POLITECNICO DI TORINO

Thesis

Master's Degree of Communications and Computer Networks Engineering

Automatic classification of healthy / diseased plants using multispectral images

Supervisors:

Prof. Morisio Maurizio

Prof. Ardito Luca

Candidate:

José Doumet

Academic year 2022-2023

Contents

Introduction

The advancements in drone technology in recent years have led their use to be widespread for different applications, ranging from search and rescue, weather monitoring, mapping and surveying to multipurpose videography. In agriculture, one of the most prominent uses for drone imagery is for monitoring plantation health. As the size of plantations expands so follows the complexity of monitoring the health status of every single plant. This can be attributed to the fact that the traditional evaluation of plants is based on visual checks that go along possibly laboratory analyses which can prove costly in not only in terms of time but also economics terms.

This thesis carries on the Dronuts Project which proposes a software and hardware-based solution in order to provide plant classification and monitoring via remote sensing with the aim of assessing the health status of each individual plant relying on collected multispectral drone images. Different software approaches will be applied in for the evaluation, analysis and processing of the drone shots taken over multiple dates to classify independently the health status of a hazelnut plant.

We will discuss at first in Chapter 1 multispectral images, how they are acquired and what type of information they carry in the agricultural sector.

Subsequently, in Chapter 2 we will introduce the different Vegetative Indices used in this project with a brief description about each one of them. These indices will allow us to study and assess the health conditions of any crop. Based on these Vegetative Indices we will extract some metrics, which numerically describe the characteristics of the plant on which will be used in later chapters to build and train machine learning algorithms that will allow a classification of the plant in healthy or diseased.

Later in Chapter 3, we will explore the dataset available, its characteristics and possible inadequacies and what approaches can we use to remedy its criticalities.

Chapter 4 will consist mainly of subdividing the images of the dataset into smaller regions using two different methods: *Grid subdivision* and *k-means clustering*. Subsequently vegetative indices will be calculated, and labels will be applied so we can proceed to the next chapter.

In Chapter 5, we will investigate the applied classical machine learning algorithms like *Random Forest, Logistic Regression* and *K-Nearest Neighbors* that will be used to classify the plants, we will highlight their training process, and evidently assess the performances they have achieved.

In Chapter 6, we will look into the application of *Convolutional Neural Networks* to our dataset. This different approach will be independent of the vegetative indices analysis, using only RGB images as input to the network.

And finally we will try to summarize the steps taken, to give some final considerations and try to list a set of strategies aimed at improvement of the processes seen.

Chapter 1

1.1 Multispectral and Hyperspectral images

Multispectral images are composed of multiple image layers where each of them is taken at a particular wavelength band. Multispectral image camera sensors generally operate in the following bands:

- Blue, 450–520 nm
- Green: 520- 600 nm
- Red: 600–690 nm
- Red-Edge (RE): 670–750 nm
- infrared (NIR): 750–900 nm

Multispectral images can used for remote sensing applications with the aim to acquire information on objects by measuring the electromagnetic radiation reflected or transmitted from their surfaces on the visible sensor used. There are a multitude of ways to obtain multispectral images for remote sensing applications, for instance: drones, planes, satellites, etc.

Hyperspectral images on the other hand are the images in which one (almost) continuous spectrum is measured for each pixel. Contrary to the limited number of bands that multispectral images can have (normally 3 to 10), hyperspectral images deliver a greater number of bands that are extremely adjacent to one another, hence the continuous aspect of these images.

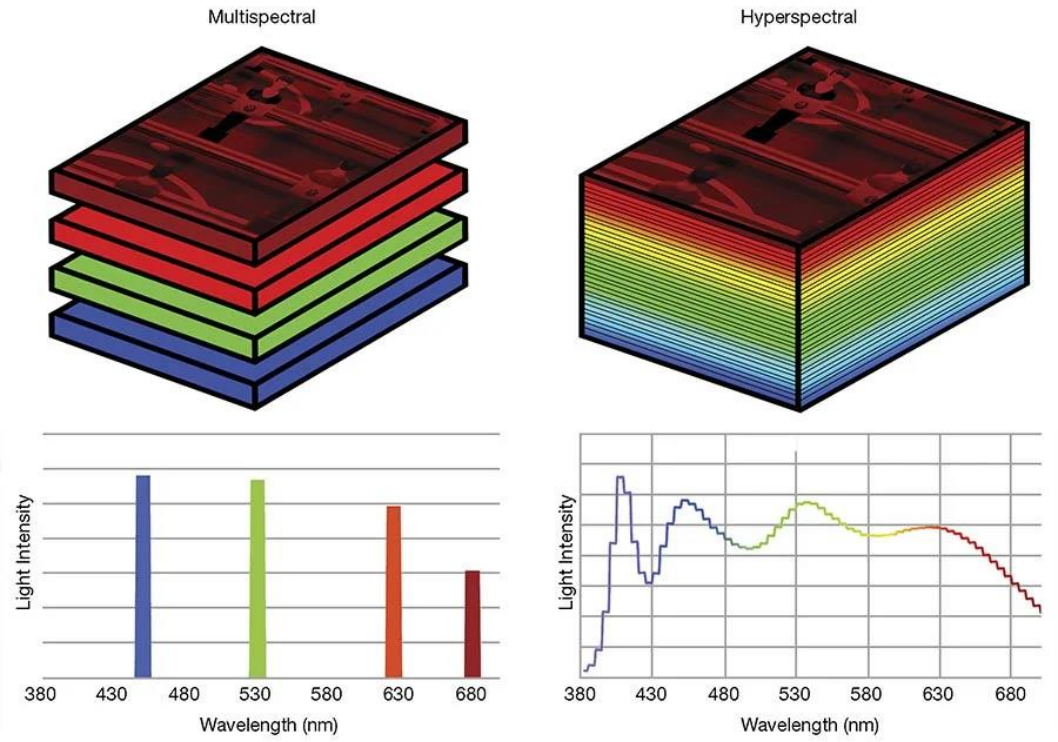


Figure 1.1: Difference between Multispectral and Hyperspectral imaging

In agriculture, hyperspectral imaging can have advantages over multispectral imaging such as, the capacity to acquire and analyze subtle differences in the electromagnetic radiation emitted by disease and soil moisture that cannot be extracted in multispectral imaging due to the limited number of bands available. However, some disadvantages arise with the implementation of hyperspectral images such as, the much greater image size compared to multispectral images (factor of 10 to 1000) and significantly higher cost of equipment and sensors.

Nonetheless, with the technology on the market today, it is possible to obtain a type of more accurate and rich information using hyperspectral sensors. However, for this project, the dataset will solely consist of multispectral images.

Remote sensing of objects in principle are carried out firstly by the acquisition of the electromagnetic radiation through the visible sensors of the necessary instrumentation, followed by data processing of the received signal and its subsequent conversion to a digital image. Finally, the acquired visual information will be interpreted and analyzed.

