one of the components of the Emergency Management Services. EFFIS provides different services in order to be up-to-date with active fire events. It also makes available historical information about fires that took place in the past. It is possible to check current or past fires by exploiting interactive maps. This important amount of data is accessible and available directly from website, through interactive services or by querying the system through several application programming interfaces.

Active Fires

Active fire detection is provided by the NASA FIRMS (Fire Information for Resource Management System). It is based on an algorithm which looks for the so-called thermal anomalies. In detail, the system compares the temperature of a potential fire with the temperature of the surrounding areas. If the subtraction leads to a value greater than a given threshold, the under-control area is labeled as active fire. MODIS is a sensor, on board of TERRA and ACQUA satellites, which labels burnt areas exploiting a similar mechanism as well as VIIRS. But, while the former has a spatial resolution of 1 kilometer, the latter has an even better spatial resolution of 375 meters, which allows detecting smaller fires and can help delineating perimeters of on-going large fires. As mentioned before, active fires are displayed on an interactive map and their information is updated six times per day and made available in EFFIS within two-three hours of the acquisition of the MODIS/VIIRS images. In order to minimize false alarms and display only wildfires, an additional filtering is applied to all the hotspots detected by FIRMS. This algorithm is based on a series of attributes like the extent of land cover categories surrounding the affected area, the distance to urban areas and artifical surfaces and the confidence level of the hotspot. In addition to these sources, further information is collected through the news coming from appropriate RSS feeds published by various forest fires sites.

Rapid Damage Assessment

Rapid Damage Assessment, also known as RDA, is a module, part of the EFFIS system, initially implemented in 2003, whose goal is mapping burnt areas analysing MODIS daily images at 250 meters spatial resolution. These images are usually processed a few hours after the acquisition. The RDA provides the daily update of the perimeters of burnt areas for fires of about 30 ha or larger, twice a day. Since 2016, the module also maps images acquired from the VIIRS sensor. Burnt areas whose extension is smaller than MODIS maximum spatial resolution will not

be mapped. It is worth noticing the metodologies used in order to map these areas. Modis imagery is mapped exploiting the identification of the active large fires; the expansion of the burnt areas is computed, starting from these coordinates, on the basis of region-growing algorithms, but the final refinement of the perimeter is achieved through manual analysis of the images. VIIRS images are mapped exploiting algorithms that derive polygons starting from the active fires detected by the sensors.

2.2.3 Utilities and Softwares

QGIS and Shapefile

Dealing with geographic information, means exploiting the right instruments and softwares. A very important data format, specifically meant to store geospatial information, is shapefile. Specifically, a shapefile is a vector data format widely used in geographic information system softwares. It is extremely common, whenever working with geometric information, especially related to geographic areas, to use a shapefile format to collect and represent it correctly. It is composed of several different files, some mandatory and some optional. The entry point is represented by the .shp file, which automatically points to all the other files that make up the whole shapefile. The .shx mandatory file contains all the geometry features indexes, which are exploited in order to seek quickly polygon information throughout the entire file; while the .dbf file contains a database-like format filled with the attributes related to each single shape stored in the shapefile. There are many applications that can be used in order to work with this specific data format, the one used in this work is QGIS [20], because of its easy-to-use and versatile nature, made possible by its straight-forward interface and the additional plugins provided by its community. This software allows to manipulate shapefiles and retrieve all the information needed for every purpose. For instance, it will be possible to list and show all the areas affected by a fire event. These areas may have additional attributes, such as the longitude-latitude coordinates, the affected country, the soil composition of the hotspot and so on.

PostgreSQL and PostGIS

PostgreSQL is an open source object-relational database management system. It is a popular solution, whenever dealing with databases and information persistency. There are different softwares that may be used whenever working with it. For instance, pgAdmin, which is an open source administration and development platform for PostgreSQL. Postgres functions can be significantly extended by using additional

softwares such as PostGIS [15]. PostGIS is a spatial database extender, so it is a ready-to-use solution for working with spatial information, since it adds support for geographic objects allowing location queries to be run in SQL.

Blob and Azure Blob storage

Blob [1] stands for binary large object. So it may refer to images, audios, application programs and more. Azure Blob storage is Microsoft's object storage solution for the cloud. It represents a very flexible solution, whenever there's the need to store a good amount of files. This flexibility comes from the different possibilities provided by this service: in terms of accessibility, since the storage can be accessed anywhere in the world using http or https protocols and in different manners, exploiting the azure storage rest api, azure powershell, azure cli or azure storage library, such the one implemented in python; in terms of software structure, since the entire files organization relies on different blob types, which can be used for specific operations, such as storing images, appending logging content and so on. All this information is available through containers, which are accessible through the basic storage account.

2.3 Image Processing

This section is about image manipulation through softwares, exploiting acquired knowledge concerning images representation and relationship between different color models. It is fundamental to understand how two dimensional information is put together, in order to be able to get the best features out of it.

2.3.1 Color models

A color model is an abstract mathematical model describing the way each color information can be represented as a tuples of numbers, typically as three or four values. For instance, RGB represents a color model made of three elements: the red, green and blue one. Though RGB represents a valid and widely used color model, also proved by the good amount of color spaces based on it, some other models have been developed during the past years, in order to provide a better flexibility and to extend the limits provided by the RGB representation. Among these additional representations, there are the HSV, HSL and the CMYK color models.

RGB

As mentioned above, RGB stands for red, green and blue. Specifically, it is an additive color model where each single component, thus the red, green and blue one, is added together in different ways in order to reproduce an array of colors. So, the three light beams are added together, wavelength for wavelength, to make the final color's spectrum. Zero intensity for each component gives the darkest color, while full intensity gives the white one. If the intensities of each single component are different, the final result is a colorized hue, while, if the intensities are the same final outcome is a particular shade of gray, depending on the starting intensity. When one of the components has the strongest intensity, the final color is one close to the primary strongest component; when two components have the same strongest intensity, the color is a hue of a secondary color. The great limit represented by this model is that it does not separate the luma, which is the image intensity from chroma, thus the color information. It is worth noticing that the rgb model and, in general, every color model, does not define the concept of color in an absolute sense, since it does not express the chromaticity of every component. In order to get the perceived color, a mapping function between the color model and the color space must be provided.

HSV

HSV stands for Hue, Saturation, Value. It is a color model, which can be easily infered from the RGB model, using a specific mathematical formula. Its great difference is represented by the splitting of the chroma component and the luma one, which allows to overcome the great limit represented by the RGB model. It is more intuitive and represents how people experience color, since each shade can be obtained working specifically on the hue, saturation and value elements separately. It can be represented as a cylinder where the hue value is placed on an imaginary surface, which is representing all its different values, moving from 0°to 359°. The saturation component is represented as the distance from the center of the cylinder to its outer boundary. Given a fixed hue , its saturation value can be modified by moving along the radius. The value, which represents the brightness, is represented as the height of the cylinder; its modulus increases by moving from the bottom to the top of the cylinder.

Hue is the pure representation of colors, excluding all the different variances determined by saturation and brightness. Saturation represents the quantity of gray contained in the color, while brightness or value is the perception of light intensity.

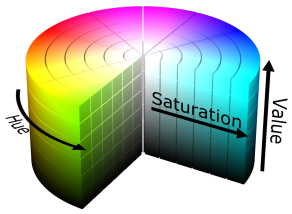


Figure 2.1: HSV cylinder model

2.3.2 Utilities and Softwares

When it comes to image processing, there is a great amount of softwares and tools that can be used in order to process and manipulate images.

OpenCV

OpenCV [13] stands for Open Source Computer Vision Library. It is an open source machine learning and computer vision software library. It was built in order to provide a common infrastructure for computer vision applications. As things currently stand, the current library has more than 2500 optimized algorithms, which include a set of both classic and state-of-the-art computer vision and machine learning algorithms. All these functionalities can be used for different purposes, such as face detection and recognition, object identification, human action classification, camera movement tracking, object tracking, object 3D model extraction, finding similar images from an image database, red eyes removing and so on. Along with these advanced functionalities, the basic ones provide image manipulation such as loading, saving images, color model transformations such as RGB, HSV and grayscale. OpenCV is widely supported and relies on C++, Python, Java and MATLAB interfaces.

Pillow

Pillow [14] is a fork of the PIL project. It stands for Python imaging library and, as it can be guessed from the name, it deals with images and image processing. It contains basic image processing functionalities, such as point operations, filtering with set of built-in convolution kernels and color space conversions. It also supports image resizing, rotation and arbitrary affine transforms.

2.4 Autonomous Burnt Area Detection

This section covers the basic concept of machine learning, thus, its meaning, goals and the algorithms currently used. There is a special mention to the neural networks and a particular version called convolutional neural networks. The final part covers the current frameworks and softwares widely used in order to work with this particular and very popular topic.

2.4.1 Machine Learning

Machine learning is a special branch of artificial intelligence. It is based on the assumption that computers can learn from data, examples, in the same way humans can learn everyday from experience. Nowadays technology is deeply involved in machine learning, because of its proven effectiveness and the exciting and huge improvements made in the last few years in this field. There are two main reasons why machine learning has seen a huge acceleration during these years: the great improvement of hardware performances, which allows working, training and testing models on the basis of an unprecedented speed and the high data availability, which allows using a lot of training samples, creating more and more refined models, capable of getting precise results. There are different techniques and algorithms that can be exploited in order to achieve any result. But, before going deeper in this particular section, it is important to define three differente categories of training, thus, three ways to make a model learn from data, as it is well explained in [34]. The first one, very commonly used, is supervised learning. It is based on the assumption that each single data used for training is labeled, thus, in other words, it is associated with the so called ground-truth output. This is a very important property, since it allows understanding, at every training step, how well the model has performed in terms of the given data prediction and update subsequently its parameters, in a way that tries to minimize a given loss function, which is the mathematical function that tries to define the relationship between the predicted value and the ground truth outcome. There are many algorithms exploited in this field, such as decision trees, support vector machines, Bayes classifier and so on. Reinforcement learning is based on the developing of a system, called agent, which is able to improve its performances by means of interactions with the enviroment. The environment produces a reward signal, based on a reward function, which measures how well the action has been performed. The goal is maximizing the reward produced, increasing the quality of actions. The last kind of learning is called unsupervised learning; contrary to supervised learning, this particular mode does not provide any ground truth label nor reward functions, so there is not any additional information about

input data. Thus, its goal is extracting meaningful information without any guidance provided by an already known output. A common unsupervised technique is the so-called clustering: its goal is organizing data into subgroups, called cluster so that the elements of a single group are similar to each other and different from elements belonging to other clusters, given a set of features.

Classification vs Regression

Given a set if training dots, and a target variable, called class label, a given machine learning model acts as a classifier if, given a specific input and a mapping function f, it produces an output which is contained in a discrete set of labels. For instance, if a mail classifier should predict whether an input email is either a spam mail or not, it would be a binary classifier, so a model with two possible outcomes. More generally, if a classifier should predict among many labels, thus greater than two, it would be called multilabel classifier. It makes sense to use accuracy as a metric of how well a classifier performs, since it analyses how many predictions are right out of all the predictions made. Regression, on the other hand, predits a continuous outcome. Predicting a house selling price, given a specific model, may be considered as a regression problem. A very common metric used in this field is RMSE, also known as root mean squared error, which measures the standard deviation of the predictions from the ground truth. These examples are inherent to supervised problems, whose goal is defining a trained model, over labeled data, that will then be applied to not seen data, with an unknown label.

2.4.2 Neural Networks

Neural networks are based on the fact that human brain computes in an entire different way from digital computers. It has the ability to organize its structural constituents, knows as neurons, in a way that allows to reach astonishing performances. As explained in [31], a neural network models the way brain produces a particular task. It is made of different layers, structured as a series of fundamental units called neurons. Neurons are massively interconnected among each other. Knowledge is acquired from input data, through a learning process and it is stored in specific parameters which are called weights, specific for each single neuron. A neuron is an information-processing unit. It is characterised by a set of synapses or connecting links, each of which is associated to a weight or strength. For instance, an input signal x_{i} related to synapse i connected to neuron k is multiplied by the synaptic weight w_{ki}. A neuron is also defined by an adder, whose goal is adding

all weighted inputs for the given unit. An activation function, for limiting the amplitude of the output of a neuron. The neuronal model also includes an additional element, called bias, which has the effect of increasing or lowering the net input of the activation, depending on whether is positive or negative, respectively. A neuron may be mathematically described exploiting this pair of equations:

$$u_{\mathrm{k}} = \sum_{i=1}^{m} x_{\mathrm{i}} w_{\mathrm{ki}}$$

and

$$y_{\mathrm{k}} = \phi(b_{\mathrm{k}} + \mu_{\mathrm{k}})$$

As it is showed, the bias has the effect of applying an affine transformation to the output u_{k}. Depending on its sign, either positive or negative, it changes the relationship between the induced local field, thus $v_{\mathrm{k}} = \mu_{\mathrm{k}} + b_{\mathrm{k}}$ and u_{k}. What it is worth noticing is the different active functions that may be exploited whenever building a neural model. Among these functions there is the threshold function, whose outcome is 1 if the induced local field is equal or greater than zero, or zero if it is lower than zero. The sigmoid function, which will be mentioned again later, has an s-shaped graph and it's the most common form of activation function used in the construction of artificial neural networks. An example of the sigmoid function is the logistic function, defined by

$$\frac{1}{1 + e^{\alpha v}}$$

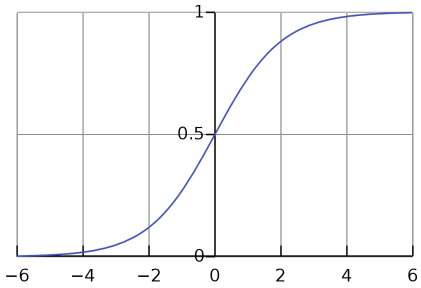


Figure 2.2: Logistic function

2.4.3 Convolutional Neural Networks

Convolutional neural networks, described in [2], briefly known as CNNs, are a particular category of neural networks widely used in image recognition tasks. They

are very similar to the original ones and are made up of neurons that have learnable weights and biases. The main difference with the classic approach is that cnn assume that the inputs are solely images. They can recognize visual patterns with extreme variability (such as handwritten characters) directly from images with minimal preprocessing. This allows encoding certain properties into the architecture which make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. The great limit represented by neural networks, whenever dealing with images, is scalability. Supposing an input image of $32 * 32 * 3$ elements and supposing fully connected layers, the first set of neurons will have a total number of weights of $32 * 32 * 3 = 3072$ each, which is still manageable, but as data input dimension grows, the total number of weights tends to increase excessively, not allowing to work with high dimensional scenarios. Each single layer in a CNN is arranged in three dimensions which are width, height and depth. Given this structure, each single neuron in a layer is not connected to the whole input volume, but just a portion of it, which is called reference field. Every layer transforms the input volume in an output volume of neuron activations. There are different layers that make up an entire convolutional neural network.

Convolutional layer

A convolutional layer is composed of a set of learnable filters. Each single filter, or kernel, is of small spatial dimensions, typically 3, 5 or 7 each axis, but its depth must be equal to the depth of the input volume. During this operation, each filter is slided across the input volume and, for each position, a dot operation is performed among every component of the filter and the corresponding elements of the input volume. The output produced by each filter is a two-dimensional activation map that shows the response of that filter at every spatial position. Intuitively, the network learns filters that are able to activate when a specific edge or color is determined at first layer, or some other structures are determined at deeper layers. Each single filter produces an output layer of depth equal to one, all the layers, equal to the number of filters, are stacked together in order to produce the output volume. The stride of a filter expresses the number of pixels the filter is moved after every convolutional operation. Usually stride is equal to one and a padding of zeros around the input volume is applied, in order to mantain the same spatial dimensions in output. So, the main parameters that can be tweaked are respectively the depth of the output volume, thus the number of learnable filters used in the convolutional layer, the stride of the sliding window and the padding. There is a very simple mathematical formula which allows computing the spatial dimension of the output volume, given the input volume size W, the receptive field size (the spatial dimension of the filter)

of the convolutional layer neurons F, the stride with which they are applied S and the amount of zero padding used on the border P. Here, it is shown the final formula:

$$\frac{W - F + 2P}{S} + 1$$

For instance, the convolution of a 7x7 input volume with a 3x3 filter with stride 1 and padding 0 would lead to a 5x5 output. It is not mandatory to keep the same dimension of the input volume, it really depends on the kind of application the neural network is meant for. What is immediately clear looking at the formula, is that not all the strides can be used. For example, given an input dimension of $W = 10$ and a filter dimension of $F = 3$ with a padding $P = 0$, it would be impossible to use a stride of $S = 2$, since this would lead to 4.5 as the final result, not an integer number.

Max pooling layer

It is very common to place a pooling layer among convolutional operations in order to reduce the overall spatial size of the volume, so that it reduces the amount of parameters and also controls overfitting. The most common pooling operation is based on a filter of 2x2 dimension with a stride equal to 2. For each single portion of the referenced volume, the max value is computed among all the elements contained in the matrix. This leads to the discard of 75% of the total activations. So, given an input volume of $W_1 * H_1 * D_1$, it produces a volume of size of $W_2 * H_2 * D_2$, given two hyperparameters, thus the spatial extent F and the stride S, where $W_2 = \frac{(W_1 - F)}{S} + 1$, $H_2 = \frac{(H_1 - F)}{S} + 1$ and $D_2 = D_1$. Though spatial dimensions reduction is fundamental, in order to apply convolution on a coarse element and extract more complex features, pooling is not the only strategy which can be applied in order to achieve it. There are some situations where it is more desirable to exploit basic convolutional layers whose filters size and stride is specifically determined in order to obtain half of the input in each dimension, as it is showed in [2]

Fully connected layer

A fully connected layer presents neurons whose connections are related to all the activations of the previous layer.

Case studies

There have been several evolutions in the convolutional networks architectures: starting from LeNet, in 1990s, developed by Yann LeCun, that was used to read

zip codes, digits... AlexNet, ZF Net, GoogLeNet, VGGNet and the current state-of-the-art: ResNet or residual network, developed by Kaiming He.

2.4.4 Image Segmentation

It is very important to extract useful information from a given image, thus, being able to split each picture in its basic components. As analysed in [38], it is a widely used technology in medical image processing, face recognition pedestrian detection and so on. As things stand, there are several techniques that may be exploited in order to achieve this result. Among these different algorithms there are: region-based segmentation, clustering techniques (such as k-means), edge detection segmentation and CNN with supervised learning. As explained in [38], regional growth segmentation is a region-based technique and it is based on looking for pixels with similar properties to form a region. After selecting a seed pixel, a given pixel is included in the region if its difference with the seed pixel is lower than a given threshold, thus the importance of choosing the right threshold value. Edge detection, on the other hand, is based on derivative operation, such as sobel and laplace, which looks for discontinuities in the source image. Segmentation based on clustering, such as k-means, relies on determing groups of pixels with similarities. The clustering operation is applied in the feature space, then result is mapped back to the original image obtaining final segmentation. In the last years, there has been a great improvement in image segmentation based on convolutional neural networks

2.4.5 Unet

As mentioned above, convolutional neural networks are widely used in image classification tasks. However, there are many applications where not only would it be important to output a classification label, but also to localize the position of the classified information. In other terms, the what and the where. As explained in [35], in biomedical images tasks it is very important to predict results in a pixelwise manner, thus a label for each single pixel. As previously showed, convolutional neural networks are a succession of different layers, whose outcome is a predicted result. There is the need to apply a further transformation in order to achieve a higher resolution result. This one is obtained using a specific architecture, named u-net. The 'u' reflects the particular shape that its inner logic creates. Common convolutional neural networks show a contraction path which, given a source image, applies several transformations, made especially of convolutional and pooling layers, which leads to a final labeled result. The u-net architecture, besides this particular path, adds an expanding path, made of several upsampling operations, whose goal is obtaining a

final labeled mask. The idea showed in [35] can be split in two parts, the first one, the contraction path, is made of pairs of two convolutions interposed with a max pooling layer, for a total of four. After it, there's the second part, which upsamples the obtained result until obtaining the final outcome. What is worth noticing is the concatenation among each level of the contraction path with the corresponding level of the upsampling part. This allows to combine the high resolution features determined at each step of the first part with the upsampled level of the second part. This helps localization. The contraction path, as it goes deeper, increases the number of features and, complementarily, reduces the spatial size, which is then increased in the upsampling path together with a reduction in depth size.

2.4.6 Data augmentation

[35] shows not only how to classify images in a pixelwise manner, but also a very innovative technique called data augmentation. In the last years, as showed in the ImageNet competitions, neural networks have been trained on giant amount of data, millions of images, which allow to achieve astonishing results in terms of all the images tasks explained above. This is possible thanks to the great hardware performances achieved and the important data sources developed during the last decade. However, not all the application fields may rely on a important amount of information to start with. For instance, biomedical images. This is the reason why data augmentation has been introduced. Data augmentation is based on applying different transformations to some specific input images, in order to get a wide range of input elements, increasing significantly the amount of present data and achieving great performances. Though it may look like this could lead to an overfitting model, it was actually proven in [36] and [30] that, quite the opposite, it avoids overfitting, even reducing it. Augmentation in data-space [37] is a techinque which is based on creating plausible transformations of existing samples that preserve label integrity. Typically, it may require an additional manual analysis in order to verify that the image-label correspondance is still valid. Even though affine transformations, thus transformations, rotations and skewing are commonly applied, exploiting a fixed displacement for each single pixel, elastic distortions are becoming more and more popular.

2.4.7 Utilities and Softwares

There are some tools that, as things stand, are widely used in neural networks applications.