In this project, multispectral images shots were carried out on a hazelnut plantation by means of a drone. The first phase of the project was completed, and it provided the dataset of images required for interpretation and analysis. The drone used to take the multispectral images is a DJI P4 Multispectral, it is equipped with six sensors, having the following characteristics:

Camera

Sensors	Six 1 / 2.9 "CMOS sensors, including an RGB sensor for visible light images and five monochrome sensors for multispectral image acquisition. Each Sensor: 2.08MP Effective Pixels (2.12MP Total)
Filters:	Blue (B): 450nm ± 16nm, Green (G): 560nm ± 16nm, Red (R): 650nm ± 16nm, Red-Edge (RE): 730nm ± 16nm, near infrared (NIR): 840 nm ± 26 nm
Targets	FOV (field of view): 62.7 ° Focal length: 5.74mm (35mm equivalent format: 40mm), ∞ autofocus Aperture: f / 2.2
RGB sensor ISO range	200 - 800
Gain of the monochrome sensor	1 - 8x
Global electronic shutter	1/100 - 1/20000 s (visible light); 1/100 - 1/10000 s (multispectral)
Maximum image size	1600 × 1300 (4: 3.25)
Photo format	JPEG (visible light images) + TIFF (multispectral images)
Supported file systems	FAT32 (≤ 32GB); exFAT (> 32 GB)
Supported SD cards	microSD with a minimum write speed of 15 MB / s. Capacity: 128GB. Class 10 or UHS-1 standard

Figure 1.2: DJI P4 camera sensors technical characteristics

The drone sensor can take RGB images, Red-Edge (RE) characterized by a wavelength of 730 nm ± 16nm and NIR (Near Infrared) to one wavelength of 840nm ± 26nm.

Therefore, the DJI drone is able to deliver a multispectral image consisting of a total of five bands that will be crucial for the study and health assessment of the hazelnut plants. Each band will bring different types of information that will be subsequently used in order to calculate and obtain the Vegetational indices.

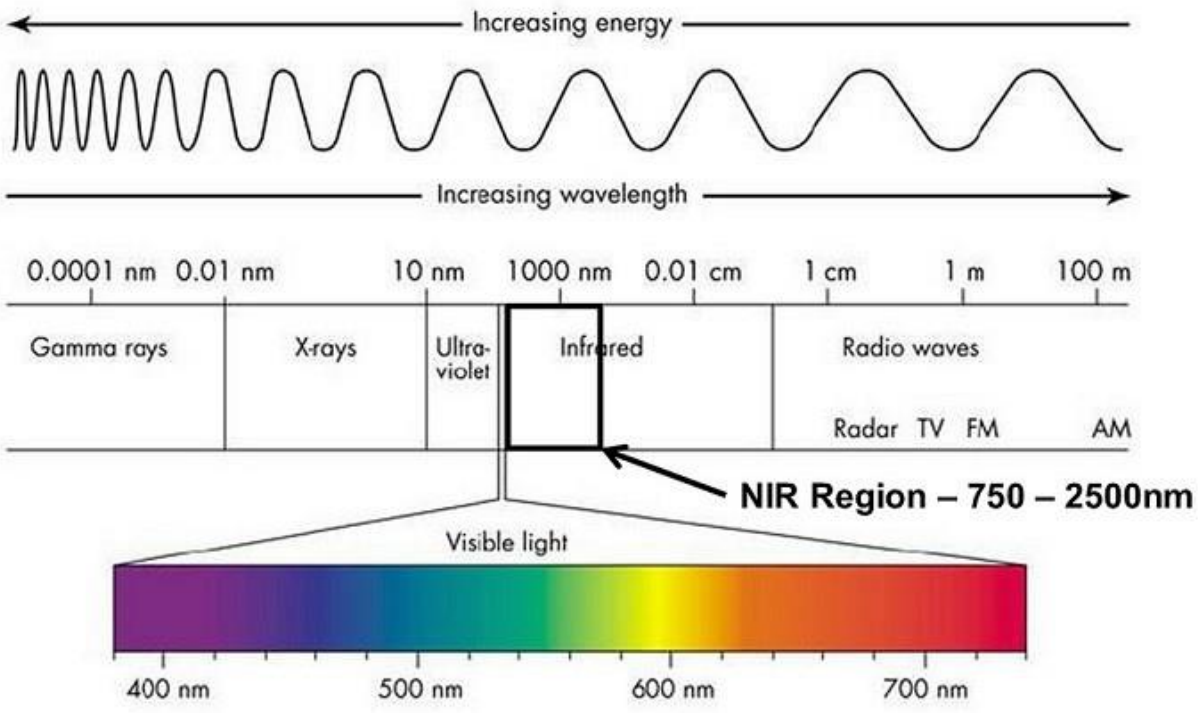


Figure 1.3: Electromagnetic spectrum

1.2 Spectral Reflectance of Vegetation

We have seen previously, that in addition to the visible spectrum (RGB), the sensors of the DJI drone are able to take measurements to two different wavelengths: Red-Edge (RE) and Near-Infrared (NIR).

- The (RE) Red-Edge band refers to a range of the electromagnetic spectrum between Red and NIR, where rapid change can be observed in the reflectance property of the vegetation.
- NIR (Near Infrared): is a region of the electromagnetic spectrum that ranges from a wavelength of 780 nm to 2500 nm. It belongs to the infrared band, but at the same time it is the region closest, in terms of wavelength, to the visible spectrum. NIR is applied in spectroscopic analysis, used for the analysis of the radiation absorbed and reflected by a surface (soil, trees, etc.). Given its properties, it sees wide use not only in the agricultural field, but also in the field of medical and physiological diagnostics, in the study of atmospheric characteristics and in astronomy.

Chlorophyll is a green pigment, present in all green plants which is responsible for the absorption of light to provide energy for photosynthesis. The chlorophyll absorbs most of it of light in the visible spectrum (400 nm - 700 nm), but the structure cellular structure of the leaves, on the other hand, is characterized by a strong reflectance when moving towards the near infrared band (700 nm-1100 nm). Chlorophyll is a good indicator of the plant's production potential. It can be also used to understand the plant's nutrient status, stress due to water, disease outbreak, and more.

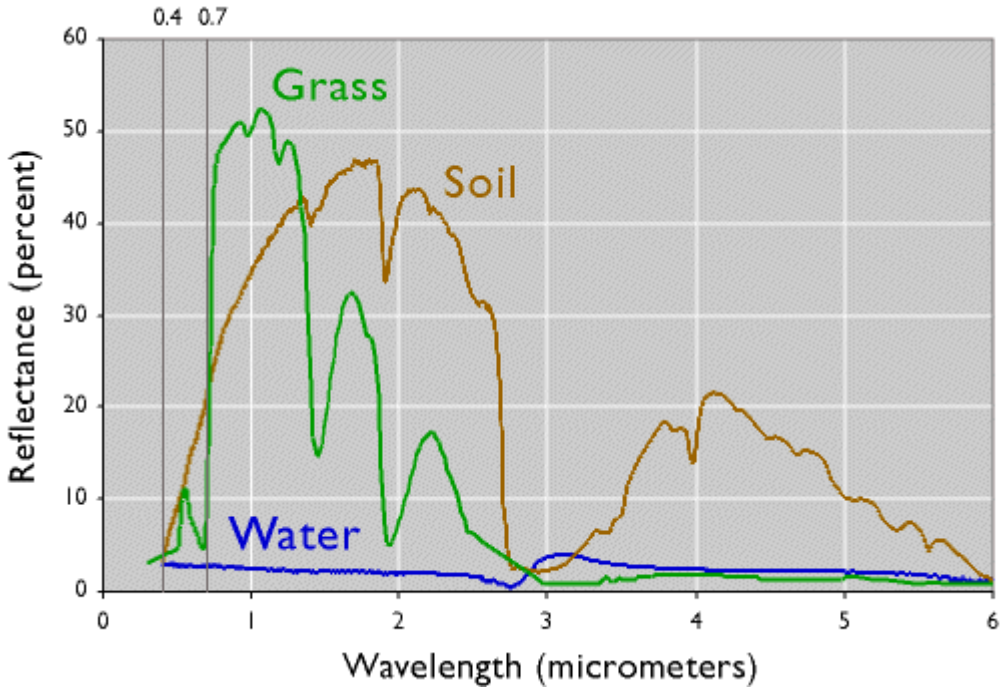


Figure 1.4: Spectral signatures of water, vegetation and soil

We can observe from figure 1.4 how green and healthy vegetation is characterized by a lower light reflectance in the visible spectrum. The low reflectance in the spectral regions of blue and especially red, which are in fact the ranges most involved in the photosynthesis process. The reflectance, however, begins to substantially rise showing as a maximum range variation of the band called Red-Edge. This can be attributed to the fact that historically the cells of the leaves have evolved to disperse the solar radiation in the range of the NIR, as the level of energy per photon in this domain is not sufficient for the synthesis of organic molecules. The solar radiation that plants use as an energy source in the photosynthesis process, is called photosynthetically active radiation (PAR). Therefore, the plants will look quite dark in the images taken in the PAR range and relatively bright in the near infrared.

Interestingly, knowing the spectral reflectance curve of healthy vegetation it is possible to go and identify the vegetation which instead turns out to be suffering, as it will be characterized from lower reflectance percentages in the NIR band (and beyond), and higher percentages in the visible bands.

It is also possible to discriminate from vegetation, both the ground and the possible presence of water. In fact, we note how the reflectance curve of the water is characterized by a very high percentage of absorption in the infrared range and beyond. While that of the ground devoid of vegetation has a lot of curves more uniform, showing no obvious spikes at any spectral range.

However, it should be noted that the spectral signature of the terrain is anyway subject to several of its characteristics, such as: the content of humidity, its consistency, surface characteristics (rocky, sandy, clayey), the possible presence of iron oxide and finally the organic material present on it.

Chapter 2

2.1 Vegetative Indices

We discussed in the previous chapter the importance of multispectral and hyperspectral imagery providing a multitude of electromagnetic bands in order contribute to a wide range of information necessary for the evaluation and examination of a crop's state of health. Using these spectral bands, we can infer distinct characteristics from the image into consideration, semantically differentiating different aspects (vegetation vs soil, healthy plant vs sick plant, etc.). The set of tools required in order to extrapolate the multispectral and hyperspectral vegetation image properties are called **Vegetation Indices (VIs).**

In simple terms Vegetation Indices (VIs) are combinations of surface reflectance at two or more wavelengths designed to highlight a particular property of vegetation. They emphasize mainly photosynthetic activity in crops and therefore are ubiquitously implemented in remote sensing agriculture applications. The majority of Vegetation Indices make use of the inverse relationship between red and NIR bands reflectance associated with healthy green vegetation. There exists a plethora of Vegetation Indices in the scientific literature, each one can be calculated using a combination of different spectral bands (in multispectral and/or hyperspectral imaging), can yield different functionalities and accentuate a wide range of characteristics.

A few of these Vegetation Indices have been considered in this project in order to evaluate the status of the plants. The choice of the selected Vegetation Indices was based upon the two mains semantic considerations required in order to process the images provided in the dataset:

- Vigor Indices used to differentiate the plants from their surrounding (ground, soil, roads, people, etc.).
- Chlorophyll Indices required to evaluate the health status of the plants (greenness of the plants, chlorophyll content, nitrogen reflectance, etc.).

2.1.1 NDVI

NDVI (Normalized Difference Vegetation Index), is one of the most used Vegetative indices used in remote sensing measurements. It quantifies vegetation by measuring the normalized difference between near-infrared band, which vegetation strongly reflects, and red band.

It is usually calculated by pixel-to-pixel basis and is quantified using this formula:

$$NDVI = \frac{NIR - RED}{NIR + RED}$$

NDVI can assume a minimum value of -1 and a maximum value of +1

While positive values can signify the presence of vegetation, strictly negative value can be attributed to water (or clouds in case of satellites imagery), values around 0 represent properties of the soil (dirt, rock, sand, etc.).

On the other hand, values exceeding 0.3 can characterize a green area such as plantation fields, forests.

NDVI can also indicate if a plant is sick if it falls below a certain threshold, which can be attributed to a low reflectance of the NIR band.

Figure 2.1: Vegetation spectral bands reflectance

2.1.2 GNDVI

The second vegetative index used in this project is GNDVI (Green Normalized Difference Vegetation) is an indicator of the plant photosynthetic activity, and it is commonly used vegetation index to assess the nitrogen and water content of the plant. Compared to the NDVI, it is a chlorophyll index normally applied at later stages, as it saturates later than NDVI.

Like NDVI, GNDVI will also assume values in range from -1 to 1:

GNDVI uses the near infrared (NIR) and green band (GREEN) of the electromagnetic spectrum. As opposed to NDVI, GNDVI is also more sensitive to chlorophyll variation, hence a better determinator of the plant health status, and has a higher saturation point. While NDVI is more sensible for estimating crop vigor during the early stages, GNDVI can be used in crops with dense canopies or in more advanced stages of development.