Tensorflow and Keras

Tensorflow [29], is an open source software library explicitely built for high performance numerical computation. It can be run across a variety of platforms, thus CPUs, GPUs and TPUs leading to great speed. What it is immediately obvious is that this library, even though very effective and powerful, is really difficult to use. This is the reason why there is a very powerful framework, built not only on top of Tensorflow, but also on other backends such as Theano and CNTK. It is well known and widely used, it is called Keras [11] . It was built in order to go from idea to a working result with the least possible delay. It supports both convolutional and recurrent networks and it can run either on cpu or gpu, allowing low training time.

Chapter 3

Burnt Area detection - a high level perspective

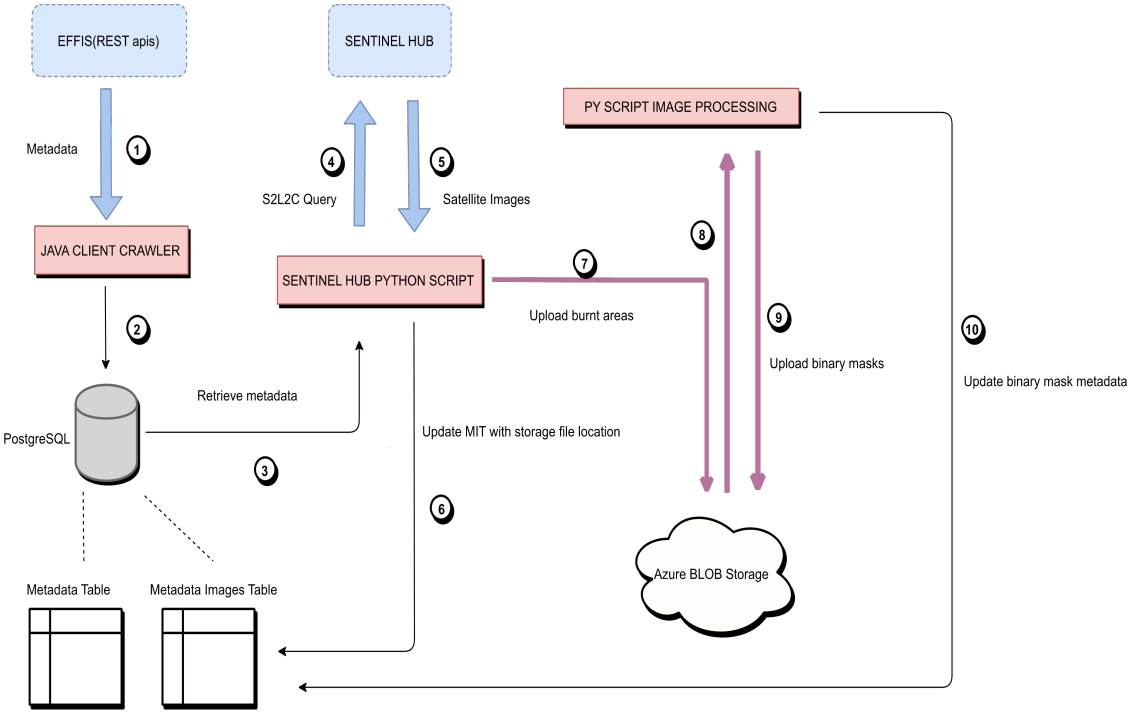


Figure 3.1: Diagram of all the operations

This chapter delineates, in a high level description, the main steps that made up this project, in a chronological order: starting from the retrieving of historical information about burnt areas; the exploitation of this information for acquiring satellite images, the storage of multimedia information and the following image

processing algorithm. The last step is about the implementation of the machine learning algorithm, in order to create the burnt area detection tool.

3.1 Metadata Retrieving

In order to deeply understand fires and burnt area phenomena, it is crucial to retrieve as many data as possible. In this particular field of research, thus, it is of highest priority to collect satellite images showing the areas of interest in specific cropped pictures. As explained in the previous chapter, the Copernicus Programme makes available a remarkable dataset, collected through its missions and, particularly, Sentinel satellites. Though, before being able to take possession of the multimedia information, there is a previous step, which involves metadata retrieving. Obviously, before querying any service providing those images, it is fundamental to know where to look for the information and its quality. Generally, the overall quality of data represents an essential aspect, since the subsequent quality is highly related to the ability of storing very good images, collected from trustable sources. This is, in general, the basis of machine learning and data science. If a certain phenomena has to be analysed, in order to understand it completely and infer the most important characteristics, the collected amount of data it is based on must be of the highest quality possible and, whenever data are available, of relevant size. While the former can be usually achieved exploring all different possibilities and sources until finding a good quality which is suitable for the studied problem, the latter not always can be achieved, since data availabilty is strongly related to the field of analysis and there are specific situations where data shortage is an issue to deal with. For example, medical imaging cannot always rely on huge datasets, so some other techniques must be exploited to achieve a specific result. As it is reported in the previous chapter, this problem can be, at least partially, solved using data augmentation techniques, in order to increase significantly the cardinality of the training set and reduce overfitting.
Metadata are data about data, in other terms, they may be considered, in this specific work, as a list of key-value elements describing the information of interest. In this field of analysis, metadata represent additional information about a certain fire event. It may be reported as simple text containing different kind of information which can be either useful or not, depending on the specific application. For example, while it is fundamental to know exactly where a specific fire has happened and, hopefully, the period of time the burnt area has expanded; it may not be so important to know, for instance, the kind of vegetation destroyed, or the amount of carbon dioxide produced by the event. This actually depends on the particular kind

of research. So, as an example, if working with air quality issues, it may be worth noticing the impact that fires have on the air and the kind of substances produced by combustion; while, if working with image classification, it is crucial to know the exact position and possibly the surrounding polygon coordinates associated with the burnt area. The metadata information collected for this work is mainly related to European fires, starting from the first of January 2015. Metadata may be acquired from different sources or exploiting just a single one. The former approach determines a subsequent data integration operation. Unfortunately, it is rare that different sources bring data in the exact same format or the exact same content. So, the first thing to do is analysing carefully how the information is structured for each single source and find the best way to merge different sources in order to get the best overall information possible. Furthermore, a cleaning operation may be needed, if duplicates occur. Being able to acquire information coming from different sources may help in getting a more precise information, since it also allows to check if the metadata collected are correct. The latter approach, differently, is easier, since data integration step is skipped, but the drawback is not having the possibility to double check the information acquired. After this first operation, it may be necessary an additional filtering step whose goal is reducing the metadata attributes. As explained before, not all the information may be useful for the specific analysis, thus, filtering out these elements reduces the size of remaining data and concentrate the useful information. As soon as data information and data cleaning has been done, it is sufficient to store the collected information in order to approach the following step, which is the one related to data collection, thus multimedia information about burnt areas.

3.2 Image Retrieving

As soon as metadata information has been retrieved, it can be exploited in order to query some services and find the correct crop of a specific area representing the burnt spot. First operation is referring to the right service. Though it may look like a very easy task, it will be shown later how this has been pretty challenging, since there are different websites, frameworks and scripts that may or not do the job. Generally, it is pretty common to deal with different possibilities that may bring the information in different formats. The service hosted by the Copernicus Programme: Copernicus Open Access Hub, provides a way to get those kind of information but, as it will be shown later, it has not been considered as the main solution, since there are some drawbacks which would have slowed down significantly the convergence to the solution. In this particular section the main issues may be related to the presence or

not of the specific queried information, the size of each single information, in terms of downloading time and storage management, the quality of the retrieved image in terms of cloud coverage, area resolution and the ability to either see or not the burnt spots.
As well explained, images acquisition is based on Sentinel 2 imagery as the main source, thus MSI payload. These images are acquired using several bands, so another issue is determing whether these additional bands are either useful or not for the aim of this work. Then, dealing with burnt areas means being able to discriminate, at least visually, what are the exact boundaries of the hotspot. This means being able to understand, for each image, what is burnt and what is not. In order to have a global vision of this problem, it may then be useful not only to collect the image containing the burnt spot, but also its corresponding version before the fire event and a wide surrounding area which clearly delineates the differences among burnt and unburnt areas. This is very important so that an immediate comparison can be done, allowing to easily understand where the hotspots are located. The biggest drawback of this approach is that it increases significantly the amount of information to download and, subsequently, the amount of images that has to be stored. Nevertheless this may be a very good solution, especially if the given manual annotation doesn not fit properly the burnt spot. In any case, it seems obvious that the possible drawbacks and issues are specifically related to the adopted solution, so there may be cases where the amount of data to download does not represent a big issue. As soon as images are retrieved, they will be stored in a suitable storage solution. This is fundamental when working with very big information. Not only finding the right solution for storing information represents a crucial aspect for data management, but also how this information is stored, thus how images are organized and what are the names assigned to each single file, in order to easily acquire the information during the following steps and being able to compare on-the-fly the various contents.

3.3 Image Processing and burnt area detection by means of image comparison

Once images have been retrieved, a further step is studying carefully the information available. The following machine learning analysis relies on a supervised learning approach. As explained in the previous chapter, supervised learning is based on already known labels; this is fundamental so that the entire model may learn from the ground truth information how well it performed during prediction and what is the error entity. In this particular topic, what it is fundamental is retrieving the

burnt area mask for each single fire, in terms of a binary image where the white area represents the hotspot and the black area, complementarily, the unburnt spots. Thus, having a precise idea of the available information represents a key step, since it lets to deeply understand the problem and to acquire, during the process, some extra aspects which have not been analysed before and which may reveal really helpful. The first immediate approach is visual. It is fundamental to understand, at least at a first level, what are the intrinsic characteristics of the burnt area. Typically, they will appear clearly different from the previous situation, and this may come in handy whenever defining distinctively the boundaries of the area of interest. But, even though it may seem visually obvious and immediately recognisable, it is a very slow approach, since it would need a lot of effort in terms of analysing manually each single image and defining the right position of each hotspot. Nevertheless, this is a very common approach, since it lets define accurately the assigned label for each single pixel and the extent of the area; but this usually means having very few images, or many people working on the same task. Since this is not the case, a very intuitive approach to get this process done faster is applying some kind of algorithm which is capable of comparing two images, the one before the fire took place and the one after the fire event, in order to retrieve differences, which, hopefully, represent exactly the area of interest. This, though, doesn't exclude a final manual approach whose goal is refining the autonomous mask extraction, as it will be covered later.

3.4 Machine Learning for burnt area detection

As soon as burnt areas imagery and corresponding labels, represented as binary masks with the same spatial size, are retrieved, the final step has to be implemented. There are a lot of machine learning algorithms, and each of them performs better in specific situations, so it is important to pick the right one, in order to achieve good results. In this particular situation, since image analysis is the cornerstone of the entire work, neural networks represent a good choice, especially convolutional neural networks, widely used in image analysis. As explained before, the long series of layers implemented in a convolution neural network allow to extract features and information at various levels of depth, letting grow the amount of knowledge on each specific image. Supposing that the specific CNN is trained over a good amount of different images, applying also some proper transformations, it will apply segmentation step over each single test image in order to obtain, as output, a binary mask roughly representing the boundaries of the burnt area. In other terms this means that, after sending as input a burnt area image to the network, this applies several convolutional operations and max pooling layers which properly transform

it, in order to obtain a high depth volume, containing high resolution features. Each single training image, feeding the cnn, determine a prediction (regression in a zero-one range) whose values are compared with the mask related to the input, exploiting a specific loss function, as explained later and determing a consequent update of the neural network parameters. This update is based on the application of a gradient over the loss function, determing the direction to follow in terms of weights update. In order to avoid overshoot, so missing the point which minimizes the loss function, a **learning rate** parameter is used as a multiplier to limit the absolute value of weights update. In formulas:

$$L(w; x; y) = f(y, pred(x, w)), \nabla_w f(y, pred(x, w))$$

where L represents the **loss function**, dependent from the inputs **x** and the weights **w**. ∇ the gradient operator.

Chapter 4

Burnt Area detection - a low level perspective

This chapter addresses the burnt area detection issue, analysing it in depth. The steps described in the previous chapter are covered in detail, showing the tools and the implementation decision that were taken during the whole process. The used algorithms are described step by step by means of the techniques and libraries exploited to achieve the result.

4.1 Metadata Retrieving

Effis, as explained in 2.2.2, is a key system whenever dealing with burnt areas and on-going fires. It provides several information that I managed to collect exploiting a specific REST interface, which, at the moment, can be found at: [8]. The first choice of the project was determing which of the exposed application programming interfaces was the one that had to be used. Since the main goal was retrieving an historical view of burnt areas, the most natural option was represented by the endpoint that allowed to retrieve information about past years. Data could be retrieved by means of a http get at the specified rest interface. This rest endpoint offered a series of get parameters, in order to filter out the given result. The most interesting, for the purpose of my project, was **firedate___gt**; it could be used to specify a starting date so that all the results would have been inherent to a fire date greater than the one established. The reason of this choice was related to the fact that the Sentinel Mission 2, which I used as the main source for satellite imagery, was started in 2015, so this limited the research in the last three years (2015-2018). In fact, I used 2015-01-01 as get parameter for limiting the working spectrum. The Effis service provided requested data in a Json format, whose structure was based on

a lot of different keys. The entire json parsing and manipulation was accomplished using the **org.json** library. Though it is not the best option as a library, if talking about performances, it is very easy to use and a valid solution for a once-only operation, just like the one implemented for data retrieving. Here, it is shown a basic structure of the json representing a specific burnt area, which is quite rich in terms of additional information, for different purposes:


```
{
        "count": 19235,
        "next": "http://effis.jrc.ec.europa.eu/rest/2/
           burntareas/historical/?limit=1&offset=1",
        "previous": null,
        "results": [
        {
         "objectid": 58980,
         "firedate": "2017-09-01",
         "area_ha": 116,
         "place_name": "SelishtÃ«",
         "province": "Dibres",
         "lastupdate": "2017-09-07",
         "yearid": "2017_190290",
         "id_source": 190290,
         "yearseason": 2017,
         "fireid": 35262,
         "flag": "",
         "shape": {
                 "type": "Polygon",
                 "coordinates":
                 [
                  [
                    [ 20.309307919, 41.589256176 ],
                    [ 20.309259531, 41.589313967 ],
                    [ 20.30920728,   41.589371086 ],
                    ...
                  ]
                ]
         },
         "countryfullname": "Albania",
         "lastfiredate": "2017-09-02",
```

```
32            "lastfiretime": "11:34:00",
33            "country": "AL"
34    }
35   ]
36 }
```

As it is shown in the listing, a single burnt area is made up of several key-value pairs. The code above refers to a get query, sent to the Effis Service, specifying a parameter **limit** = 1, in order to reduce the returned elements to just one single hotspot. Among all these elements it is worth noticing the **results** object array, which reports a json object, showing some data related to the area of interest. The **firedate** key refers to the date the fire event was originated, while, complementarily, **lastfiredate** and **lastfiretime** should refer to the last date and the last time of ongoing fire respectively. Geographic information are reported as **country**, which refers to the country code, AL in the example; **countryfullname**, **place name** and **province**. The most important information that can be acquired referring to the json document is the **shape** object and, specifically, the **coordinates** array, which reports the list of pair longitude, latitude for each single vertex of the polygon surrounding the area of interest. Those coordinates were reported in the widely used EPSG 4326 coordinate system. It is important to highlight that the polygon is retrieved as a result of a manual annotation, thus it is not guaranteed that it respects perfectly the shape of the burnt area. My goal was retrieving just the information useful for the particular purpose of my analysis. There are two important questions to answer in order to locate the hotspot: the *where* and the *when*. *Where* represents the place, but, instead of retrieving either the country or the city of interest, the most important information was represented by the polygon coordinates. It is obvious that the former is superfluous when you can obtain the latter. Besides the *where*, there's the *when*; it was crucial to retrieve the temporal range of the fire event in order to determine not only when the fire event happened, but when it was finished too. I also decided to collect the extension of the burnt area, which is reported by the **area ha** attribute, so, as it can be noticed from the parameter name, the used unit of measure is hectares. All these operations have been accomplished exploiting the **JAX-RS** framework and the **Jersey** restful api implementation. The following operation was represented by the storage solution. As explained in 2.2.3, the solution adopted relied on the open-source dbms: PostgreSQL. The metadata information were organized exploiting a single table, reporting for each single row the information related to a single burnt area. So, besides the auto-increment id, the other information were the **firedate**, **lastupdate**, **area ha** and **coordinates**. An extract of the table is showed below:

Id	FireDate	Coords	AreaHA	Bbox	LastUpdate
1	2017-01-02	0103000...	274	null	2017-01-03
2	2017-01-13	0103000...	24	null	2017-01-14
...	...	...	...	...	...

Table 4.1: Retrieved metadata example before ST_Extent

The result table represents the information of each burnt area in each single row. As it is immediately noticeable, there are six columns in total, one of them is represented as null. The reason why, at first, this information was equal to null, was related to the fact that the bounding box column could not be retrieved from the information provided by the json file. It was fundamental not only to collect information related to the burnt area perimeter, but also retrieving the smallest rectangular bounding box, immediately obtainable from the coordinates of the polygon. I managed to achieve this particular task exploiting the function given by the PostGIS extension: **ST_Extent**. **ST_Extent** takes as input the list of coordinates of the polygon and gives, as output, a bounding box reporting two points, the lower left and the upper right point of the surrounding rectangular area; each of this point is reported in a longitude, latitude order. As soon as the sql query was done, this was the output table:

Id	FireDate	Coords	AreaHA	Bbox	LastUpdate
1	2017-01-02	0103000...	274	BOX(18.76 43.86, 18.79, 43.87)	2017-01-03
2	2017-01-13	0103000...	24	BOX(15.47 42.10, 15.48 42.11)	2017-01-14
...	...	...	...	...	...

Table 4.2: Retrieved metadata example after ST_Extent

The Bbox column is reported with coordinates that present only two decimals point for representability reasons. There are two pair of points, where the former represents the lower left point of the bounding box, the latter the upper right. Looking at the first row, this means that the burnt area can be found within a bounding box described by a left longitude of 18.7°, a lower latitude of 43.86°, a right longitude of 18.79°and an upper latitude of 43.87°.
The metadata table presented above, considering the applied get parameter, filtering results starting from 01-01-2015, is composed of 21889 different burnt areas. These data are mainly related to Southern Europe, Turkey and North Africa, since the Mediterranean Region represents one of the most affected areas in the world.

4.2 Satellite imagery retrieving

Before enunciating in details how I managed to implement the retrieving of images, it is worth showing an overview of the methods that, as things stand at the moment, are exploitable in order to retrieve satellite imagery and, additionally, metadata information.

4.2.1 An overview of satellite data

In-depth overview of Sentinel Missions

As mentioned in the first chapter, the first two Sentinel Missions, were launched respectively in April 2014 and June 2015. Both of these missions rely on two orbiting satellites, but with different purposes and on-board instruments. Sentinel-1 mission is provided with a SAR (Syntethic Aperture Radar) instrument, which is capable, working in a C-band spectrum range, to acquire data over day and night, on every site, independently from clouds and weather conditions. Sentinel-1A and Sentinel-1B are the names assigned to the two satellites working on this mission, which share the same orbital plane. The capability of acquiring continuously landmasses and ocean information gives access to data of the same site on a bi-weekly basis for the entire world. Sentinel-1 satellites work on four different acquisition modes: **stripmap**, which provides coverage based on a $5mx5m$ resolution over a swath of 80 km; **interferometric wide swath mode**, which is used on landmasses, based on a $5mx20m$ geometric resolution over a swath of 250km; **extra wide swath** which is based on the previous technique, but relies on a wider swath of 400 km at $20mx40m$ spatial resolution; while the last one is the **Wave** technique, especially used for ocean analysis, it is based on the acquisition of a series of 20kmx20km at $5mx5m$ resolution vignettes, at two different incidence angles so that each vignette is separated by 100 km and two following vignettes with the same incidence angle of 200 km. Sentinel 2, on the other side, is supplied with a MSI(Multispectral Instrument) payload, which samples 13 spectral bands: four at 10 metres, six bands at 20 metres and three bands at 60 metres spatial resolution. The elementary level of Sentinel2-MSI products are granules of a fixed size, whose dimension depends on the particular level analysed, from 25kmx23km to 100kmx100km.

Sentinel Products

As already mentioned, Sentinel data are provided as products. Those can be considered as compressed data, holding various information and organized into different possible levels. For what concerns Sentinel-1 mission, there are four different levels,

each of them available for each acquisition mode. Level-0 are the basic products, which consist of compressed and unfocused SAR raw data. This represents the basis from which all the other products are built. Level-1 products are then obtained from raw data by applying a series of algorithms, obtaining two possible product classes: Single Look Complex (SLC) and Ground Range Detected (GRD) [28]. Sentinel level 3 products, known as OCN, contain information about ocean analysis, such as ocean wind field, ocean swell spectra, surface radial velocity. Sentinel-2 mission, on the other hand, has five different levels of products, but neither Level-0, nor Level-1A and Level-1B are public formats. The two levels made available are the Level-1C, which provides a orthorectified Top-Of-Atmosphere reflectance and Level-2A which containts Bottom-Of-Atmosphere reflectance and a classification map. The latter can be obtained from the former by the user itself with proper softwares, like SNAP. Sentinel products are organized into a SAFE format specification, specifically a variation of the Standard Archive Format for Europe. It wraps several elements which are explained in more details right below. Sentinel 1 product is composed of different elements, such as a manifest.safe, which is an xml document containg the mandatory metadata common to all SENTINEL-1 products; the Annotation folder contains information related to the measurements, their characteristics and how they are generated. The binary data are contained in the Measurement folder; these information are the instrument data for level-0 products, processed images for level-1 and derived data for level-2. A comprehensive review of product formats can be found at: [16], [17], [18]. The focus of this analysis must be on Sentinel-2 data formats and data naming specifications. Sentinel-2 products are organized in a different way from the previous mission products. There's, of course, a manifest.safe file, which, as in the previous case, contains all the metadata information; a granule folder, which contains one or more granule subfolders, each one corresponding to a tile composing the level 2A user product. Images are compressed in jpeg2000 format and saved in different files for each different band in different resolutions. thus images are organized into three folders containing respectively images at 10, 20 and 60 metres resolution. Datastrip folder contains the list of folder, each one corresponding to the datastrips making up the Level-2A user product. Data structure is defined in detail in: [19].

Copernicus Open Access Hub

Copernicus Open Access Hub, which was known in the past as Sentinel Scientific Data Hub represents the official, free and most important information aggregator for, as things stand at the moment, Sentinel-1, Sentinel-2 and Sentinel-3 products; as reported on the website, Sentinel-5P products will be made available soon. It

offers several ways to properly interact with data. The availabilty of the products, of any level, is generally close to the time of the acquisition, even though it depends on the specific area. In real time context, the online availability of product is no later than 100 minutes after data sensing, while in near real-time (nrt), the online availability of product is between 100 minutes and 3 hours after data sensing. One way to access information is: **Open Hub**; it is the access point for all the Sentinel Missions, providing access to the graphical user interface. This hub, also known as data hub, is a java web based system which was designed for handling and delivering all kind of products supplied by various Sentinel missions. As mentioned above, it is based on a trivial gui, which includes a search box properly defined in order to filter out results and look for the products of interest. The search box presents a lot of different options available, such as **sort by**, which is based on three parameters: sensing date, ingestion date and cloud coverage. It is worth noticing the difference between the two date parameters: the former represents the time of acquisition of a given product, whose content depends on the kind of Sentinel mission, while the latter represents the time when the product has been made available in the hub. It is possible to define a time range for both sensing and ingestion period, but the most interesting part is represented by the choice among three different Sentinel Missions: 1, 2, 3. So, for each diffent mission, it may be specified the satellite platform, the product type used, the polarisation and the sensor mode, thus the different modes of acquisition exploited by the system (the latter is specifically for Sentinel 1 satellites). Besides these filters, it is possible to create specific text queries, which enhance the possibilities given by the search box; it is, for instance, possible to define a polygon, looking for all the tiles that either intersect or contain the given polygon. The access point for the API service, available at: [3] provides a lot of extra possibilities in order to retrieve products from client applications and external scripts. It actually exposes two different api access points: the **Open Data Protocol** and the **Open Search** services. Both of them represent technologies build on top of the REST technology. The open data protocol allows to easily retrieve information from the data hub. Starting from the base url, it is possible to navigate through all the resource provided by the service, such as the Products one. Querying at Products, means retrieving a list of fixed entries, representing some of the available products, each of them reporting the name, the UUID and the download URI. It is obviously possible to achieve further customization by using several parameters, such as format, in order to personalize the response retrieved from the service, filter, orderby and so on. OpenSearch can be considered as the complementary service of Odata, it can be queried using several parameters, such as Polygon, in order to specify specific boundaries of a given hotspot, exploiting the same keywords used in the data hub system. So, there are many ways to access data provided by

the official aggregator, it is quite easy to get information, which is continuously updated with a very low delay, unfortunately, the main drawback is represented by products manipulation, which is not easy, in terms of bands processing, merging and eventually, for the purposes of this project, band cropping. There is a know tool: **Snap**, which allows products manipulation but, it is based on a manual analysis of all the products, this is the reason why a python library called **Snappy** may come in handy in this particular situation.

Sentinel Sat

Sentinel Sat [27] is a python library which provides several apis in order to easily access and interact with Copernicus Open Access Hub. It gives the user the possibility to either use the interfaces to interact with the Open Data Protocol or the Open Search one. At first, it was developed in order to give an easy-to-use tool to access metadata of every queried Sentinel-1, Sentinel-2, Sentinel-3 product; but, as things stand at the moment, it also provides some methods to directly download products, specifying different query parameters. Supposing that the user is interested in all the products related to a specific area, defined by some polygon coordinates and at a specific time range, it is possible to query the service, specifying those parameters, in order to retrieve all the metadata related to the specified filters. This library represents a very powerful way to bypass the direct usage of data hub api, giving full access to all the features provided by the original service. For instance, the **query()** method of the SentinelAPI class allows to specificy not only some fixed parameters, but all the keyword parameters decided by the user, in order to face possible evolutions of the referenced service. The **query_raw**, as explained in the documentation, allows to do a full-text query on the OpenSearch API using the format specified in the Copernicus Open Access Hub.

Google Earth Engine

Google Earth Engine [9] is a planetary-scale platform for Earth science data and analysis, powered by Google's cloud infrastructure. Its dataset is of great importance and its information comes from different sources. Specifically, the satellites the imagery is taken from are Sentinel missions, 1 and 2, Landsat, Modis, High-Resolution Imagery, Defense Meteorological Satellite Program's Operational Linescan System. Landsat is a long lasting joint program of the USGS and NASA, which has been observing the Earth continuously from 1972 through the present day. Today Landsat satellites make acquisitions at 30-meter resolution about once every two weeks, including multispectral and thermal data. Earth Engine makes this data available

as level-0 products, thus raw form, as TOA-corrected reflectance, and as computed products such as EVI and NDVI vegetation indices. The high-resolution imagery are brought by the US National Agriculture Imagery Program at one-meter resolution. This imagery covers almost the entire US, starting from 2003. Old images were retrieved using RGB channels, newer images are provided using a near infrared band. So, Google has been capable to build a vast dataset of information, which can be properly queried for different purposes. Besides imagery, it is also possible to exploit data related to geophysical, climate, weather and demographic topics [6]. The service is free of charge for private users and research projects, while a pricing policy may be applied for commercial purposes, but typically Google provides paid commercial licenses for appropriate use cases. There are several ways to interact with the data. Google provides a graphic user interface, available at [10] and an online integrated code editor, with a full documentation, in order to work with data of interests. The idea delivered by Google is exploiting its astonishing resources not only to properly get in touch with all these data, but also being able to properly process the information acquired applying custom algorithms developed by the user, letting the developer publish results on any web portal. Exploiting the online IDE it is possible to see immediate results, by applying the apis provided by Google under the javascript class **ee**. The functions at disposal provides the user with very powerful instruments, that let not only to retrieve the data of interest, but also properly manipulating the information, the bands and ease the usage of machine learning algorithms. All these apis are also available for Python, in a wrapper library. The biggest drawback of the service provided by Google is related to the need to use their services; this means that, whenever exporting data, there's no way to save them locally, but it is just possible to export information to Google Drive or to Google Cloud Storage. Obviously, the free amount of storage is very limited, so this may represent an important obstacle in using this kind of service.

Sentinel Hub

Sentinel Hub [25] is an additional service which handles satellite imagery. The most interesting thing represented by this service is that, as reported on its documentation, it makes satellite data, from all the available sources, easily accessible for browsing and analysing, exploiting different possibilities, without worrying about synchronization, storage, processing, de-compressing algorithms, metadata or sensor bands. It provides many features, such as global coverage, multi-temporal processing, custom scripting, access of imagery at any scale and preconfigured eo products. Just like previous services, SentinelHub relies on many data sources, such as Sentinel-2, even though the available products are restricted to L1C, and to L2A just

for a limited area over Europe starting from March 2017. Sentinel-1 and Sentinel-3 products are also available. Landsat, Triple-CubeSata, designed and manufactured by Planet Labs inc., Envisat Meris, Digital Elevation Model, which is a 3d representation of a terrain's surface created from terrain elevation data, Modis sensor, which is hosted on Amazon Web Service and Commercial Data. Data are usually available as soon as ESA uploads products to the official hub, taking into account the time needed to download the new information and to process the data. Sentinel Hub delivers different services in order to properly interact with its processed information. One of those services is called: **Playground**, which is an interactive online tool giving the possibility to discover and explore Sentinel-2 imagery; the user may look for scenes, band combinations and it allows to apply additional image effects. EO Browser, on the other hand, combines a complete archive of Sentinel-1, Sentinel-2, Sentinel-3, ESA's archive of Landsat 5, 7 and 8, coverage of Landsat 8, Envisat Meris, Prova-V and MODIS products, allowing to look for specific time range, cloud coverage and sources; exploring the map and downloading the images of interest. The images can be prepared and visualized exploiting different representations, which are based on different merging of the provided bands; for instance images may be visualized exploiting the TRUE COLOR visualization, by means of B4 B3 and B2 bands, NDVI based on combination of bands (B3-B8)/(B3+B8). Sentinel hub also offers a python api [26], which provides several interfaces to deliver some of the functionalities provided by the original service. The access point is provided by an OGC API exploiting either WMS, WMTS, WFS or WCS standards. WMS (Web Map Service) conforms to the WMS standard, so it provides a base url and several filter parameters in order to get the queried results. It provides either access to the 13 unprocessed bands or to processed products such as true color imagery, NDWI index and so on. Standard WMS URL parameters are service, version, request, time, bbox. WCS, WMTS and WFS offer similar capabilities, even though each of them present some peculiarities [12]. Sentinel Hub offers a trial period of 30 days, where the user can access all the features provided by the service, then there's the possibility to get different subscription plans with different features provided. In the last period Sentinel Hub started offering its products on the Amazon Web Service, which provides better performances than the original hub of the Copernicus Programme.

Additional Services

There are several additional services which provide access to satellite imagery coming from various sources. For example: [24], which presents an almost cloudless view of the entire world, combining pixels acquired during an important time range.

Another service is: s2.boku.eodc.eu [21], provided by EODC. EODC is the Earth Observation data provider and distributor, specialized in data procurement, management and efficient processing. It acquires data from ESA's data hubs and make information available on the EODC archive. It currently delivers only Sentinel products, specifically Sentinel-1, Sentinel-2 and Sentinel-3 missions. It is based on a collaborative IT infrastructure for archiving, processing and distributing EO data. It can be queried exploiting a REST interface, whose reference may be found at: [22]. The main resources exposed by the service are **product**, which represents the Sentinel product, **granule**, representing the elementary tile of each single product; also in this case it is possible to specify several parameters in order to retrieve the desired results, image, on the other hand, allows to retrieve a jpeg2000 file, related to the specific filters specified, such as a date range, a given resolution, a given band. There's a R api which wraps the mentioned interfaces in order to properly interact with the service from a client application: [23]. Unfortunately, the service is not free, each granule costs one credit and the number of free credits, after registering, is equal to three. Each time a new granule is needed for user purposes, it must be bought from the service in order to be able to proceed with further analysis.

Copernicus Open Access Hub (Science Hub)

- **resource link** https://scihub.copernicus.eu/
- **sources** Sentinel 1 (Level0, Level1, Level2), Sentinel2(L1C, L2A) and Sentinel3 (Level1, Level2), Sentinel 5P will soon be available
- **data availability** 1hr, 3hr up to 1 day
- **pros** Free
- **cons** Products Management and Post processing
- **GUI** OpenHUB
- **Pricing** Free

Sentinel Sat

- **resource link** https://pypi.org/project/sentinelsat/
- **sources** Copernicus Sources
- **data availability** Copernicus Availability

- **pros** Easy access to Copernicus services
- **cons** Products Management and Post processing
- **GUI** None
- **Pricing** Free

Earth Engine

- **resource link** https://earthengine.google.com/
- **sources** Landsat, Sentinel, MODIS, NAIP...
- **data availability** N.A.
- **pros** Powerful instruments, Google cloud platform
- **cons** Locked in Google Services
- **GUI** Explorer
- **Pricing** Free for research, education, and nonprofit use

Sentinel Hub

- **resource link** https://www.sentinel-hub.com/
- **sources** Sentinel-2, Sentinel-3, Sentinel-1, Landsat, Planet Labs, Envisat Meris, DEM, Modis, Commercial Data
- **data availability** As soon as available on Copernicus Open Access Hub ->Time Downloading + Time Processing
- **pros** Easy to use, no products processing
- **cons** Limitations, Pricing
- **GUI** Playground and EO Browser
- **Pricing** (30 days trial) from 0 to 550 euros per month (free research licenses)

Sentinel Hub Py

- **resource link** https://sentinelhub-py.readthedocs.io/en/latest/
- **sources** Same sources of Sentinel HUN

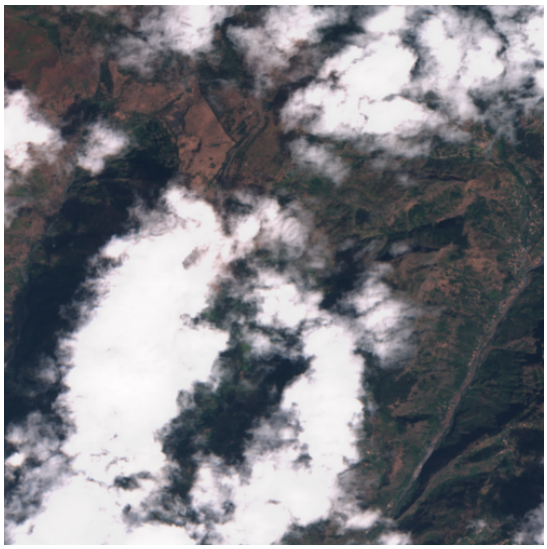
- **data availability** As soon as available on Copernicus Open Access Hub ->Time Downloading + Time Processing
- **pros** Wrapper library to Sentinel Hub API
- **cons** Limitations (Limited amount of request per minute), Pricing
- **GUI** None
- **Pricing** Sentinel-Hub pricing

4.2.2 Satellite imagery retrieving - How

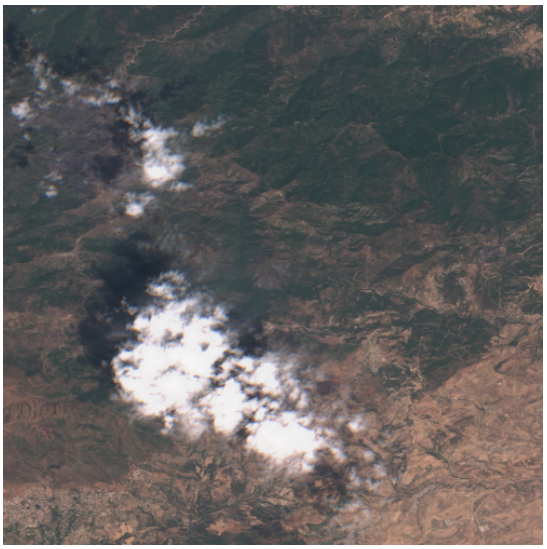
The idea behind images retrieving was the following: if a given area was hit by a fire event, the burnt land determined by it may be defined comparing the given area before the fire event and the same area after the fire event, close to the date of the end of fire. So, this may be resumed saying that there was a **before** and **after** period of interest that had to be analysed for each single fire event, so a before and after image. In order to retrieve the images whose metadata were collected in the previous step (see 4.1), I decided to use Sentinel Hub and, specifically, the Sentinel Hub python library previously mentioned. The reason of this choice is related to a very simple consideration: handling source products takes a long time, because of their size, which is usually equal to hundred of megabytes up to some gigabytes and because of post processing: as mentioned before each product is made of several bands, so there is a first need to merge those bands in order to get the desired result and applying a crop operation so that the desired area of interest can be taken into account. Even though it is technically possible to work directly with low level products, it is also advantageous to work with Sentinel Hub service, since products have already been processed and imagery can be directly accessed. All the imagery retrieved relies on the Sentinel 2 L1C products. As saw before, Sentinel Hub python library offers the possibility to work with different interfaces provived by the OGC. The one I decided to use is the Web Map Service. Before being able to properly download imagery from the Sentinel Hub service, it was crucial to retrieve the metadata saved in the previous step in the Postgresql database. Metadata retrieving was possible by means of psycopg2: a specific python library which allows to interact with Postgresql database. Among the standards provided by OGC and supported by the service, the one I decided to exploit was Web Map Service, because of the parameters and the functionalities it provides. The key metadata retrieved were the firedate, the lastupdate and the bounding box for each single fire event. Now, as mentioned previously, the bounding box was obtained starting from a Polygon geometry by

means of the PostGIS function ST_Extent; this functions returns two points: a lower left and an upper right point, as a longitude, latitude pair each. Since the wms calls imposed that the bounding box was expressed as an upper left point and a lower right point in a latitude, longitude order; I had to properly modify the previous expression. The fire geometry expressed the **where**, while the two dates the **when**. The imagery retrieving was made of two subsequent operations expressed as two wms requests. Since requests were a lot and the entire process would have taken a long time, I decided to implement a concurrent software where three threads were committed to querying the Sentinel Hub service, getting better performances. The first attempt was retrieving the image representing the specific area before the fire took place, while the second one, complementarily aimed to obtain the area after the fire event. Obviously the mentioned operations had to be done in a time range constraint, in order to make sure that the images were as faithful as possible to the situations described. So, concerning the first retrieving, the image search started from the firedate minus one day going backwards, until a result was found. What it's clear is that it was important to set a limit, in order to avoid to look for the image too far in the past. This means that, if no images were found for the set time range, that specific event would have been skipped, going immediately to the next one. For a single wms calls there were some parameters specified such as: **layer**, which was used to specifiy the output bands of the image; the bounding box, specifying the upper left and lower right corner of the retrieved rectangular geometry. What it is worth noticing is that I did not use the closest bounding box to the polygon coordinates, but a wider one, which could include some unburnt areas, since I thought that would have been necessary for the following operations in terms of clear distinction between the hotspot and the surrounding areas. Besides the bounding box I used the width and height parameters in order to specify the exact dimensions of each output and make sure that it was the same for all the retrieved images. There actually were two possibilities: defining the height and/or width or the spatial resolution, in terms of meters per pixel. The former represented the final choice since it allowed to define a fixed dimension, that I set at 5000x5000 pixels. Finally, the last parameter specified was the cloud coverage, going from 0 to 1, in order to define the percentage of clouds of the queried tile. It is important to highlight that the cloud coverage was referred to the entire tile containing the queried polygon, instead of the solely area of interest. As soon as the first image was retrieved, the process was re-applied for the following one, this time looking for a result containing the burnt area. In order to maximize the probability of a good confirmation, the time range was set at 15 days, just like the previous example. Again, the parameters used were the same, except for the days of analysis, which were related to the days after the fire event. Supposing a successful outcome, the

following operation was determing if those images could be good candidates for further operations. Unfortunately, not all the images could be considered valid, since some of them could contain a very high cloud coverage percentage or some bad light conditions. Since looking at each single image, using a manual approach, would have taken too much time and effort, I managed to implement an empiric solution, based on image sizes. Supposing that two images were too different in terms of size, it could mean that at least one of them lacked in terms of content, so this would have led to a bad comparison between the two, as well as images whose size was under a certain threshold, probably meaning some clouds or white areas. In the following page it is shown four examples of satellite images showing clouds:

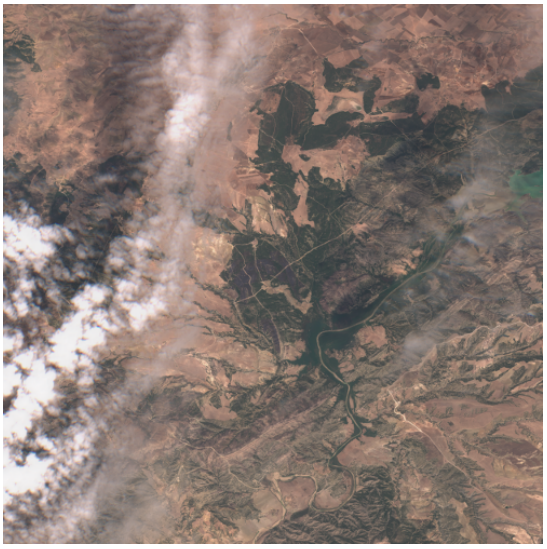


(a) Madera, Portogallo



(b) Menaceur, Algeria

(c) Belmonte Calabro, Italia



(d) Tunisia

Figure 4.1: Cloudy Areas

As it may be clear, images reported in 4.1 are compromised in terms of visibility over the ground area, so they were not really useful.
All the images acquired were then stored in the Azure Blob Storage application. The naming was related to the coordinates of the bounding box used to query the service, plus a "before" or an "after" keyword to either refer to the image before the fire event or the one after it. In order to avoid any name collision between different fire events, a timestamp was appended at the end of every name.

4.2.3 Satellite imagery processing

As soon as all the images were properly collected and stored in the Azure Blob Storage, the following step was represented by the creation of a specific annotation for each single datum. The so-called annotation had to be created so that the burnt area could be properly highlighted over the unburnt one. Even though a manual definition of the polygon coordinates was available, I preferred not to use it because, after conducting some experiments, it showed that it did not describe accurately the boundaries of burnt areas. There could be several reasons for this kind of behaviour. An explanation may be found in the stretching of the image, applied to get squared images of 5000x5000 pixels, which could determine a translation of the hotspot in respect of the annotation provided. Another reason may be found in the linear conversion between pixel and corresponding geographic coordinates, which did not take into account the shape of the Earth, resulting in a not so accurate result. In order to overcome the described issues, it became necessary to find another solution, which could define a proper mask for the burnt area, without using the manual annotation as the main information. This meant that I was forced to implement some kind of comparison between two images, representing the same area, in the before and after fire event conditions, so that the differences and then, hopefully, only the burnt spots could emerge. In the experiments chapter it will be described in details the steps I took to get to the final result, also explaining the good things about the automatic approach implemented and the possible drawbacks of it. The basic idea of all the approaches was working with image processing algorithm in order to figure out some possible features and characteristics that could let the burnt area stand out. As shown in the first chapter there are many color models and, in general, many ways to represent color information, as well as many ways to apply transformations over that information in order to exalt or repress some aspect of it. All these operations were conducted exploiting the numpy and opencv libraries, plus a visual approach, during the tuning of the algorithm, to determine empirically if the considered choices could be either evaluated as correct or not. The algorithm goal was, thus, having as input a sequence of pair of images, representing

the before and after area of interest, retrieving the binary mask representing the hotspot under analysis. In general, a binary mask is an image reporting white areas and black areas, where the white spots represent the positive elements, while the black spots represent the negative ones. For the purpose of this research, the positive event was considered related to the burnt area, which was then represented as white in the final binary mask, while the negative event, complementarily, was considered to be the unburnt sposts, represented in black. The final approach, which gave the best results in term of the hotspot identification, consisted of transforming the rgb model of the input images into the hsv one, before applying any other step in the algorithm.