GNDVI is calculated using the following formula:

$$GNDVI = \frac{NIR - GREEN}{NIR + GREEN}$$

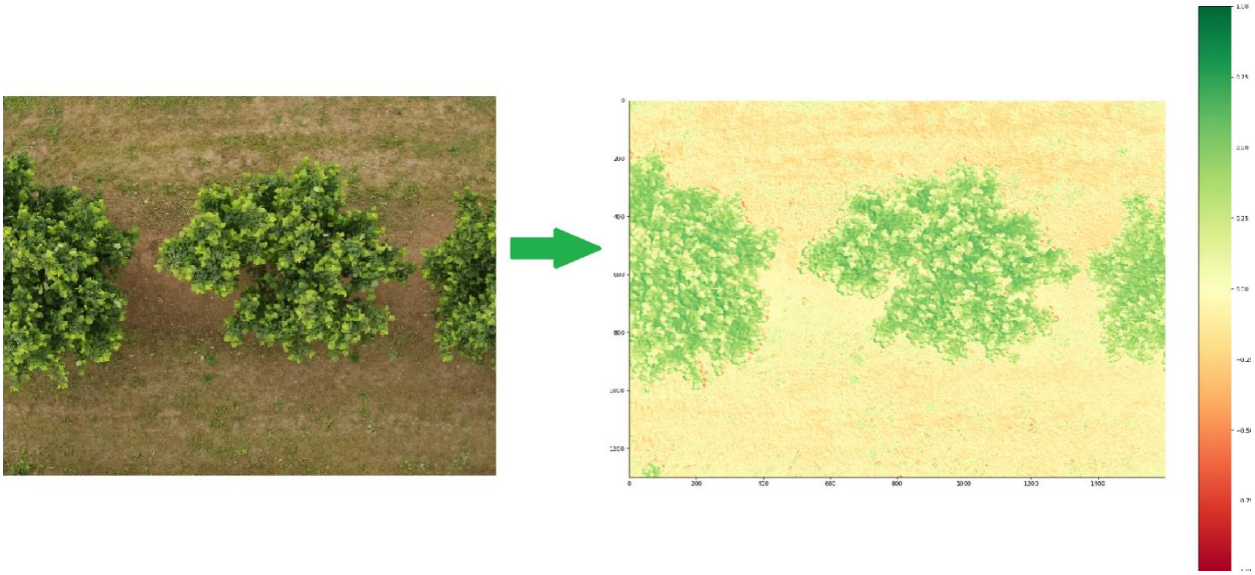


Figure 2.2: GNDVI transformation

2.1.3 GCI

GCI (Green Chlorophyll Vegetation Index). This index is used to assess the health condition of the crop by estimating its chlorophyll content. It is dependent on the NIR and Green spectral bands and is calculated using the following formula.

It can assume values from -1 to + infinity

GCI is computed as follows:

$$GCI = \frac{NIR}{GREEN} - 1$$

2.1.4 NDREI

NDRE (Normalized Difference Red-Edge Index) is another index that is commonly used to determine the amount of chlorophyll in plants. It is calculated using a combination of a Near-InfraRed (NIR) band and the Red-Edge range between visible Red and NIR.

NDREI is calculated using the following formula:

$$NDREI = \frac{NIR - RED\_EDGE}{NIR + RED\_EDGE}$$

The choice of using this vegetative index can be attributed to the fact that the optimal time to apply NDRE index is mid-to-late growing season when the plants are starting to mature, which fits parts of the dataset to be analyzed.

2.1.5 RECI

This fifth index (Red-Edge Chlorophyll Index) is an estimator the chlorophyll abundance of leaves. Like NDRE, it is determined using the ratio of reflectivity in the near-infrared (NIR) and red-edge (RE) bands.

$$RECI = \frac{NIR}{RED\_EDGE} - 1$$

2.1.6 NRI

NIR (Nitrogen Reflectance index) is an index responsible for determining the nitrogen content in plants. Nitrogen is an indispensable macronutrient for plant growth and function and is a key component in plant production of proteins and enzymes. Nitrogen is also a part of the chlorophyll molecule, and it ensures the availability of energy when and where the plant needs it to optimize yield.

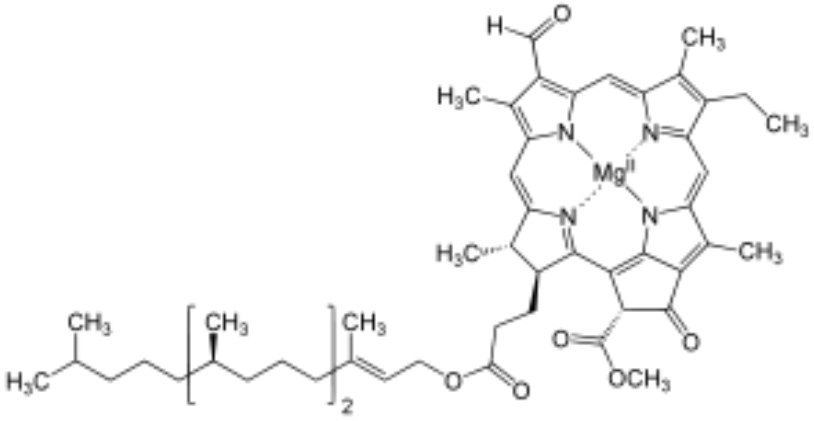


Figure 2.3: Chlorophyll molecule

Nitrogen deficiency, however, can manifest symptoms which includes:

- Poor plant growth.

- Small Leaves.

- Low abundance of chlorophyll which make the leaves become pale green or yellow.

Nitrogen Reflectance Index can highlight these symptoms using a combination of a green and red bands expressed in the following formula:

$$NRI = \frac{GREEN - RED}{GREEN + RED}$$

It should be noted that the lower the value of the NRI, the higher the reflectance is. Consequently, healthier plants will exhibit lower NRI values.

2.1.7 GI

GI (Greenness Index) highlights the level of greenness in plants. It is directly correlated with the chlorophyll content on the plants and therefore their health status (the lower the value the higher is the green reflectance).

Greenness Index is calculated using this formula:

$$GI = \frac{GREEN}{RED}$$

Similar to NRI, healthy vegetation will have lower GI values than unhealthy one. The higher the reflectance the lower NRI values.