(a) Vagos, Portugal - after fire event

(b) Vagos, Portugal - before fire event

The images above represent one of those "tough" cases, since there are different lights conditions which, at least visually, may disturb the correct identification of the burnt area. But, taking a look at the absolute values of both rgb and hsv models, it is self-evident how the latter behaves better in terms of distinguish the hotspot from the surrounding, even though the colors may look different all over the image. The explanation is an immediate consequence of what was explained in the first chapter, in the hsv and rgb models section. The hsv model splits the chroma component from the luma one and the former is represented by the hue value. This means that, even though two colors may look completely different, it is mainly related to a different "quantity of light" of the same hue component, which results, at least apparently, in two different colors. While the rgb model, which represents in its information both luma and chroma, merged together, cannot define a proper way to represent these information separetely, resulting in the impossibility of determing any relationship

between the rgb color of the same pixel of the first image and the rgb color of the same pixel of the second image. Tables 4.3 and 4.4 represent the rgb and hsv values for both before and after image for a pixel outside of the burnt area and a pixel inside of the burnt area:

Image	Pixel Coordinates	RGB	HSV
Before	(920, 3204)	(22, 34, 44)	(209,47,18)
After	(920, 3204)	(43, 55, 64)	(206, 33, 25)

Table 4.3: Example of RGB and HSV values for a pixel outside of the burnt area

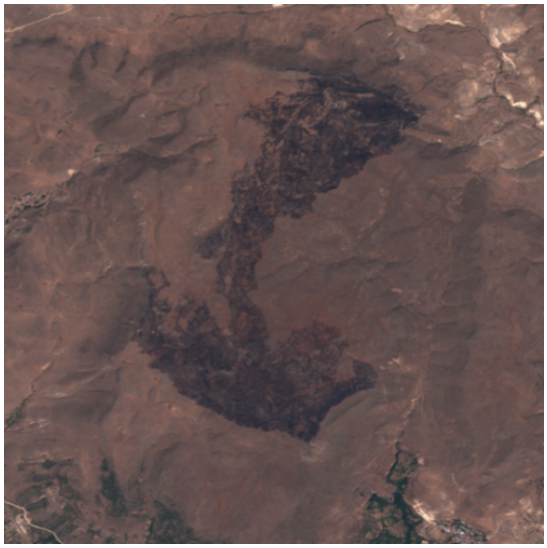
As it is immediately noticeable, the r, g, b components don't have a visible relationship between the two pixels, so this means that it would be very difficult to extract any kind of information to confirm the integrity of the ground and, thus, the absence of a burnt area. What it is, on the other side, pretty evident, is that the hue component of the two pixels is pretty similar, just three points of difference. This means that, even though, apparently, the two colors are way different, it is actually confirmed that their hue component difference is within a small threshold which may be exploitable for further analysis. If two pictures of the same area are taken with different light conditions, the hue component of corresponding pixels will be pretty similar, while the saturation and the value component will tend to diverge, because of a different light contribution. Here below (4.4) are reported the different values of two corresponding pixels related to the burnt area. Obviously, the pixel of the before image does not contain any "burnt" information while, the pixel of the second image, is referring to a specific point in the hotspot.

Image	Pixel Coordinates	RGB	HSV
Before	(2732, 3496)	(41 62 63)	(183 35 25)
After	(2732, 3496)	(36 40 53)	(226 32 21)

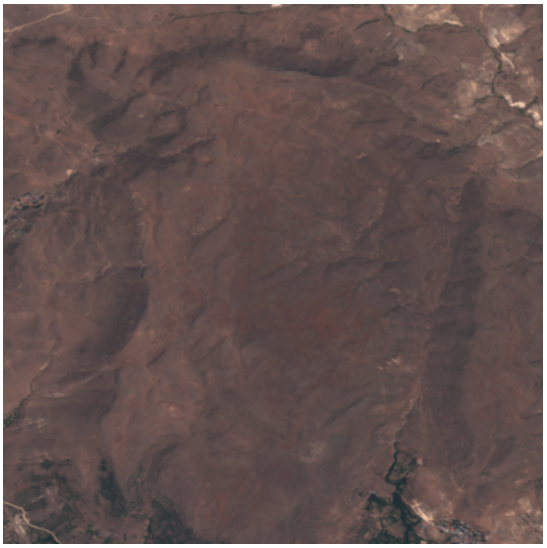
Table 4.4: Example of RGB and HSV values for a pixel inside of the burnt area

The r, g, b components, still, do not show any human readable information, it is just a series of values with no evident meaning. Again, what it is worth noticing is the hue component of the two images. 226 is the value related to the after image, while 183 is the one corresponding to the before image. Their difference is pretty considerable, and this represents the hint that it is the kind of information useful for defining the boundaries of the area of interest. This information was the one exploited in the binary mask creator algorithm. Taking a before and an after image as input, the hue values of both could lead to a mask representing the burnt area

pretty consistently in all the fire event pairs. Here below it is reported a more evident example of a burnt area:



(a) Danisment, Turkey - after fire event



(b) Danisment, Turkey - before fire event

The burnt area stands out, more evidently than in the previous case. This is due to better light conditions, thus, probably, the two images were taken in the same period of the day. 4.5 reports the results for a pixel outside of the burnt area and a pixel within it:

Image	**Pixel Coordinates**	**RGB**	**HSV**
Before	(304, 1760)	(106 83 78)	(11 26 42)
After	(304, 1760)	(123 95 87)	(13 29 48)

Table 4.5: Example of RGB and HSV values for a pixel outside of the burnt area

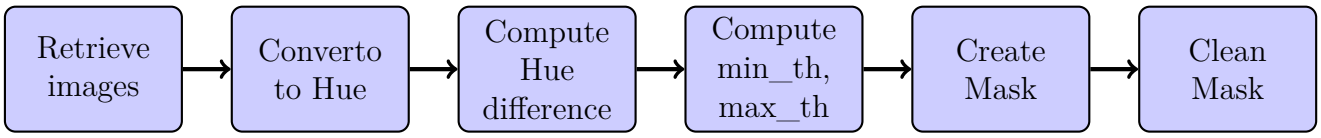
In this example, it's immediately clear that the color shades tend to be more uniform between the two images. This perceptual aspect reflects on the information reported in the table. The h, s, v values are pretty similar: the hue component is almost the same, as the saturation and value elements. What it is worth noticing is that the r, g, b components look different. This is due to the nature of the rgb model, in fact, taking a look at a color visualizer, the rgb tuple (106, 83, 78) corresponds to: , while (123, 95, 87) corresponds to: . So, though those pixels are pretty similar in terms of shades, still, their rgb model doesn't reflect this information. This aspect strengthens the idea that rgb doesn't suit this kind of analysis, while

Image	Pixel Coordinates	RGB	HSV
Before	(2088, 3632)	(91, 75, 75)	(0, 18, 36)
After	(2088, 3632)	(64, 58, 65)	(291, 11, 25)

Table 4.6: Example of RGB and HSV values for a pixel outside of the burnt area

hsv tends to reflect what it is visible more easily. Looking at a pixel of the burnt area 4.6 show the advantages provided by HSV.

Omitting the rgb analysis, which would lead to the same considerations, again, the hue difference between the before and after values is important. The hue difference value, measured between 0 and 359 degrees, is determined as the smallest difference between the two points on the circumference, as it will be shown later; so, in this case, it is equal to 69 degrees. This difference is also clear looking at the image, the before hsv value corresponds to: ■, while the after hsv value corresponds to: ■. Looking at these examples, it is clear that the hsv approach represents a valid solution which is the one implemented in the algorithm. Here, it is shown the high level workflow of the algorithm exploited in order to create the binary mask creator:

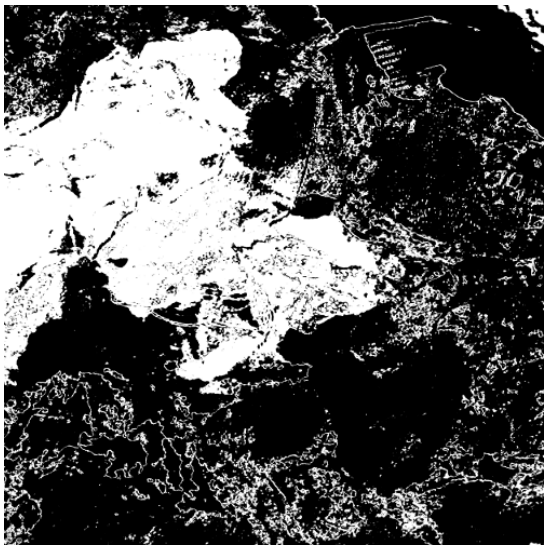


In details, the steps taken in order to implement this process were, at first, retrieving the images, for each single fire event, in order, from the Azure Blob Storage service. So, exploiting the psycopg2 library, after collecting metadata information containing the coordinates of the polygon, file name after, file name before, the pair of images were retrieved. These images were internally represented as rgb model, so there was the need to convert them to HSV, by means of the open cv library and, subsequentely, isolate the h channel from the other two. Unfortunately, the open cv library, as written in the documentation, divides by two the original hue values, in order to fit them in 8 bits. This could represent a loss in terms of information, this is the reason why I applied a further operation, after conversion, in order to represent hue with its original values, exploting a wider 16 bits representation. After doing this conversion, the two arrays determined were subtracted, so that the final array would have contained only the hue differences for each single pixel. Since all images were different, there was the need to determine a threshold value, on the base of that applying a filtering operation. Finding a threshold meant determing the hue differences composing the burnt area. The only way to know, a-priori, where to look

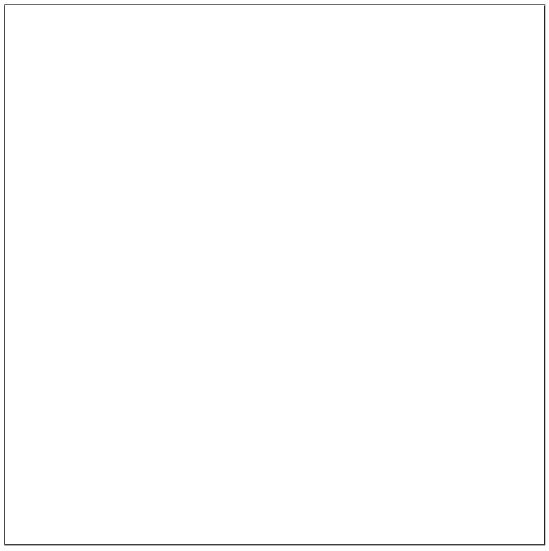
for that values, was exploiting the manual annotation provided by the Effis service. Even though, as explained previously, the annotation didn't match properly the hotspot area for the images acquired from the Sentinel Hub service, nevertheless it surrounded a great portion of the burnt spots, making it a good starting point for analysing the hue values of interest. Again, this was a method to get faster results, without worrying too much about manual cleaning of images, thus, it was absolutely admitted some noise due to a not-very-precise threshold acquisition, since a further cleaning step was then applied. So, in order to get a better chance to look in the right burnt area, instead of analysing the entire portion of image surrounded by the polygon annotation, just a smaller bounding box within it was used. This meant isolating the portion of the numpy array, representing the hue differences, taking a rectangular inner portion, related to the burnt area. The approach I decided to use is pretty conservative, supposing to define two thresholds instead of a single one. The first threshold is the **min_th**, thus the minimum threshold, which was determined computing the average hue value of the burnt area divided by two, while the maximum threshold, **max_th**, was determined adding to the maximum value the difference between the maximum value and the average value, divided by four. These formulas were used in order to preserve as much information as possible, relying on further cleaning operations for eliminating general noise. As soon as the minimum and maximum threshold were determined, the next step was represented by retrieving the binary mask. This step was composed of two inner steps, the first one determing the binary mask out of the minimum threshold while, the second one, complementarily, the binary mask out of the maximum threshold. Figure 4.4 reports the obtained results.

As it is immediately noticeable, the maximum threshold mask is white, this means that all the hue difference values are below the maximum threshold. So,in this case, the logical and bit a bit between the two images can be considered redundant, since the final mask will be equal to the input minimum threshold mask. Those masks correspond to the areas showed in 4.5.

The resulting mask is still noisy, this is absolutely normal, since no further operations were applied. This additional operations will be showed right below and they are part of the following steps, in order to obtain the final cleaned mask. In fact, after obtaining the result above as a consequence of the logical 'and bit a bit' operation of the two threshold outputs described above, the following operations applied were erosion and dilation. Erosions and dilations represent two of the most used morphological image processing techniques, they rely on a small matrix called kernel, also known as **structuring element**, which is applied iteratively all over the mask, by sliding it at every step of a given stride. The kernel, whose dimension can change significantly, depending on the size of the input image, is filled with values



(a) Contrada Acquavite, Sicily(TP). Min threshold mask



(b) Contrada Acquavite, Sicily(TP. Max threshold mask

Figure 4.4: Thresholding

(a) Contrada Acquavite, Sicily(TP) - after fire event

(b) Contrada Acquavite, Sicily(TP) - before fire event

Figure 4.5: True color images

equal to one, in this example; at every step the matrix is slided, for each single value of the input image as a moving window. For what concerns the former: if the matrix

is completely contained within the covered area of the input image, thus, if the input image contains one all over the checked area (if it fits the covered area), the value in the center is mantained to one, otherwise it is turned to zero. In practice, this mechanism will eliminate small noise all over the image, even though this process depends on the size of the kernel, the bigger the sliding window, the more of the image will be eliminated. At the same time it will soften borders, reducing size of bigger spots within the image. This process allows to get a cleaner overall output, but it woul not be so useful without the corresponding dilation process. Dilation behaves in a complementary way. Again, given a kernel of a specific size, it is slided through the entire input image; if the kernel **hits** the image, thus, if at least one element covered is equal to one, the value at the center is set to one, otherwise it is mantained to zero. In other terms, this means that all the borders, reduced by the erosion operation, will be almost restored to the previous state while, and this is the key point, the eliminated noise won't be back, since its corresponding area values were set to zero. Summarising, the consecutive application of this two operations, results in a cleaner image whose borders are still pretty similar to the input image. Obviously, the final results are strictly related to what parameters have been chosen both in erosion and dilation. Here it is presented a small example:

erosion

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.1}$$

dilation

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{4.2}$$

For what concerns satelllite imagery analyis, referring to the image previously showed, here is the mask after applying erosion and dilation, specifically in the following order: 20x20 erosion followed by a 5x5 dilation, after some experiments:

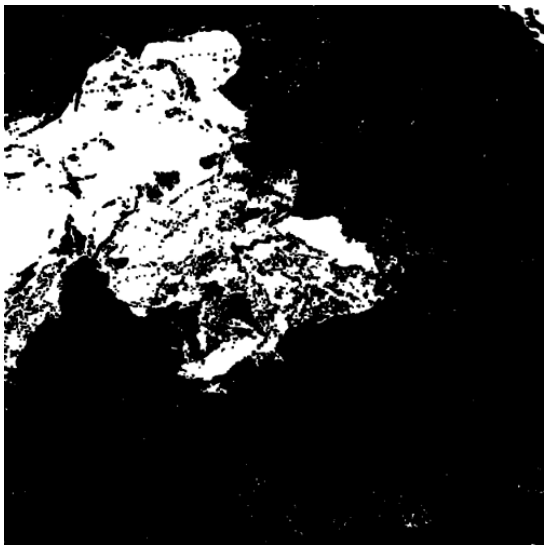


Figure 4.6: Contrada Acquavite, Sicily(TP) - after erosion and dilation

It is definetely cleaner than the previous example; some borders have been eroded and there is still some noise, but this was partially removed using a further technique, this time relying on the **findContours** function provided by the opencv library. As it may be understood from the name this function goal is retrieving all the borders of all the shapes within an image. The basic idea was simple: whenever there's additional noise, its boundaries are typically small, so finding the length of all contours and not drawing the smaller ones was the key to apply an additional noise reduction. But, what I wanted to do, was preserving inner borders, so shapes within shapes, representing inner unburnt spots or streets, in order to have an higher accuracy and accomplishing this task the main key was splitting borders in two categories: external and internal borders. There was a parameter, which could be specified in the findContours function in order to filter out the kind of contours to be retrieved; so the first retrieving was related to outer contours, by means of the **RETR_EXTERNAL** flag. This contours were then filtered exploiting the inner area contained within them, the contours containg small areas were filtered out, while the others were kept. Starting from this list, it was possible to draw as white all the areas within those contours, keeping black the background. Then, the following operation, was retrieving the inner contours, by means of the **RETR_CCOMP** flag and create a negative mask, thus a mask where inner areas, the ones contained by the inner contours, were totally black, while the external were white. Then, the last operation was applying a logical bit a bit between the two masks in order to obtain the final one 4.7.

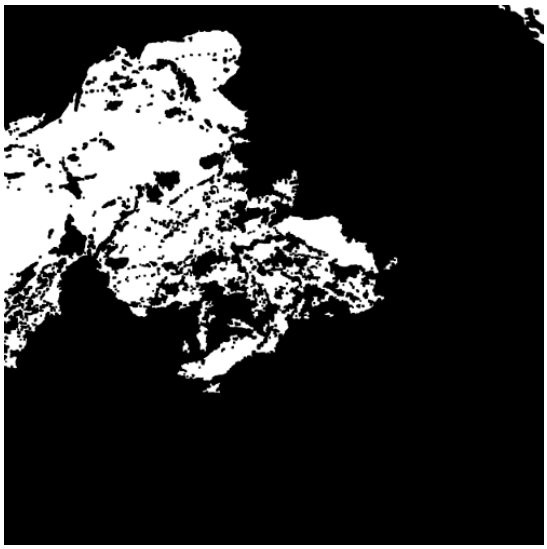
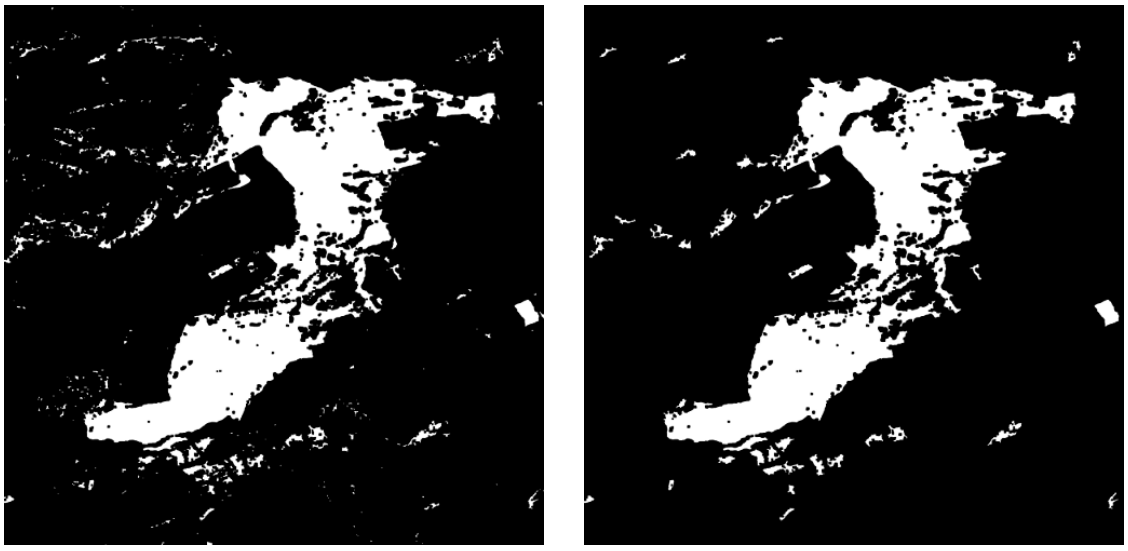


Figure 4.7: Contrada Acquavite, Sicily(TP) - after findContours()

In figure: 4.9 as as summary, it is presented the before, after, mask before findContours operation and the final mask of another fire:

(a) Poggioreale (TP) - after fire

(b) Poggioreale (TP) - before fire



(a) Poggioreale, Sicily(TP) - after erosion and dilation

(b) Poggioreale, Sicily(TP) - after findContours()

Figure 4.9: Masking images process

The last image shows the final mask, it still has some noise all around, which can be easily remove by exploiting a cleaning operation.

4.2.4 Dataset creation

In order to feed the machine learning model, it was crucial to define a way to store the amount of information retrieved, in terms of both images and masks in a format which was suitable for the specific purpose. There were a number of alternatives, such as exploiting a remote database and cloud storage, or save all the images locally and then apply a specific algorithm in order to feed the neural network. However this solution were expensive for different reasons, such as download speed, the need of an internet connection, or a huge amount of storage available for all the images and the corresponding masks. This is the reason why the chosen option was building a portable database, in a binary format, holding all the images and the corresponding masks, following a specific order of bytes. This allowed not only to have a easy to use solution, but also a very compact one, in terms of memory footprint, which contained everything needed in just one place. The algorithm is based on the fetching of the images representing the burnt areas and the corresponding masks. A single image is saved as a sequence of rgb values, storing for each of these elements one single byte. So, supposing to save only the rgb values and the corresponding element of the binary mask, this meant storing four bytes for each single pixel of the image. It

is useful to remember that red green and blue were coded in true color 24 bits, thus 8 bits for each single channel; this means no loss of information after saving. There were different approaches for this problem, since I exploited different techinques for images analysis. The first idea was the so-called **windowed approach**: it was based on splitting the source image, with the native resolution cropped to 4960x4960 pixels in sub-images of different dimensions. Specifically, these windows were chosen for experiments purposes, and the different dimensions were actually used in order to do further analysis, as it will be explained in the next chapter. Since the number of images was important and looping through a numpy array of 4960x4960 was computationally intense, I decided to exploit a multithreaded approach, where each single thread was involved in the computation of a single sub window. Each result was then properly embedded in a message sent to a queue emptied by a consumer thread, saving each single message, appending its content to the binary file. This approach determined that each single window was saved one after another. This is a simple schema representing the windowed approach:

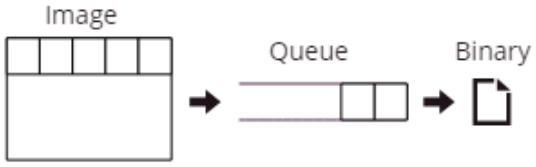


Figure 4.10: Windowed dataset creator schema

The other approach was saving the entire source image, without applying any windowing. Unfortunately, for the following step, a size of 5000x5000 pixel was too big and that would have taken too much time in order to converge, besides the computationally intensive tasks it would have determined. So, this was the reason why, before saving the images and the corresponding masks, a resize down to 496*496 pixels was necessary. So, this time, the multithreaded approach was not applied to each subwindow of the original tile, but, instead, to a single image, which was then embedded in a single message and sent to the consumer. Obviously, the final dataset obtained with the latter approach was definetely smaller than the ones obtained with the windowing approach.

4.2.5 Convolutional Neural Networks - Unet

As mentioned in the first chapter, convolutional neural networks often represent a valid option when dealing with input data as images. This was one of the reasons why it looked like a good option to start with. As explained, convolutional networks are

able not only to find recurrent patterns in the input data, but also infer many features at deeper levels of the chain. Plus, it was necessary, looking at the application field, to apply a segmentation over the image in order to find, if present, the boundaries of the burnt areas, exploiting a supervised learning approach, thanks to the binary masks computed in the previous chapter. The specific network exploited was the unet, a specific cnn highly exploited in the last years for segmentation purposes. The main goal was achieving a pixelwise classification, by means of a network which was able to output, for each single pixel, a value referring either to a burnt or an unburnt spot. So, in other terms, the final output segmentation had to be in a binary mode, thus, white areas for burnt areas and black areas for the unburnt one. Here, before explaining its inner layers and the overall architecture, it is presented the unet model:

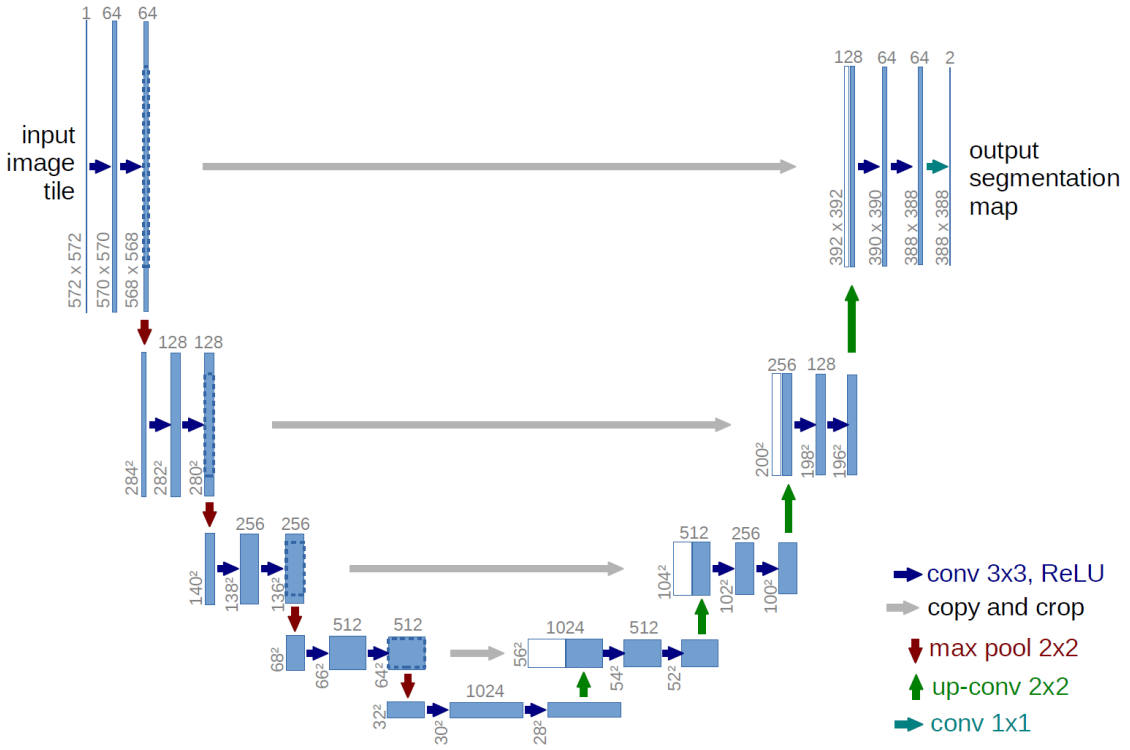


Figure 4.11: Unet for biomedical segmentation, [35]

This model was specifically used for biomedical segmentation [35]. Even though it was a different analysis from mine, the basic idea could be considered similar. There was the need not only for a mere classification, but also a localization of pixels in the image. Not only the model showed behaved brilliantly in the following tests, but it could be able to achieve those results with a very small training set, thanks to

data augmentation through elastic distortion. The network is made of several layers, as explained before the left part of it represents the contraction path, the one whose goal is reducing the overall spatial resolution of the input image and, at the same time, increase the number of features extracted by the model. Complementarily, the expansion path, will increase the spatial dimensions and reducing the depth of the computed volumes in order to retrieve the final output image. erring to the example, which is pretty similar to what I used for my field of analysis, it is composed of a recurent structure which can be resumed with two consecutive convolutional operations, followed by a max pooling layers. The pattern is increasing the number of features at each step of the network, while the max pooling layer halves the spatial dimensions. Starting from the first layer, the system will be able to detect smaller features of the input image, such as small shapes, corners; while the deeper the network goes, the more features it will be able to extract, looking at bigger portions of the image. The expansion path, complementarily, doubles up the spatial dimension at every up-convolution and the following convolutions will progressively reduce the number of filters, until obtaining a proper depth. The last operation, which outputs the result, computes a sigmoid function, which constraints each pixel value in a continuos range from 0 to 1. Thus, the model, applies a regression which is then converted into a classification by means of a threshold applied at 0.5. The more an output value is closer to one, the more chances that pixel is related to a burnt pixel and vice-versa. The networks I managed to create for the purposes of my research followed the basic structure presented in this paper, although it presents some minor changes in order to avoid dimensionality reduction, due to the absence of any padding strategy in the work above and some changes to the layers, so that the model could work with different inputs, as it will be explained in details in the experiments chapter.

Chapter 5

Experiments

The experiments showed in this chapter are related to two main parts of the project: image processing and machine learning. For the former it is reported the different operations applied on the images until achieving a satisfying result in terms of the obtained binary mask. The latter shows the different steps that were taken in order to create the final machine learning model, capable of detecting burnt areas without the presence of a 'before" version of the image containing the burnt area.

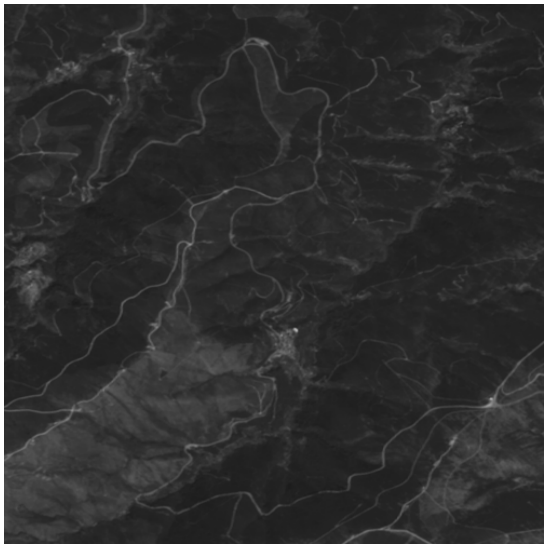
5.1 Image Processing

The image processing step, as deeply explained in the previous chapter, was intended as a way to determine the burnt area, in order to define a precise annotation for its boundaries. So, many experiments were carried out trying to figure out a way to achieve this important goal. The inputs were, at this point, represented by a pair of images for each single fire event. This pair of images represented a **before** and an after version of the area of interest. Here below there's another example of a before/after event:

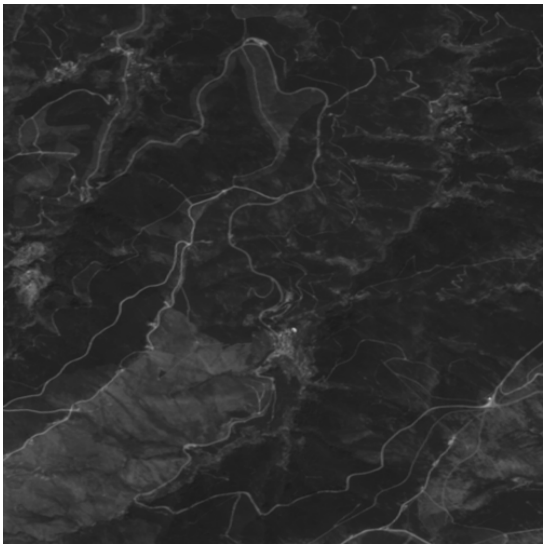
The burnt area is located in the center of the left image, in the darker area. At a first glance its shades are made of gray and brown. The first approach was pretty straightforward: supposing that two images were the same, except for the burnt area, a mere difference between the two would have created a final mask where the unchanged color components would have been equal or close to zero, while the hotspot area, since different, would have showed different shades. This problem was faced using two different approaches: the first one was working with grayscale images, since this would have facilitated the difference process, having to deal with just one single channel per image. So, the first operation was converting before and after images into grayscale.

(a) Oliveira do Hospital, Portugal - after fire event

(b) Oliveira do Hospital, Portugal - before fire event



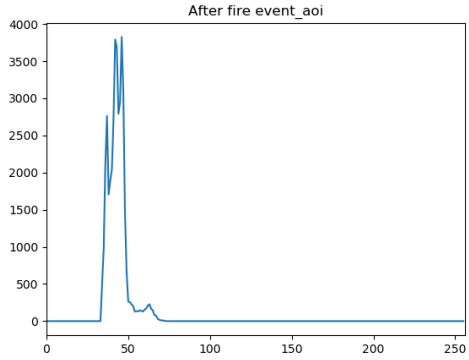
(a) Oliveira do Hospital, Portugal. Grayscale - after fire event



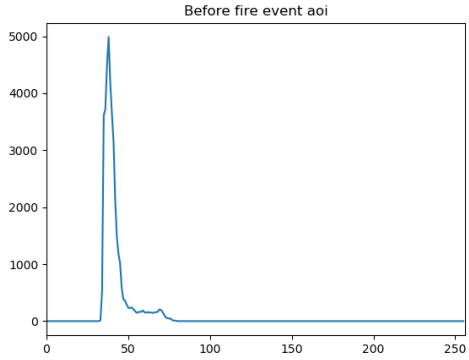
(b) Oliveira do Hospital, Portugal. Grayscale - before fire event

The two images look very similar, much more than the previous case. The main issue is that, working with grayscale, the chrominance information was lost, while the luminance component was the one kept. This meant that, trying to identify the area of interest with the grayscale approach would not have been very useful, since the elimination of the color component would have flattened the image information, losing the ”burnt” spot. It is pretty clear that the areas related to the fire event are

really close in terms of grayscale levels. In fact, this is confirmed by analysing the histograms of the same cropped region of both images:

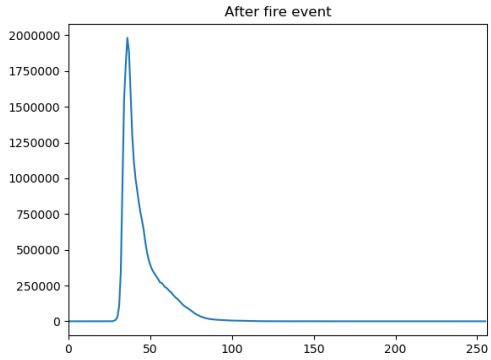


(a) Oliveira do Hospital, Portugal. Histogram grayscale after event aoi

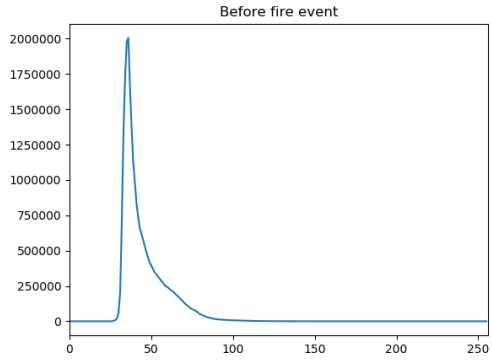


(b) Oliveira do Hospital, Portugal. Histogaram grayscale before event aoi

The two figures cleary show what it was mentioned before. The chroma information was lost due to the conversion process and the luminance information is not enough in order to distinguish the two areas, whose histogram is really close; thus, the difference between the two images would have been a shades of gray really close to 0 level, thus zero. The situation is pretty similar for the entire images:



(a) Oliveira do Hospital, Portugal. Histogram grayscale after event aoi



(b) Oliveira do Hospital, Portugal. Histogaram grayscale before event

The components are contained within a very narrow range, the same range that contains the components of the area of interest. This means that the useful information was lost during the conversion process. It is also pretty obvious that the difference between the two gray images would have resulted in an almost-black output mask, not showing the desired hotspot. Here, it is shown the difference:

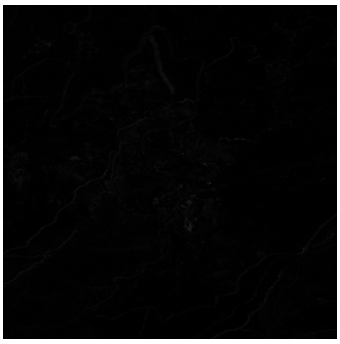
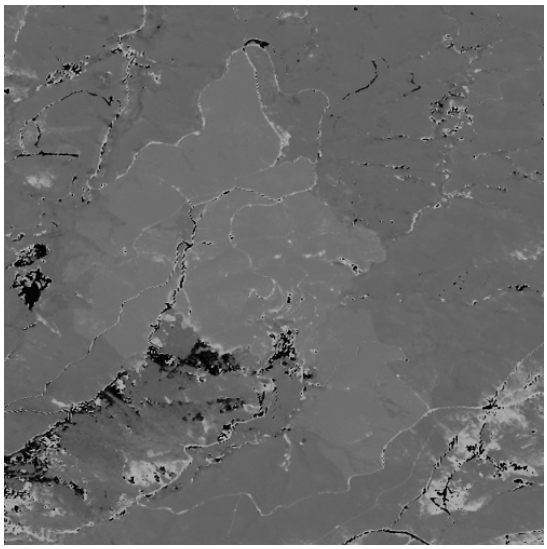


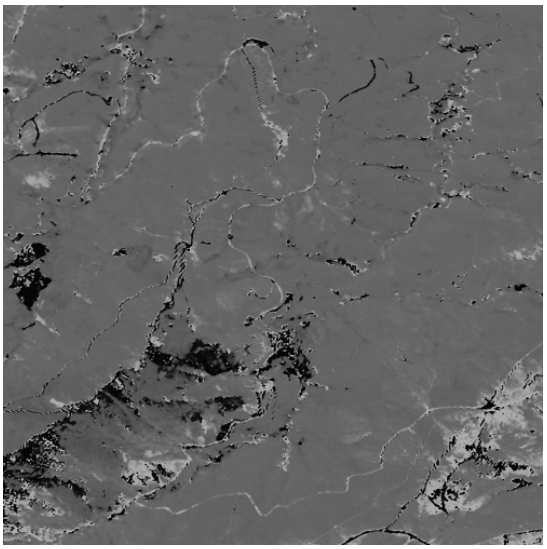
Figure 5.5: Oliveira do Hospital, Portugal. Histogram difference

Summarising, this first algorithm was not useful for the preset goal. That's why the following operation was trying to apply a different approach, trying to convert images into a different color model, which could split the chrominance from the luminance, allowing to analyse images from different perspectives. The picked model was HSV. The first operation was splitting each image into the three components: hue, saturation and value. Then, applying a separate analysis for each different channel.

hue

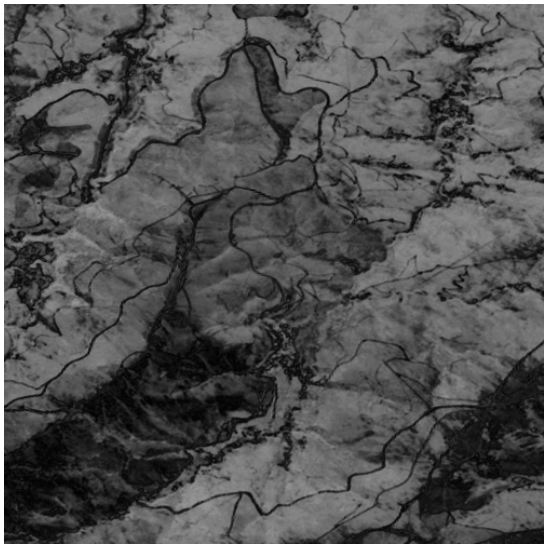


(a) Oliveira do Hospital, Portugal. Hue component. After event.

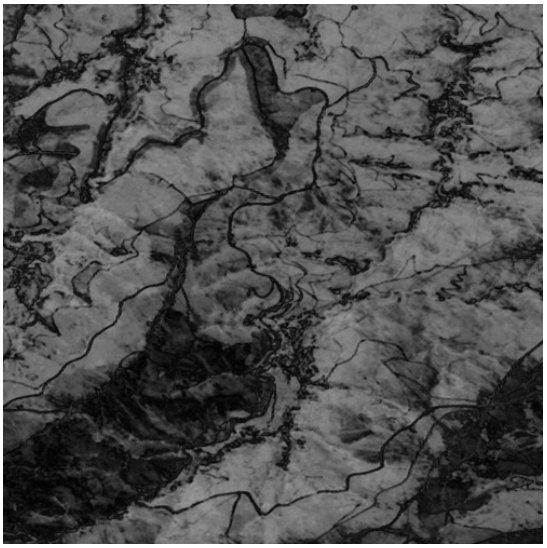


(b) Oliveira do Hospital, Portugal. Hue component. Before event.

saturation

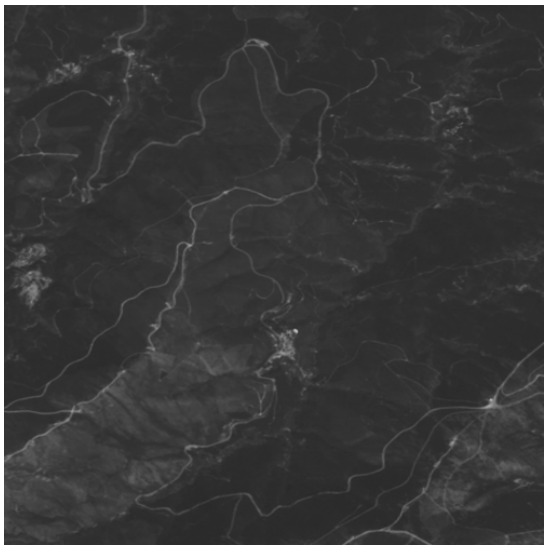


(a) Oliveira do Hospital, Portugal. Saturation component. After event.

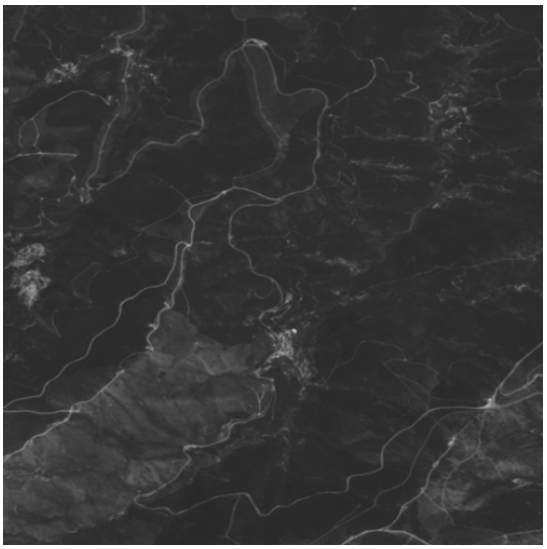


(b) Oliveira do Hospital, Portugal. Saturation component. Before event.

value



(a) Oliveira do Hospital, Portugal. Value component. After event.



(b) Oliveira do Hospital, Portugal. Value component. Before event.