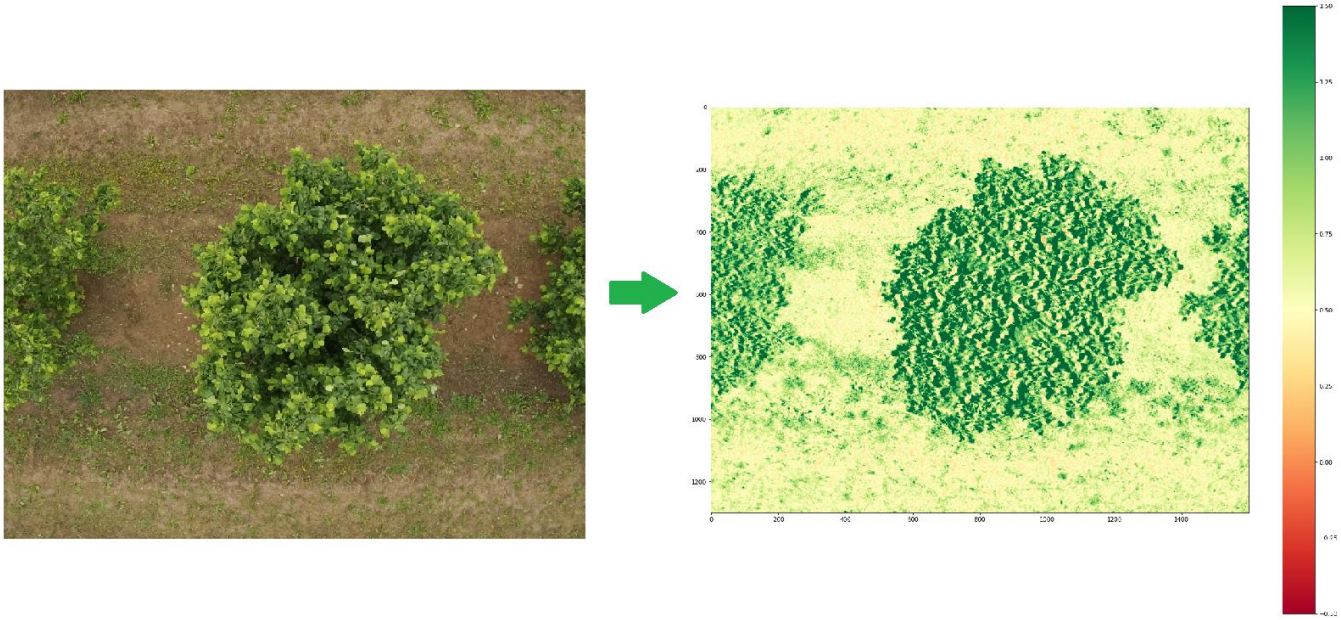


Figure 2.3: GI Transformation

We can clearly from figure 2.3 that the vegetation highlighted is darker compared to the ground, therefore having lower values.

2.1.8 TCARI

TCARI (Transformed Chlorophyll Absorption and Reflectance Index), is another index that allows to identify the areas of the field affected by chlorosis, which could be caused by nutritional deficiencies or attacks of plant diseases. It makes use of the red, green and red-edge bands.

TCARI is calculated by the following formula:

$$TCARI = 3(RED\_EDGE - RED)\frac{0.2(RED - GREEN)}{RED\_EDGE - RED}$$

2.1.9 SAVI

The last used vegetative index is SAVI (Soil-Adjusted Vegetation Index) which applies a correction Normalized Difference Vegetation Index (NDVI) for the influence of soil brightness in areas where vegetative cover is low. The correction factor is designated by L.

It makes use of the NIR (near infra-red) and red bands, it is calculated by the following formula:

$$SAVI = (1 + L)\frac{NIR - RED}{NIR + RED + L}$$

The L correctness factor can range from 0 to 1. We can notice that for L = 0 we get the NDVI formula. Normally, for areas with scarce vegetation we can set L = 1. L can assume different values depending on the environment in question. However, a good compromise is to set L = 0.5.

Chapter 3

3.1 The Dataset

After having discussed and examined the vegetative indices, these evaluative tools can be applied to our dataset.

The dataset is comprised of 3330 images of hazelnut trees taken from a DJI drone. The images relate to about 185 plants taken in hazelnut plantation fields in the Province of Cuneo in the Italian region Piedmont, an area renowned for hazelnut farming. The images were taken in 2022.

The dataset consists of two fields shootings, and therefore can be divided in to 2 subsets:

The **Carrù** dataset and the **Farigliano** dataset.

- The **Carrù** dataset consisting of 127 hazelnut trees with 3 drone shooting days for each tree over a period of around 2 months (30/05/2022, 22/06/2022 and 15/07/2022 respectively). The images were taken in a hazelnut tree field in Carrù (Carrù is a *comune* in the Province of Cuneo in the Italian region Piedmont).

Figure 3.1: Carrù photo areal image

Figure 3.2: First example of a Carrù plant images over the three shooting sets

Figure 3.3: Second example of another Carrù plant images over the three shooting sets

- The **Farigliano** dataset consisting of 58 hazelnut trees with 3 drone shooting days for each tree over a period of around 2 months (03/06/2022, 24/06/2022 and 16/07/2022 respectively). The images were taken in a hazelnut tree field in Farigliano (Farigliano is also a *comune* in the Province of Cuneo in the Italian region Piedmont).

Figure 3.4: Carrù photo areal image

Figure 3.5: First example of a Farigliano plant images over the three shooting sets

Figure 3.6: Second example of a Farigliano plant images over the three shooting sets

For every drone shooting set and for each hazelnut tree, we have 6 different images consisting of 1 *.JPG* that includes all RGB channels and 5 multispectral images:

- 3 *.TIF* format image for RGB (Red, Green and Blue) bands.
- 1 *.TIF* format image for RE (Red-Edge) band.
- 1 *.TIF* format image for NIR (Near-Infrared) band.

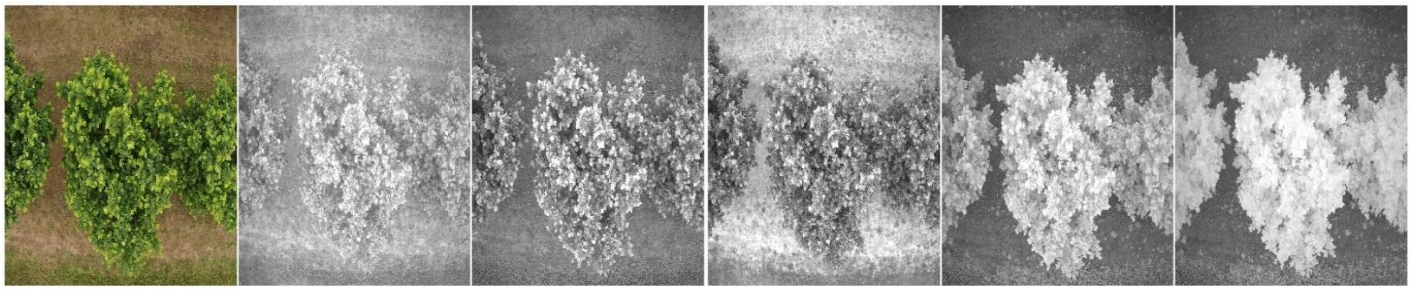


Figure 3.7: Set of multispectral images for one plant (From left to right: RGB, Blue, Green, Red, Red-edge, NIR)

For every shooting, each plant has its own folder consisting of these images. The datasets also include 5 additional aerial multispectral images of the field for every drone shooting set. No labeling regarding the health status of the trees was given at this stage.

As previously mentioned, our main objective is to be able to explore different analysis approaches in order to classify diseased from healthy hazelnuts trees using an automated process. We should therefore try to mimic an expert botanist on the field trying to visually assess the health status of trees. The latter requires a semantically driven approach in order to isolate the tree from its surroundings before applying our vegetative indices for evaluation.

3.2 Programming Tools and Language Used

Due to the large size of the dataset previously presented (3330 images). The choice of our programming tools used in order to analyze and explore the dataset will be the Google offered products: **Google Colab** for programming along with **Google Drive** for storage.

Colab is a free **Jupyter** notebook environment that runs entirely in the cloud. It does not require a setup and can be accessed directly from any browser using a **gmail** account. The programming language used **is Python 3.7** which has a multitude of practical libraries that can be applied to make the coding experience shorter and more efficient. Python also supports many popular machine learning libraries which can be easily loaded in the notebook such as **TensorFlow, scikit-learn, PyTorch.**

The main benefit and advantage of **Colab** is making use of cloud computing: The code will be executed on **Google** servers. This means that the execution of the code will be independent from its coding environment which will allow less-performing computers in the development and execution of the code, more importantly saving on spatial and temporal resources.