It is pretty clear that the saturation channel and the value channel do not bring any additional information to the analysis, since those images look pretty similar

in both versions. What, instead, shows additional information is the hue channel. Specifically, the after version shows a clear spot, that is represented with a different shade from all the other colors surrounding it. That area corresponds to the hotspot, or the so called area of interest. Thus, it was interesting going deeper in order to figure out if it could represent a possible solution to this problem. The following step was applying the difference between the two hue channels of both after and before images and look at the outcome:

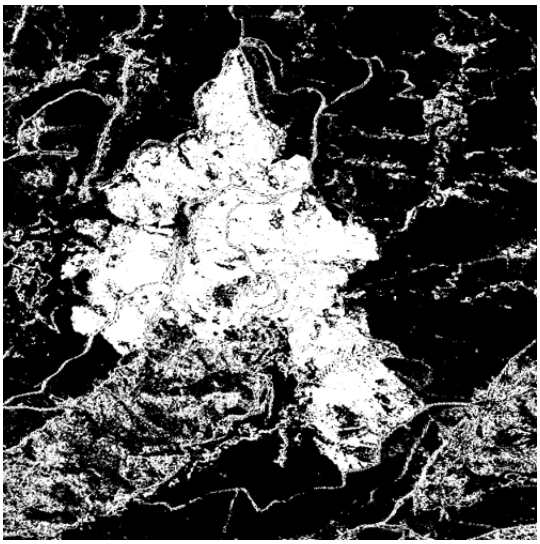


Figure 5.9: Oliveira do Hospital, Portugal. Hue difference mask

The result above shows the difference between the two hue outcomes, applying a mask which *turned on* the pixels whose hue values were above a certain threshold and *turned off*, complementarily, the pixel whose hue values were below the given threshold, properly computed as described in the previous chapter. This result was also achieved rescaling the hue values to 0-360 range, from the 0-180 provided by opencv. One more thing, which has to be highlighted, is that the absolute difference between two hue points was considered as the smallest difference on the circumference, so the smallest gap, between the two determined. As mentioned in the previous chapter, the mask had to undergo further cleaning operations in order to obtain the final result, applying, in the order, dilation and erosions plus the find contours operation.

Although the process determins a faster approach for manual annotation, it requires, most of the times, an additional cleaning operation to get the best outcome possible. This is the reason why, for the purposes of the analysis, I decided to get the final annotation of two hundred different burnt areas, instead of working on all the

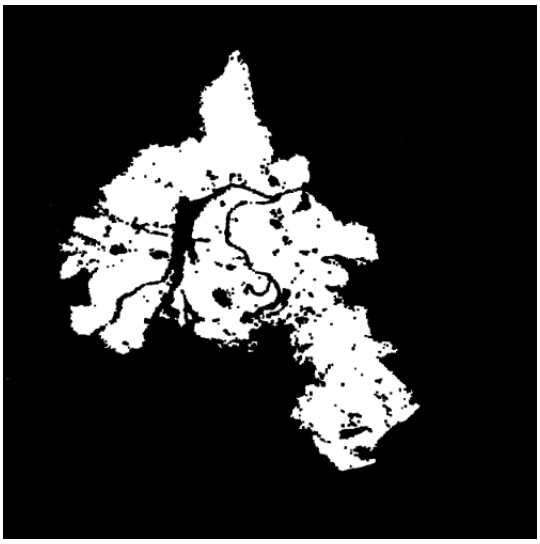


Figure 5.10: Oliveira do Hospital, Portugal. Final mask

acquired images. Obviously, being able to rely on a bigger dataset means having a more comprehensive vision of various burnt areas all over the world, but this is easily fixable with further *manual* work. 5.11 represents a subset of annotated images:

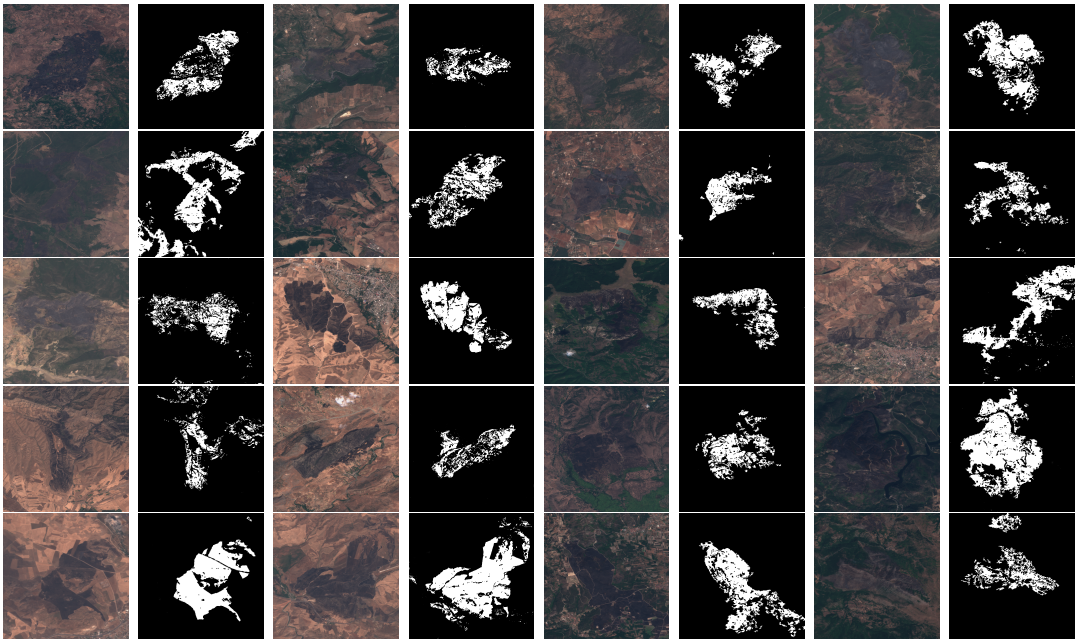
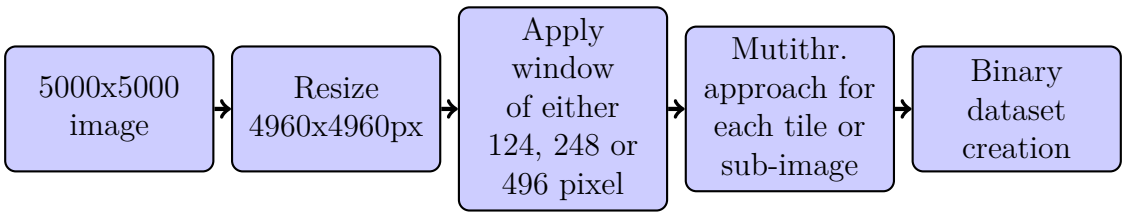


Figure 5.11: Mosaic of burnt areas

5.2 Machine Learning

This part relies on all the techniques exploited and the various experiments whose goal was finding the most suitable solution in order to determine the boundaries of the burnt area. The first problem was represented by the choice of the algorithm to use to classify the various images of the dataset. Despite the possibility to apply different tests and techniques in order to figure out what machine learning algorithm to pick, I choosed immediately the neural networks solution. The reason of this choice was mainly related to literature experiments. It was showed during the last years that CNNs represent the state-of-art when working with image classification. The starting conditions, for image classification, was composed of two sets of images: satellite imagery reporting burnt areas and their corresponding annotations, thus, binary masks, which for the sake of this analysis, could be considered as the ground-truth. The first decision was how treating each single image, because of its size. As mentioned in the previous chapter, each single burnt area was stored as 5000x5000 pixels image. Thus, it would have been pretty difficult, in terms of computation to handle such a big file, in terms of dataset size and machine learning computations. Thus, the final decision was applying two different approaches: a **windowed approach** and a **resized approach**. The former was based on the splitting of each single input image into several tiles, whose number was influenced by the size of each single sub-window while the latter was based on a initial rescaling operation of the image, without any cropping operation. For the former, the **windowed** approach, I decided to exploit three different window sizes, in order to compare the performances of the model applied on vignettes of different dimensions. This sizes were respectively: **124**, **248** and **496** pixels. Here below, it is showed the workflow for those three sizes:



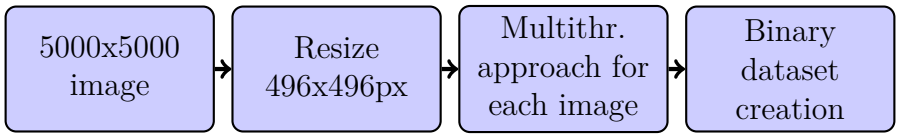
It is obvious that, supposing to have 200 images for the dataset of all three windowing approaches, the number of sub-tiles, obtained as output, was different, depending on the particular window size.

Overall dataset dimension tends to be pretty important if working with a windowed approach, especially with the smallest one. The parallel approach was, instead of splitting each single image into sub-tiles, working on each different tile as-is,

Window Size (pixel)	Tiles per Image	Total no. tiles
124	1600	320000
248	400	80000
496	100	20000

Table 5.1: Dataset for each window size

resizing the original image and work on that without doing any cropping operation. Here it is presented the workflow for this particular operation:



What is is worth noticing is that the output of both approaches was a binary file, where each image and the corresponding binary mask were encoded in a specific format: for every pixel of the true color image, representing the burnt area, its red, green and blue values were encoded in 1 byte, plus 1 byte for the corresponding pixel mask, for a total of 4 bytes for each single pixel. Thus, again, supposing a dataset of 200 images, the size for each output dataset is reported below:

Algorithm	Size per tile (KB)	Size dataset (GB)
Windowed 124 px	≈ 60	≈ 18.32
Windowed 248 px	≈ 240	≈ 18.32
Windowed 496 px	≈ 961	≈ 18.32
Resized 496 px	≈ 961	≈ 0.18

Table 5.2: Size for different algorithms

Obviously, the smaller the window, the smaller each tile size, but, what it's clear, is that the overall dataset size did not change for all different windowed approaches. The resized algorithm, on the other hand, lead to a definetely smaller dataset because of a 10 times reduction operation.

5.2.1 Resized Algorithm

The resized algorithm, as explained previously, took as input a 5000x5000 pixels image and produced, as output, its transformation obtaining a definetely smaller 496x496 pixels image, without any windowing whatsoever. The reason why it was

necessary to implement such an important resizing, is because of the memory footprint produced by the model during the training process, which is very important because of the many layers the CNN was composed of.

Resized Algorithm - Training

Architecture

Operating System	CPU	Ram	GPUs
Linux Ubuntu	Intel Core i9-7940x @ 3.10GHz	126GB	NvidiaGeForce 1080 TI x 4

Table 5.3: Hardware resources

As explained in the previous chapters, the architecture used for this work was a modified version of the unet showed previously. It is composed of two paths: a contraction path and an expansion one. Each single level is composed of two convolutional operations, which, in the contraction path, followed by a max pooling layer resizing the spatial dimensions to half of the input volume. The expansion path progressively increases the size of the spatial dimensions, reducing the depth of the volume, until obtaining a regression mask of a depth equal to 1 whose spatial size is equal to the input volume.

Architecture

Layer	Input Volume	Output Volume
Conv. Layer_1	496x496x3	496x496x32
Conv. Layer_2	496x496x32	496x496x64
Max Pool. Layer_1	496x496x64	248x248x64
Conv. Layer_3	248x248x64	248x248x64
Conv.Layer_4	248x248x128	248x248x128
Max Pool. Layer_2	248x248x128	124x124x128
Conv.Layer_5	124x124x128	124x124x128
Conv.Layer_6	124x124x128	124x124x256
Max Pool. Layer_3	124x124x256	62x62x256
Conv.Layer_7	62x62x256	62x62x256
Conv.Layer_8	62x62x512	62x62x512
Max Pool. Layer_4	62x62x512	31x31x512
Conv.Layer_9	31x31x512	31x31x512
Conv.Layer_10	31x31x512	31x31x1024

Table 5.4: Contraction Path

Layer	Input Volume	Output Volume
Up_1	31x31x1024	62x62x1024
Conv. Layer_11	62x62x1024	62x62x512
Conv. Layer_12	62x62x512	62x62x512
Up_2	62x62x512	124x124x512
Conv.Layer_13	124x124x512	124x124x256
Conv.Layer_14	124x124x256	124x124x256
Up_3	124x124x256	248x248x256
Conv.Layer_15	248x248x256	248x248x128
Conv.Layer_16	248x248x128	248x248x128
Up_4	248x248x128	496x496x128
Conv.Layer_17	496x496x128	496x496x64
Conv.Layer_18	496x496x64	496x496x1

Table 5.5: Expansion Path

Up is defined as a layer which combines three following operations, in the order: **upsampling** which doubles up the spatial dimension of the input volume, a **convolution** and a concatenation between the obtained result of the output and the corresponding volume of the contraction path. The output of the entire model is a

mask containing a regression, between 0 and 1, for each single pixel. The closer the outcome, defined by a sigmoid function, is to 1, the more likely the pixel belongs to a burnt spot, otherwise, complementarily, it will be related to an unburnt area.

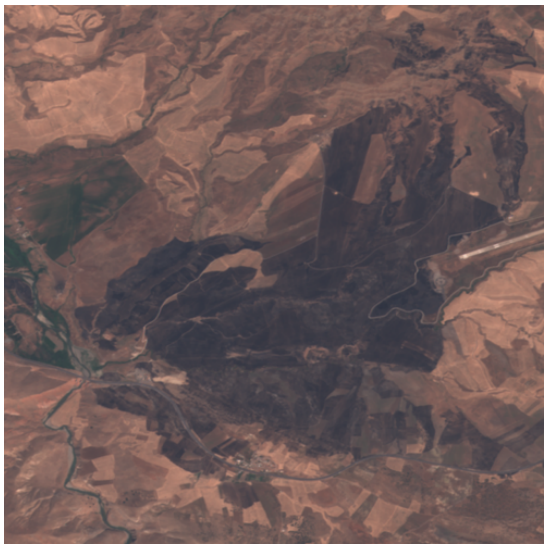
Data augmentation

Data augmentation, as explained in the previous chapters, is a particular technique whose goal is applying specific transformations to the input images in order to increase significantly the dataset. There are two main goals for this technique: the first one is avoiding overfitting offering a variety of images as input to the learning model, the other one is increasing the input dataset so that there are more images to learn from. The dataset size chosen for the experiments was not that big because of two reasons: the former one was avoiding a manual approach for the annotations which would have been excessively time consuming, the latter was related to the time it would have taken for the model to be trained. The last issue was especially important if working with a windowed approach since the size of the original image, thus, before applying a splitting into subtiles, was equal to 4960*4960 pixels, with no further resizing.The time taken by the model to be trained on that huge amount of pixels was very important, thus, relying on a wider dataset would have taken too long for all the experiments to be successfully done. So, by chosing a smaller dataset, I was able to compare different techniques during the training process. Data augmentation was based on a series of transformations illustrated below:

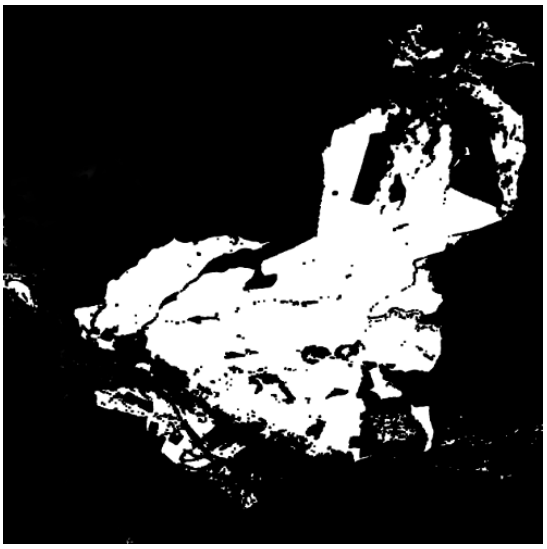
Transformation	**Value**
shear range	0.5
rotation range	50
zoom range	0.2
width shift range	0.2

Table 5.6: Augmentation algorithm transformations

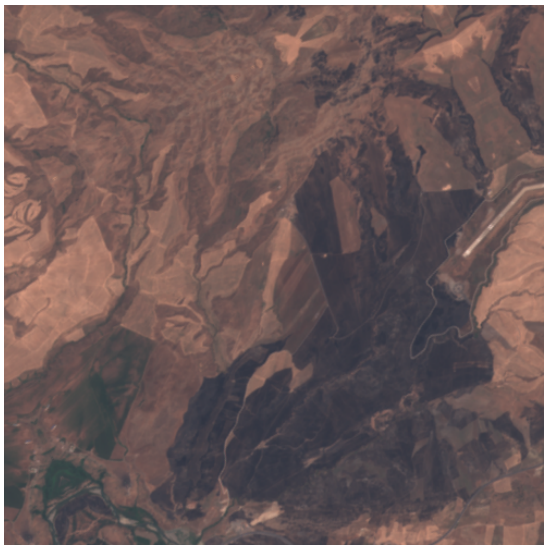
The algorithm acquired information from the given dataset and continuously applied random transformations over each single batch. This meant having the possibility to extent significantly the feeding information, letting the network learn on a wide variety of shapes, very important when dealing with burnt areas, which are segments of ground that do not belong to any specific shape or color. Figure 5.12 and 5.13 show the examples of some images transformations:



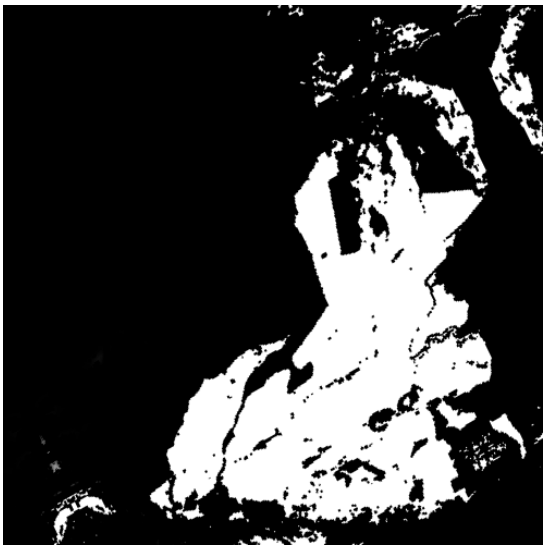
(a) Not augmented



(b) Not augmented mask



(c) Augmented



(d) Augmented mask

Figure 5.12: Augmentation example

(a) Not augmented

(b) Not augmented mask

(c) Augmented

(d) Augmented mask

Figure 5.13: Augmentation example

Above are showed two possible transformations applied over the original image and its corresponding mask; what it is worth noticing is that those transformations are the same for the image and its corresponding mask, so that the burnt area will still correspond to the white parts of the mask and vice-versa.

5.2.2 Training with data augmentation

Training set	Validation set	Batch size
1600	200	5

Table 5.7: Training Parameters

No. steps per epoch	No. validation steps	No. epochs
320	40	13

Table 5.8: Training Parameters

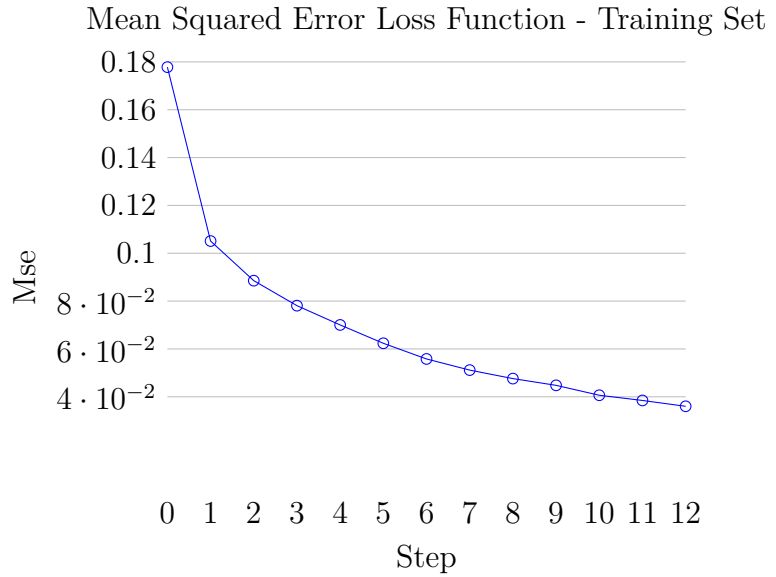
Loss Function *Mean Squared Error* $\sum_{i=1}^{n} \frac{(x_i - \hat{x_i})^2}{n}$

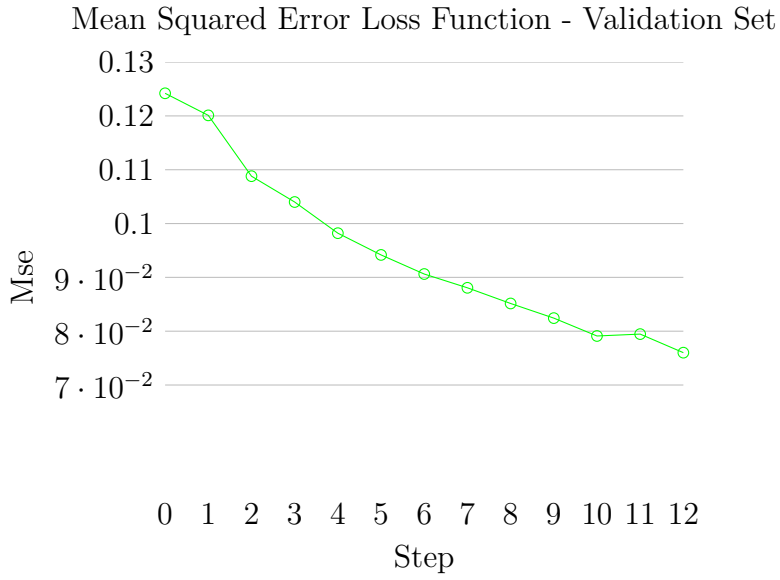
Optimizer *Adam* with *learning rate*: 0.00001

The training set size of 1600 was obtained by applying augmentation to the starting training set of 40 elements. Since the number of steps per epoch was set equal to 320, with a batch size of 5, the number of samples *seen* by the network at every epoch is equal, clearly, to the training test size. This means that each single image of the training set was brought to the network with 40 different transformations since:

$$sampletransformations = (nostepsepoch * batchsize)/(notaugmenteddataset)$$

The number of augmented validation images was then set to 200, by feeding the network with forty different distortion for each single image from the five of the original validation set. This allowed to kept the same ratio between training and validation set, as the one presented before the augmentation process.