Figure 3.8: Jupyter and Google Colab logos

Google Drive is a free cloud-based storage service that enables users to store and access files online**.** It will be used as our storage drive in which the input and output of our code will be saved. The usage of a cloud-based storage with direct access to our coding environment and the execution of our code will prove crucial in terms of saving resources on downloading and uploading our data every time we execute our code. A cloud-based storage means that the data we are using in our project will not be limited to a single localized storage but can be accessed at any time and at any place using a **gmail** account.

3.3 Exploring the Dataset

As we saw earlier, our dataset is made of multiple multispectral images belonging to hazelnuts plants, the images of these trees were taken from a DJI drone.

We notice however a few problems in our dataset:

- Most of the images do not appear to have only one tree as their subject, instead, multiple trees can be present in a single image.

Figure 3.9: Example of image containing many trees

This makes it sometimes hard to differentiate the outline of trees from one another (overlapping leaves and/or branches).

Figure 3.10: Example of image with overlapping trees

The image above figure 3.10 shows how difficult defining the contour of a plant can be.

- Image lighting conditions can differ greatly between shooting sessions including brightness and sun angle and it can be shown across all multispectral bands.

Figure 3.11: Same plant with different lighting conditions

Looking at the image figure 3.11 on the left we can almost distinguish no shadows in the image, however in the image of the right we can almost directly notice that

the sun hits the tree unevenly, creating areas of shade. In consequence, this will lead to a loss of details in the images and eventually information loss.

– For some plants, photos taken across the different bands do not seem to line up perfectly on top of one another.

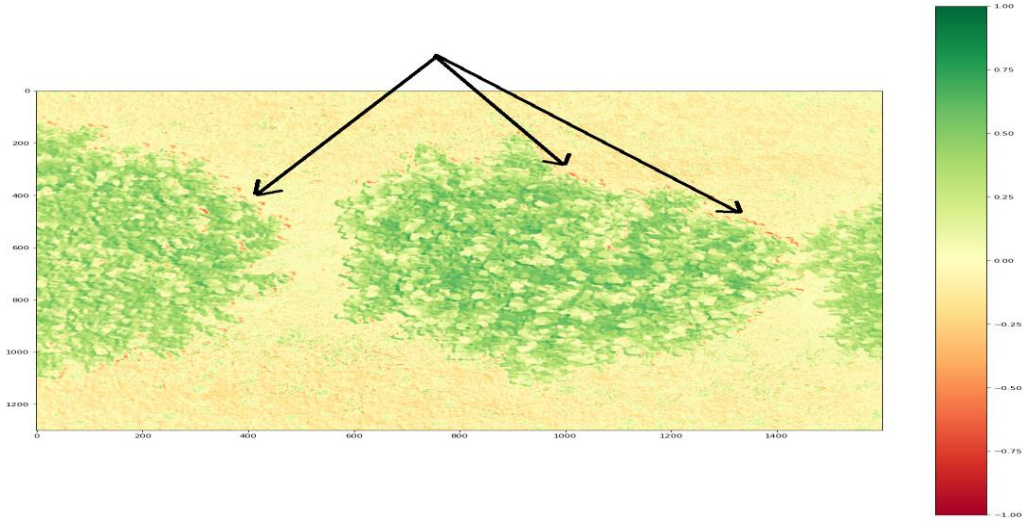


Figure 3.12: GNDVI transform showing artifacts due to multispectral bands not being superimposable

We can clearly see from figure 3.12 artifacts along the contour of the tree, the artifacts signify a low value for the vegetative index, which in practical terms can be considered erroneous. This discrepancy in alignment between multispectral images (which can be vertical and horizontal) can be attributed to the delay and the drone movement between the shots, making some of the multispectral images not superimposable. This might pose a challenge in the image processing phase of our project.

3.4 Possible Solutions

Solving the dataset problems requires the use image processing tools on our dataset. These tools will prove necessary in defining and isolating subject trees regardless of their surroundings, lighting conditions and the misalignment of the multispectral images drone shots. This step is necessary in order to proceed in our project

We suggest three approaches that can be applied:

3.4.1 Manual editing

Editing the picture manually which consists of:

- In case of the multispectral images being misaligned: Manually aligning all the multispectral images to the RGB photo for each plant using a photo editing software so they become superimposable.

- Manually drawing the contour of subject tree on the RGB photo hence removing other trees not under consideration, extracting the RGB drawn mask and applying it to rest of the multispectral images.

3.4.2 Automated editing

Creating a script that can automatically predict the boundaries of the subject tree regardless of lighting conditions and misalignment of multispectral bands, then removing trees not under consideration. In this approach, vegetative indices come into play.

Our first step is to highlight the vegetation present in the image using a vegetative index. The choice of our vegetation index is the commonly used NDVI that, as described in the previous chapter, can detect vegetation in multispectral imaging using the red and NIR spectral bands.

In our dataset, the NDVI mask will be applied in order to differentiate the trees from their surroundings (soil, ground, water), where according to the NDVI formulation, a value below 0, is not considered as vegetation, so 0 and negative values can be assumed to be non-vegetation.

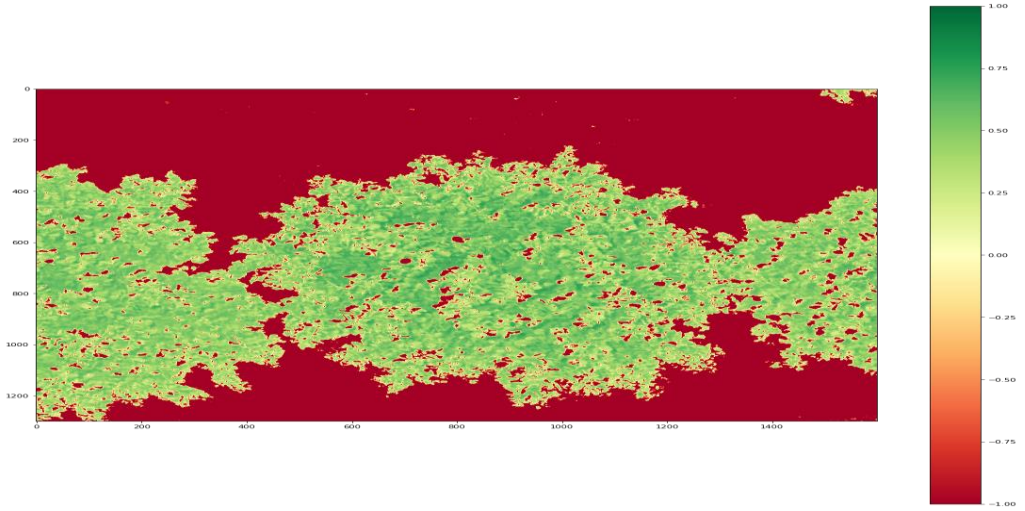


Figure 3.13: NDVI transform with threshold = 0.2

In figure 3.13 ,the NDVI is computed with a specific threshold. If the value of the NDVI computed for every pixel falls below 0.2, the NDVI pixel value will be automatically set to a normalized negative value (-1 in our case). Using this technique we can solely highlight the vegetation present in the image while omitting everything else (ground, soil, road, etc.).

In order to account for uneven lighting conditions and misalignment in the multispectral images, a higher threshold for the NDVI will be selected. This will allow the NDVI mask to be more conservative in terms of contouring and masking the vegetation and therefore the **misalignment effect** between the multispectral bands will be reduced. The NDVI threshold described in figure 3.13 could be set for instance to 2.5 or 3, consequently reducing the number of artifacts considered within the tree boundaries.

It should be noted that this approach may lead to information loss as we are trimming portions of the edges of the tree. However, the removed portion will only consist of a small fraction relative to the whole tree and consequently this method seems to be a plausible compromise for our analysis.

The next step will be differentiating and isolating the subject tree relative to other trees or vegetation in an image. The difficulty of the implementation of this step is entirely dependent of the vegetative content of the images. Let's say for instance the tree under consideration is the only vegetation present in the image, an NDVI mask will solely be sufficient in isolating the tree figure 3.14.

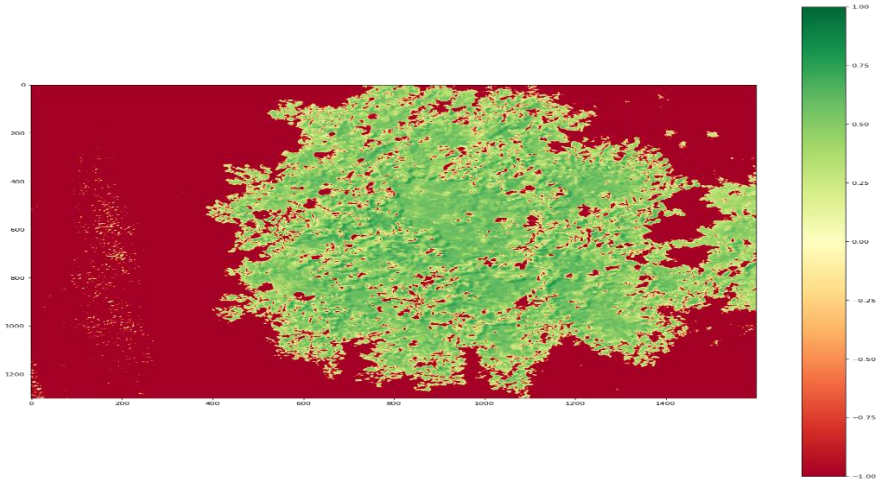


Figure 3.14: Tree image where NDVI (threshold = 0.2) mask alone is sufficient for its isolation

In another example if there are multiple unconnected trees, we can predict vertical and horizontal bounds to the tree under consideration and therefore we will be able to isolate it from the rest of the vegetation (figure 3.15).

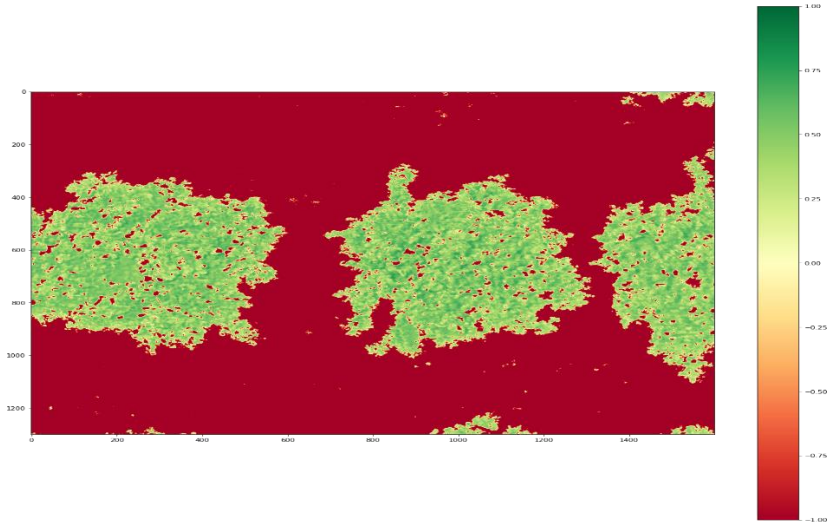


Figure 3.15: Image where an NDVI (threshold = 0.2) mask is not sufficient for the tree isolation, but boundaries can be predicted

However, if our subject tree is surrounded by other trees where they appear to be intertwined and overlapped, it becomes almost impossible to establish an automatable method to be able to distinguish the contour of the tree under consideration (figure 3.16).

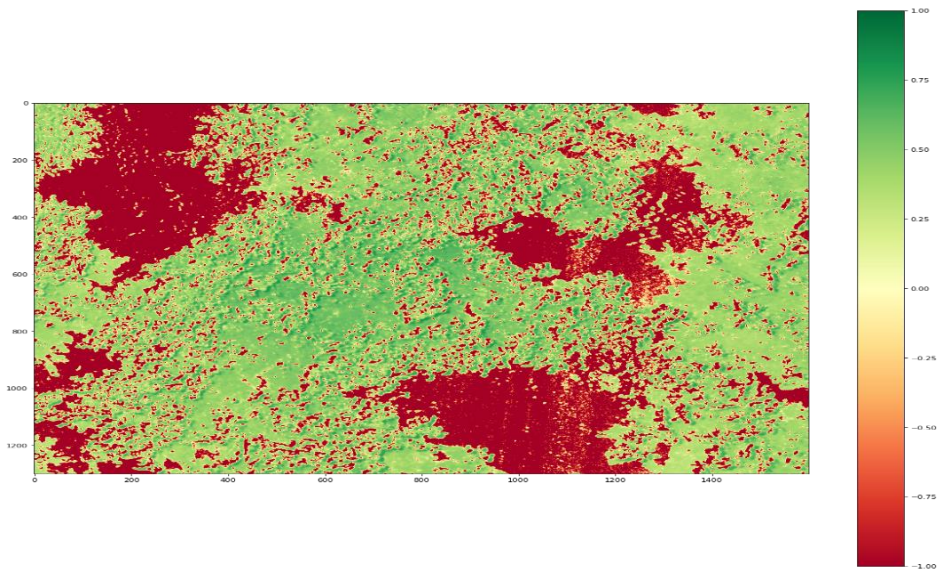


Figure 3.16: Image where an NDVI (threshold = 0.2) mask is not sufficient for the tree isolation and boundaries are impossible to predict

3.4.3 Semi-Automated editing

Semi-automated editing consists of using a combination of manual and automated editing. For instance, we can apply an automated contour delimiter based on a vegetative index in order to define the body of the tree and then we can manually define horizontal and vertical boundaries respectively. The automated contour delimiter in that case will also account for the multispectral images misalignment.