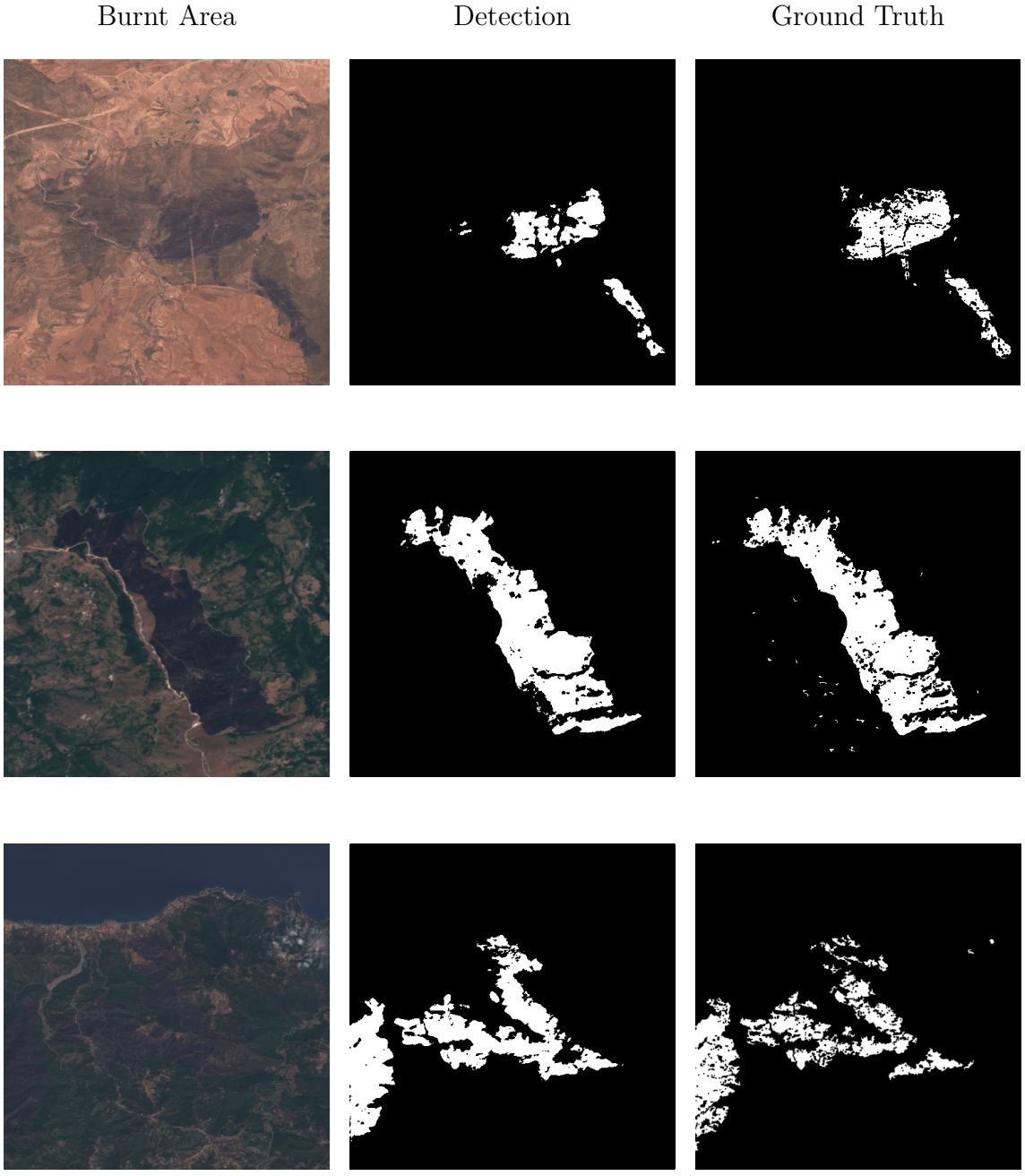


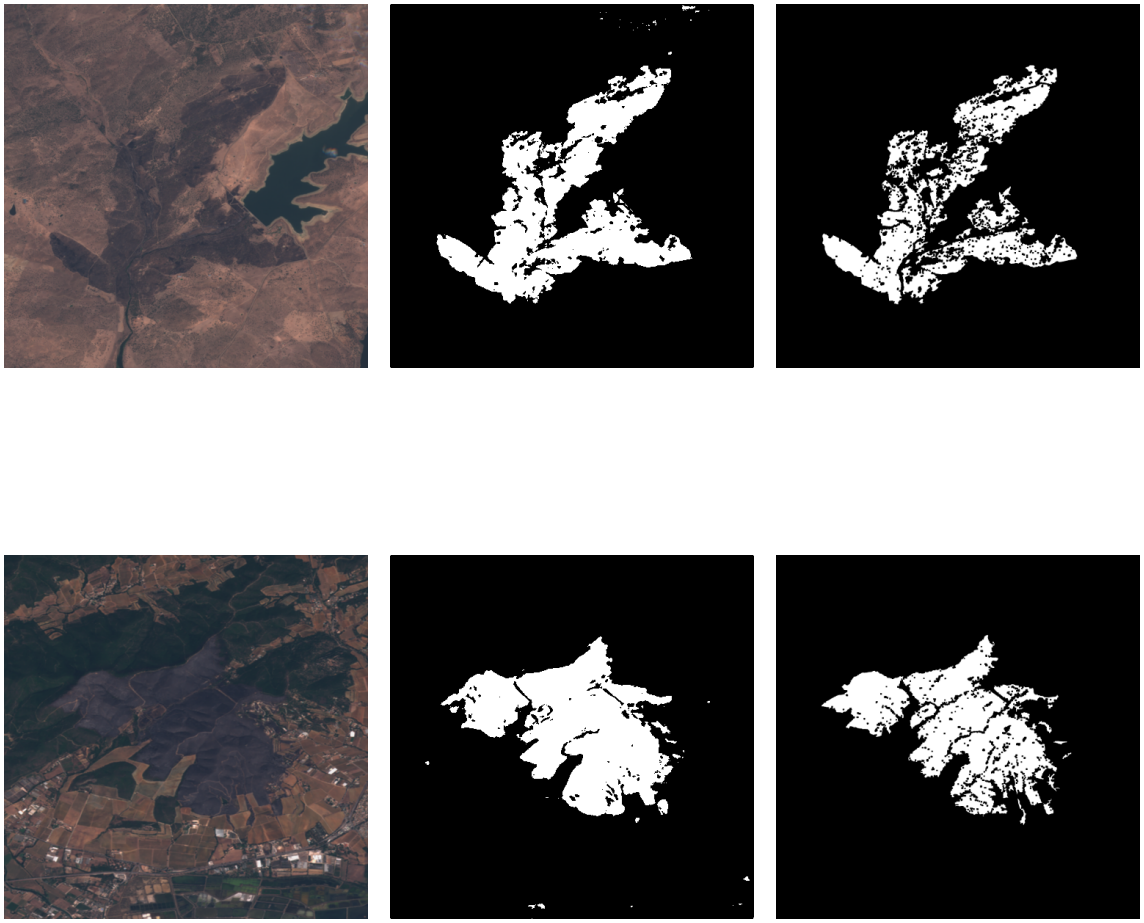


Set	Best Mse
Training	0.036
Validation	0.076

Table 5.9: MSE comparison

Detection Results





Burnt area	Precision	recall	F1 score
1	0.96	0.81	0.87
2	0.91	0.88	0.89
3	0.75	0.84	0.79
4	0.71	0.79	0.75
5	0.84	0.96	0.89

Table 5.10: DICE coefficient comparison

5.2.3 Training with no data augmentation

Training set	Validation set	Batch size
40	5	5

Table 5.11: Training Parameters

No. steps per epoch	No. validation steps	No. epochs
8	1	13

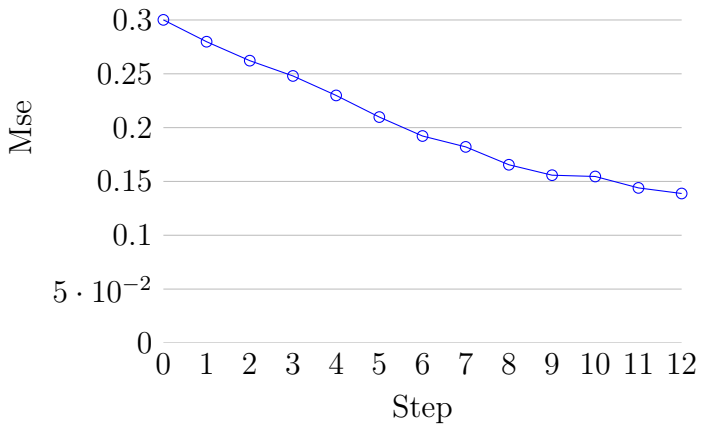
Table 5.12: Training Parameters

Loss Function *Mean Squared Error*

Optimizer *Adam* with *learning rate*: 0.00001

In this experiment it was used the same dataset as before, but with no data augmentation. This meant that the dataset size was restricted to the original number of elements, thus 50: 40 images for the training set, 5 images for the validation set and the remaining 5 images for the test set. The number of steps per epoch was obtained by dividing the original dataset for the batch size, equal to five. The same step was applied for the validation set as well.

Mean Squared Error Loss Function - Training Set

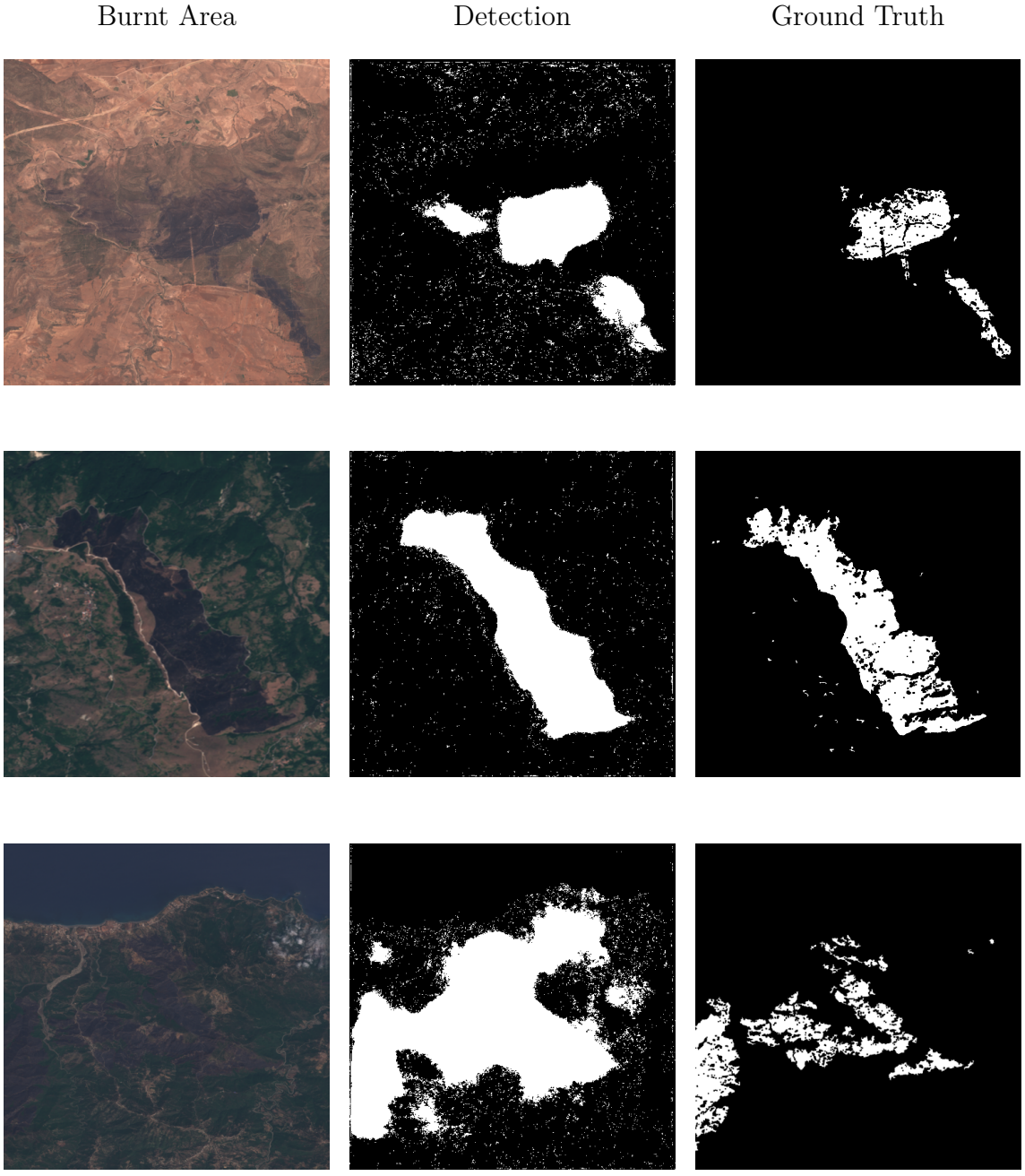


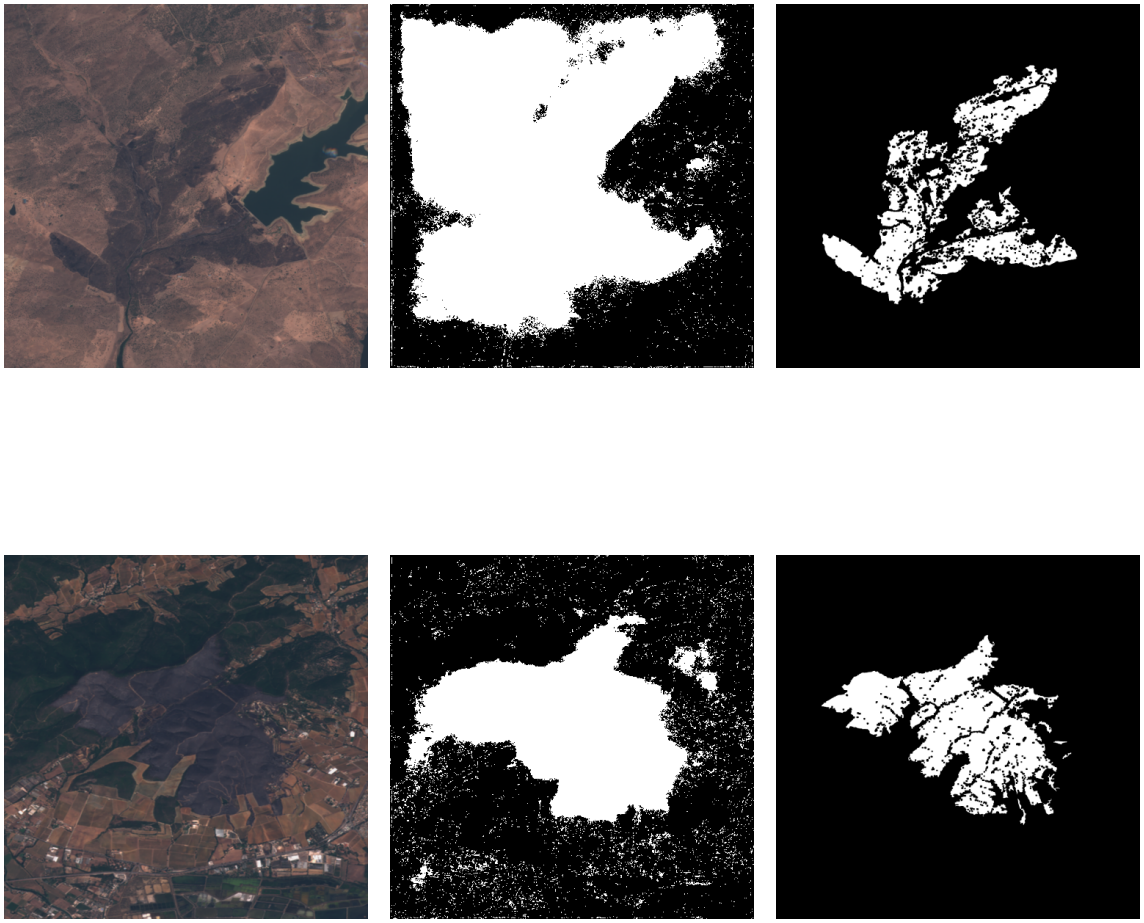
Clearly, the results obtained are worse then the previous case. The interesting thing is that the dataset was the same, except for the augmentation process. This seems to give value to the previous technique, which clearly shows better results in terms of mean squared error. The model was not sufficiently aware of a general context, in order to infer some features which could allow a better result on the validation set.

Set	**Best Mse**
Training	0.1404
Validation	0.1500

Table 5.13: MSE comparison

Prediction Results





Burnt area	Precision	recall	F1 score
1	0.33	0.97	0.49
2	0.70	0.98	0.81
3	0.20	0.97	0.33
4	0.16	0.99	0.26
5	0.40	0.99	0.57

Table 5.14: DICE coefficient comparison

5.2.4 Traning with data augmentation - Bigger dataset

Training set	Validation set	Batch size
7000	800	5

Table 5.15: Training Parameters

No. steps per epoch	No. validation steps	No. epochs
1400	160	9

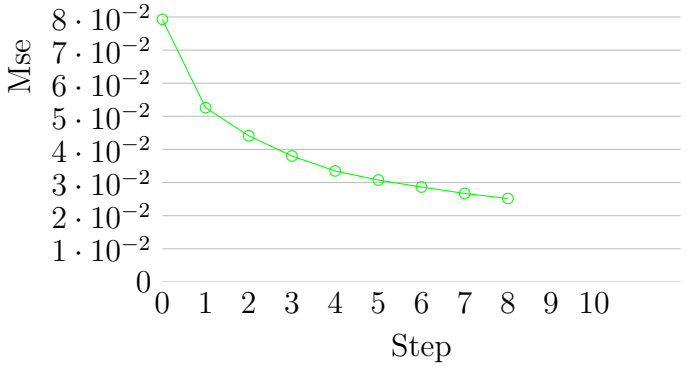
Table 5.16: Training Parameters

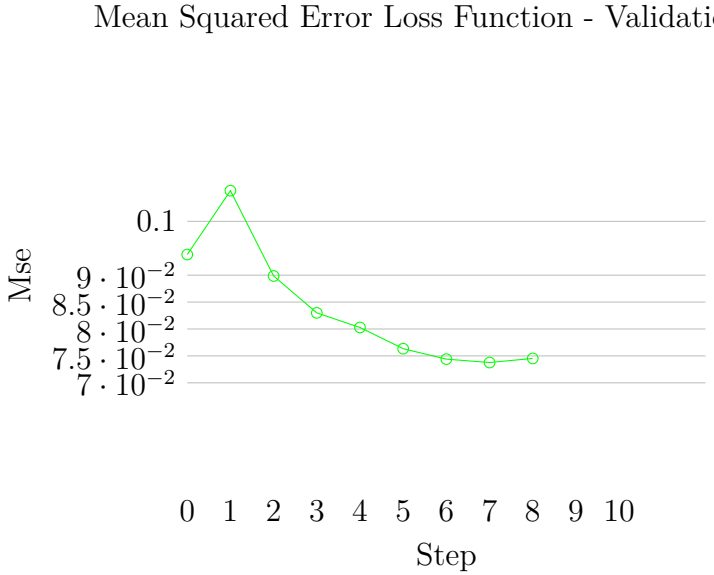
Loss Function *Mean Squared Error*

Optimizer *Adam* with *learning rate*: 0.00001

The training set size of 7000 was obtained by applying augmentation to the starting training set of 175 elements. Since the number of steps per epoch was set equal to 1400, with a batch size of 5, the number of samples *seen* by the network at every epoch is equal, clearly, to the training test size. This means that each single image of the training set was brought to the network with 40 different transformations how already explained for the other data augmentation experiment. The same considerations were applied for the validation set, 20 images augmented forty times each, in order to obtain 800 images as augmented validation set.

Mean Squared Error Loss Function - Training Set

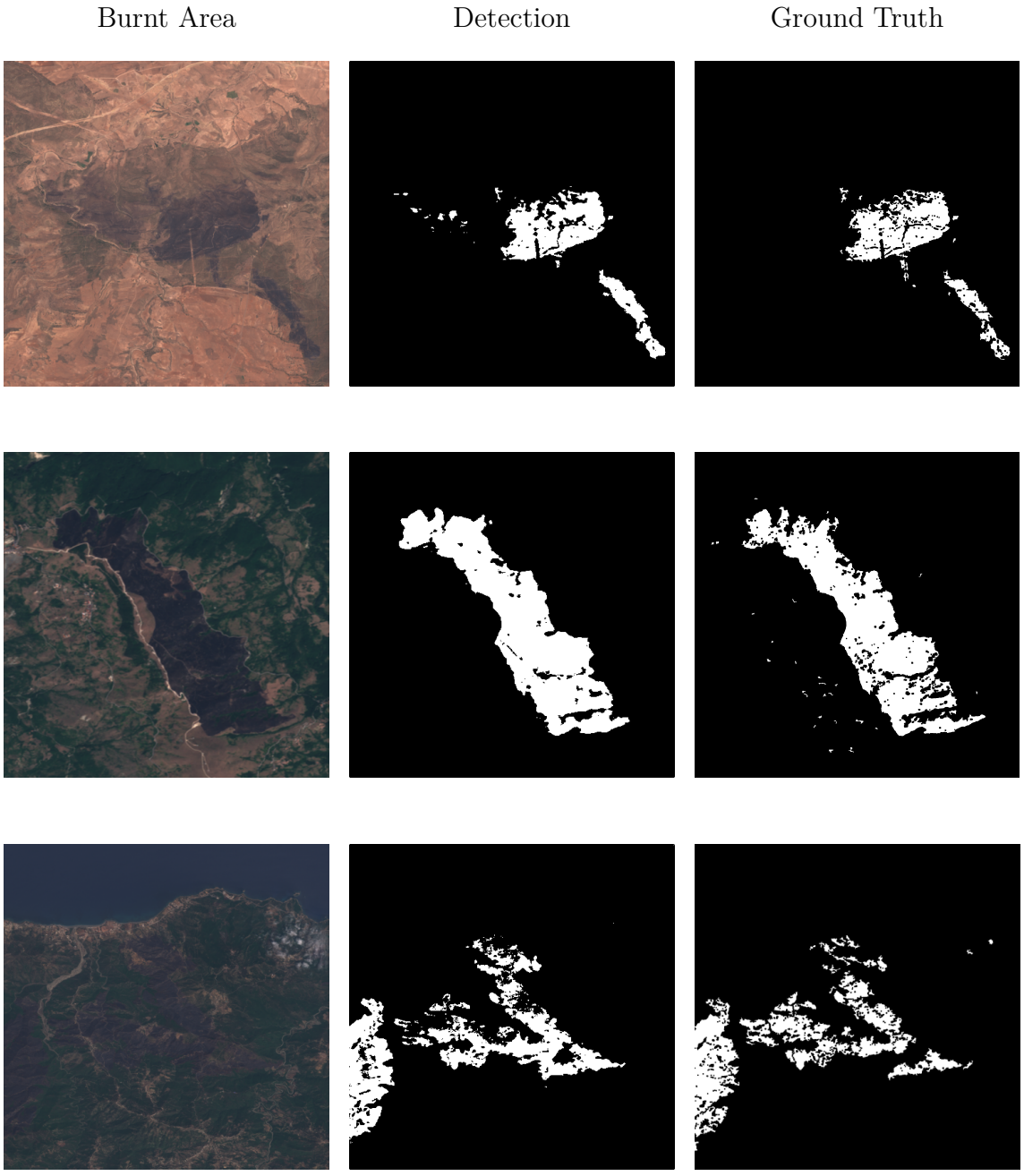


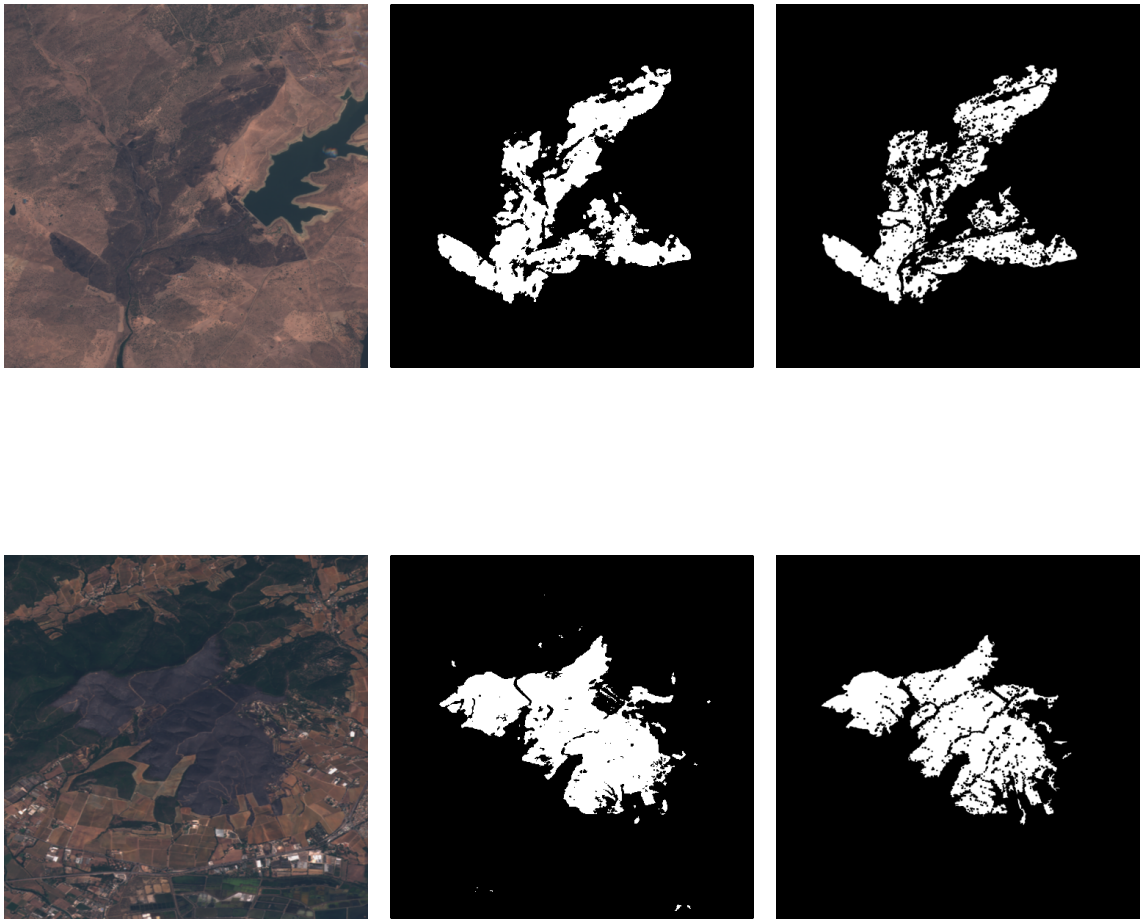


Set	Best Mse
Training	0.025
Validation	0.074

Table 5.17: MSE comparison

Prediction Results





Burnt area	Precision	recall	F1 score
1	0.94	0.98	0.96
2	0.86	0.97	0.91
3	0.73	0.97	0.83
4	0.83	0.91	0.87
5	0.87	0.91	0.89

Table 5.18: DICE coefficient comparison

5.2.5 Training - windowed approach

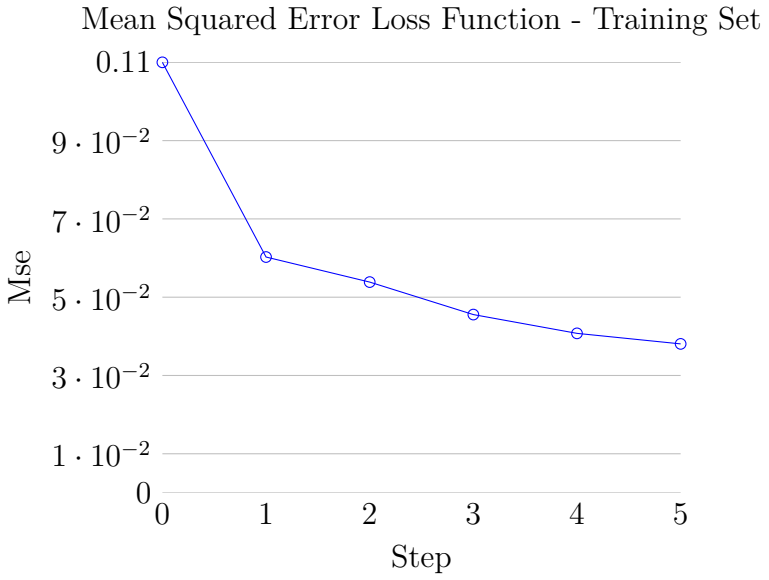
Training 124px window

Training set	Validation set	Batch size
64000	8000	5

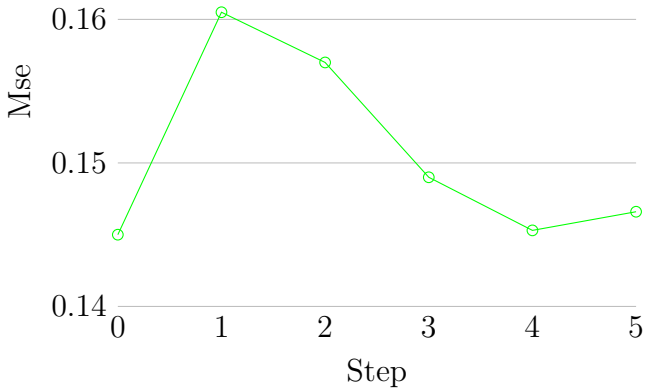
Table 5.19: Training Parameters

No. steps per epoch	No. validation steps	No. epochs
12800	1600	6

Table 5.20: Training Parameters



Mean Squared Error Loss Function - Validation Set



Set	Best Mse
Training	0.03807
Validation	0.1453

Table 5.21: MSE comparison

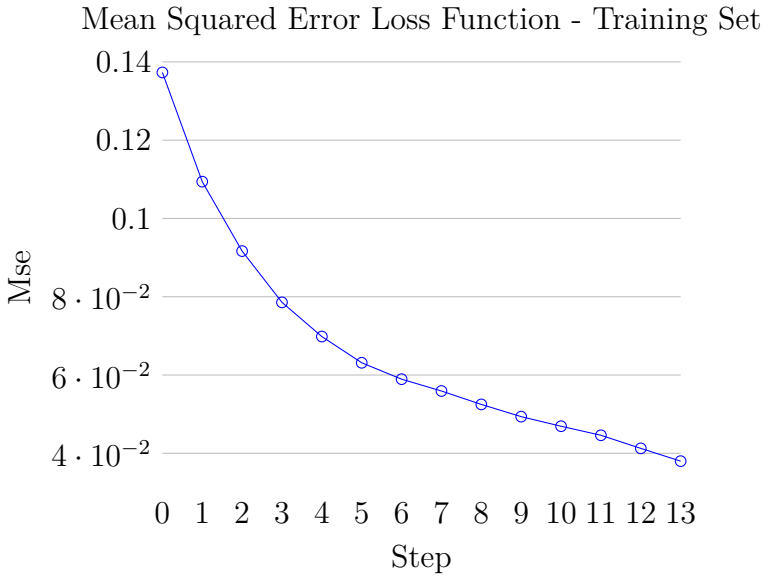
Training 248px window

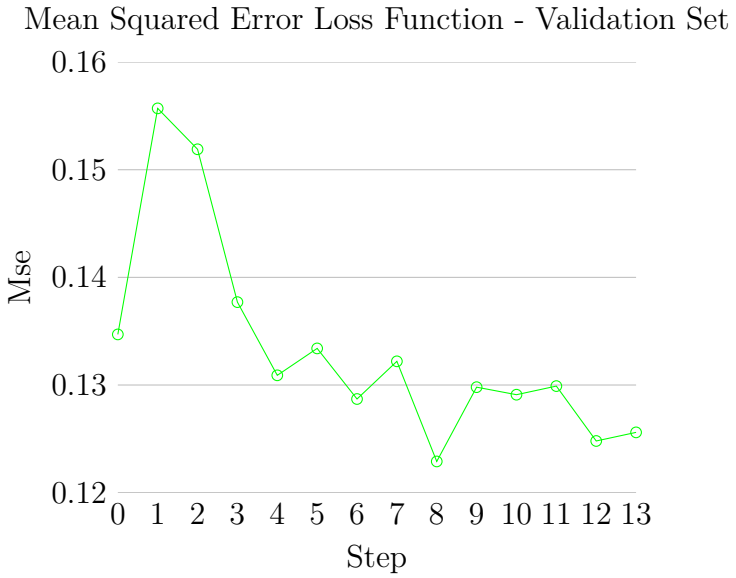
Training set	Validation set	Batch size
16000	2000	5

Table 5.22: Training Parameters

No. steps per epoch	No. validation steps	No. epochs
3200	400	14

Table 5.23: Training Parameters





Set	Best Mse
Training	0.03800
Validation	0.1248

Table 5.24: MSE comparison

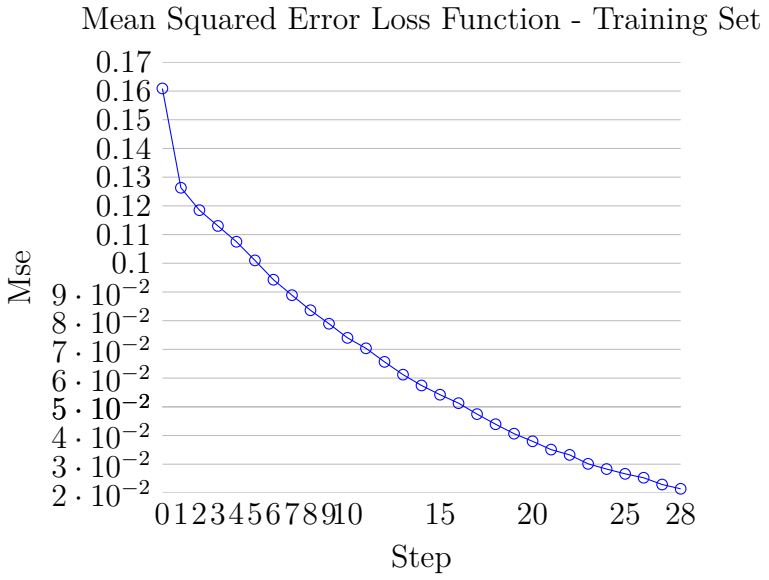
Training 496px window

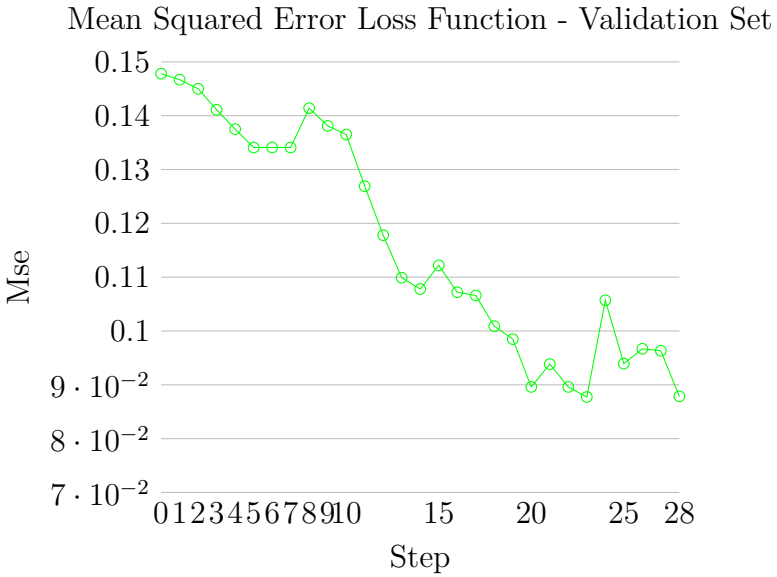
Training set	Validation set	Batch size
4000	500	5

Table 5.25: Training Parameters

No. steps per epoch	No. validation steps	No. epochs
800	100	29

Table 5.26: Training Parameters





Set	Best Mse
Training	0.02143
Validation	0.08785

Table 5.27: MSE comparison

All those three techniques were applied on the same dataset, whose sub-tiles were supplied in the same order, with no shuffle operation. There are two main things to highlight: the number of epochs for each different training was strictly related to the size of the window, the smaller the window, the smaller the number of epochs. This was determined, as already explained, by an increase of the frequency of weights updates, that lead to a faster convergence of the model. The best mse was given by the bigger window. I assumed that it was determined by the fact that this size provided a wider context than the other two solution, which overcame the fact that a wider window tended to have lower performances in terms of localization, due to a deeper model and a higher number of pooling operations.

Chapter 6

Further Developments

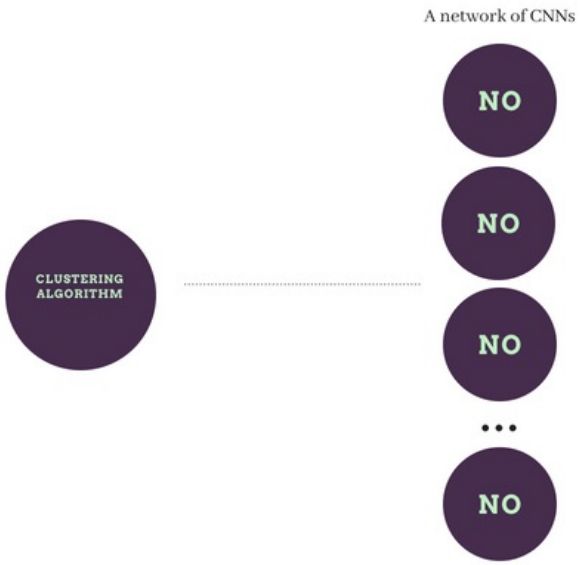
6.1 Image Retrieving

There are two main developments in this section. The first one would be using other services for satellite imagery retrieving, which may provide some additional quality in respect of the used solution. The ideal approach woul be using the Copernicus Open Access Hub, exploiting its important resources, obtaining the information right from the root source. As I mentioned in the previous chapters, there are some drawbacks related to this solution but, having the possibility to work with the original source, would allow to manipulate at the best resolution possible the maps information and, at the same time, it may make possible exploiting the manual annotation provided by the Effis source. Being able to rely on these annotations, would mean eliminating the manual cleaning of the binary masks provided the semi-autonomous approach already explained; thus, it would lead to a bigger dataset, not having to obtain the *before* version of the image, since no comparison would be necessary. The more images obtained, the better the training process, thus, it may lead to a better MSE over the validation images. The other development would be exploiting all the bands provided by the SAR instrument, on board of the Sentinel2 mission. Being able to use bands from b01 to b12 plus the b8a, could determine the inference of additional features, besides the red blue and green channels already collected. An additional development would be not only using the data provided by the Sentinel2 mission, but also those provided by the Landsat and Sentinel1 missions.

6.2 Machine Learning

It would be interesting developing the model obtained by exploiting new data and binary masks. At the same time, there would be the need to enhance the machine

learning architecture by introducing several models so that it woul be possible to get overall better performances. My concerns are about the difficulty to determine the boundaries of all burnt areas around the globe. The problem is mainly represented by the variety of shapes and colors that burnt areas present. I think that a possible solution would be defining a clustering algorithm, capable of splitting input images into different categories, based on vegetation and, thus, color values. All those categories would be assigned to different classifiers able to recognize the burnt areas boundaries in some different environments. So, hotspots in a green area would be assigned to a specifiec classifier while those developed in barren grounds would be assigned to some other. Here, it is shown a possible schema:



6.3 Further studies

It would be important to exploit different machine learning techniques, besides the one used in this project. A final comparison would highlight the pros and cons of each technique, having, eventually, some images comparison. The different approaches showed in the previous chapter were based on an hold out approach, because of the time it would have taken to implement a cross-validation approach for each single experiment. A further step in this direction would be useful.

6.4 Final Considerations

Satellites data represent a huge possibility for data analysis and prediction. There's still a general lack of knowledge in this particular field, due to the youth of this technology. The effort made by Europe and Esa delivered some important programmes, which allow mapping almost every aspect of our planet. This project showed a basic technique in burnt areas mapping, but, though it was based on some limits and constraints, it showed its effectiveness, which can be further developed taking into account all the different parts and layers that compose this tool.

Bibliography

[1] Azure blob storage. https://azure.microsoft.com/it-it/services/storage/blobs/.

[2] Cnn. http://cs231n.github.io/convolutional-networks/.

[3] copernicus api hub. https://scihub.copernicus.eu/apihub/.

[4] Copernicus programme. http://www.copernicus.eu.

[5] Copernicus programme - sentinel missions. http://copernicus.eu/main/sentinels.

[6] Earth engine datasets. https://earthengine.google.com/datasets/.

[7] Effis. http://effis.jrc.ec.europa.eu/.

[8] Effis api. http://effis.jrc.ec.europa.eu/api-test/.

[9] google earth engine. https://earthengine.google.com/.

[10] Google earth engine code editor. https://explorer.earthengine.google.com/#workspace.

[11] Keras. https://keras.io.

[12] Ogc api. https://www.sentinel-hub.com/develop/documentation/api/ogc_api.

[13] Opencv. https://opencv.org/.

[14] Pillow. https://pillow.readthedocs.io/en/5.1.x/.

[15] Postgis. https://postgis.net/.

[16] Product format l0. https://earth.esa.int/web/sentinel/technical-guides/sentinel-1-sar/products-algorithms/level-0-products/formatting.

[17] Product format l1. https://earth.esa.int/web/sentinel/technical-guides/sentinel-1-sar/products-algorithms/level-1-product-formatting.

[18] Product format l2. https://earth.esa.int/web/sentinel/technical-guides/sentinel-1-sar/products-algorithms/level-2-algorithms/formatting.

[19] Product format sentinel 2. https://sentinels.copernicus.eu/documents/247904/685211/Sentinel-2-MSI-L2A-Product-Format-Specifications.

pdf.
[20] Qgis. https://qgis.org/it/site/.
[21] S2 boku. https://s2.boku.eodc.eu/.
[22] S2 boku api. https://s2.boku.eodc.eu/wiki/#!api.md.
[23] S2 boku lib. https://github.com/IVFL-BOKU/sentinel2.
[24] S2 maps. https://s2maps.eu/.
[25] Sentinel hub. https://www.sentinel-hub.com/.
[26] Sentinel hub python. https://sentinelhub-py.readthedocs.io/en/latest/.
[27] Sentinel sat. http://sentinelsat.readthedocs.io/en/stable/api.html.
[28] Slc grd. https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/product-types-processing-levels/level-1.
[29] Tensorflow. https://www.tensorflow.org/.
[30] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. volume 22, pages 3207–3220, 2010. PMID: 20858131.
[31] Simon Haykin. *NEURAL NETWORKS*. Prentice Hall International, Inc.
[32] MikeFlannigana, Alan S.Cantin, William J.de Groot, MikeWotton, AlisonNewbery, and Lynn M.Gowman. Global wildland fire season severity in the 21st century. 2013.
[33] Varun Mithal, Guruprasad Nayak, Ankush Khandelwal, Vipin Kumar, Ramakrishna Nemani, and Nikunj C. Oza. Mapping burned areas in tropical forests using a novel machine learning framework. 2018.
[34] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning*. Packt Publishing, 2017.
[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.
[36] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, Aug 2003.
[37] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell. Understanding data augmentation for classification: When to warp? In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–6, Nov 2016.
[38] Song Yuheng and Yan Hao. Image segmentation algorithms overview. 2017.