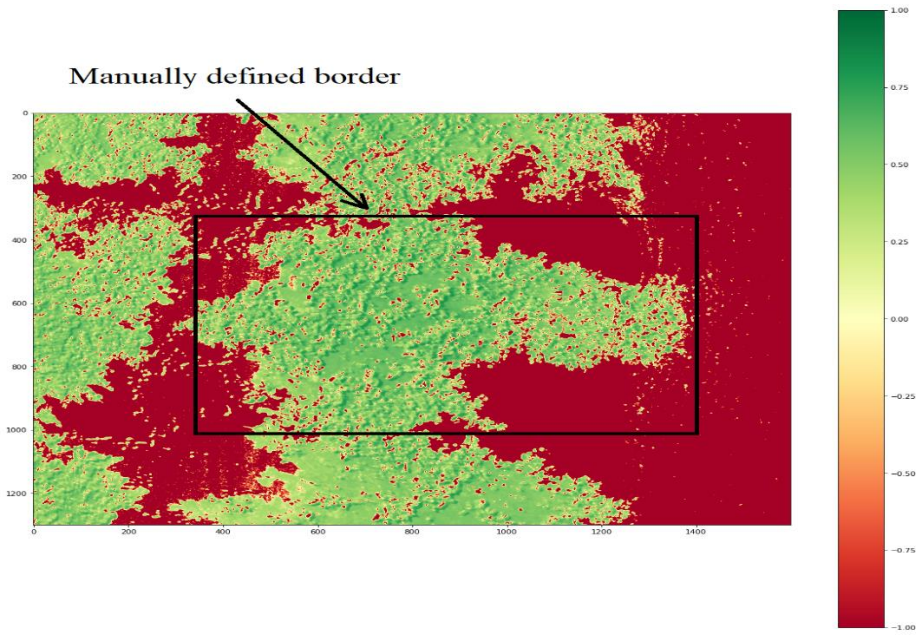


Figure 3.17: Manually defining boundaries on NDVI (threshold = 0.2) calculated image

Due to the massive number of images in our dataset, manual editing will not be considered as it will take an absurd amount of time to edit the images by hand, therefore defying the purpose of the project. Depending on the dataset, the images will be edited using an automated and/or semi-automated editing process.

3.5 Implementation of Automated Editing

In this approach we will apply an automatic editing process to the images. The process applied consists of multiple steps implemented in **Python** where each of the steps will be explained as follows:

Let’s say we want to automatically edit this image in order to isolate the tree under consideration (in this case the tree in the middle):

Figure 3.18: Input of automated editing process

The first step consists of applying the NDVI transform pixel-by-pixel to our image. The NDVI will have a variable threshold for which pixel values can be considered or rejected and a variable added bias value that can be useful in images containing an excessive amount of vegetation around the considered tree. The chosen threshold will assume values of 0.2 or 0.3 depending on the dataset to be analyzed. Given that the values of the NDVI transform belong to [-1,1], if a NDVI pixel value is under the threshold, it will automatically assume a value of -1 (figure 3.18).

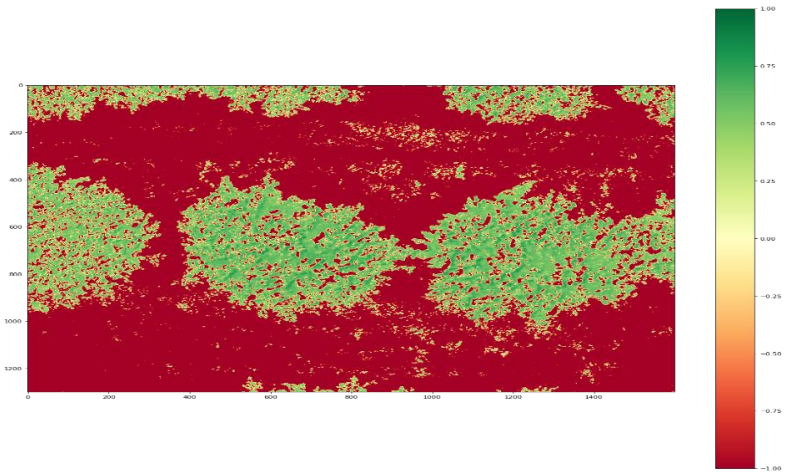


Figure 3.19: NDVI transform with threshold

The output of the NDVI mask will be used as an input to our contour delimiter. This step will draw a contour around the NDVI image using the **Python Open CV** library. It will then consider hierarchically the biggest contour drawn and neglect the other ones, due the fact that for instance, small bushes or foliage scattered on the ground formulate a smaller area compared to the subject tree, nonetheless, they might yield an NDVI value greater than the threshold applied, however, these insignificant vegetation patches do not belong to the tree under consideration and hence should not be considered in our analysis. The drawn contour is then filled and passed to the next step.

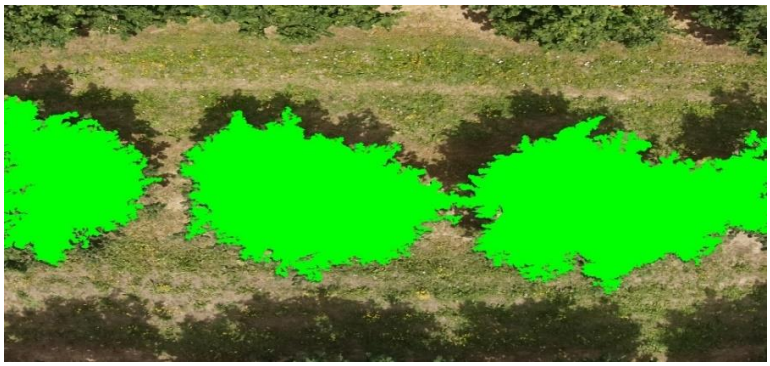


Figure 3.20: Drawn contour around vegetation

This next step aims to estimate the horizontal boundaries around the hazelnut tree under consideration. Only horizontal boundaries estimation was considered and implemented in our project since we will discuss subsequently the different image processing methods applied to our separate datasets including the fact that horizontal boundaries estimation will be sufficient for most of the automated editing process that will be applied to the major portion of images studied.

In consideration of the fact that in the given dataset images the plant under consideration is almost always at the center of the image, the boundaries are estimated by firstly considering the middle of the image as our starting point. We apply 2 well defined left and right horizontal **sliding windows (SW)** with a window size and an overlap factor as parameters. We compute the total pixel count that belong to these windows given that the pixels under consideration belong to the contour estimated in the previous step. Once all the considered windows are computed, we find the first minimum from the center on each side and set the horizontal boundaries accordingly.

$$left\ bound\ =\ index(min\ (SW_{left}))$$
$$right\ bound\ =\ index(min\ (SW_{right}))$$

In order to correct for rare cases where the contour drawn of the subject tree does not belong to the center of the image, we add a shifting starting point parameter aiming to let the user manually decide the position of the horizontal starting point of the algorithm.

We finally obtain two bounds that will delimit our tree horizontally, so we can proceed to the next step of our process.

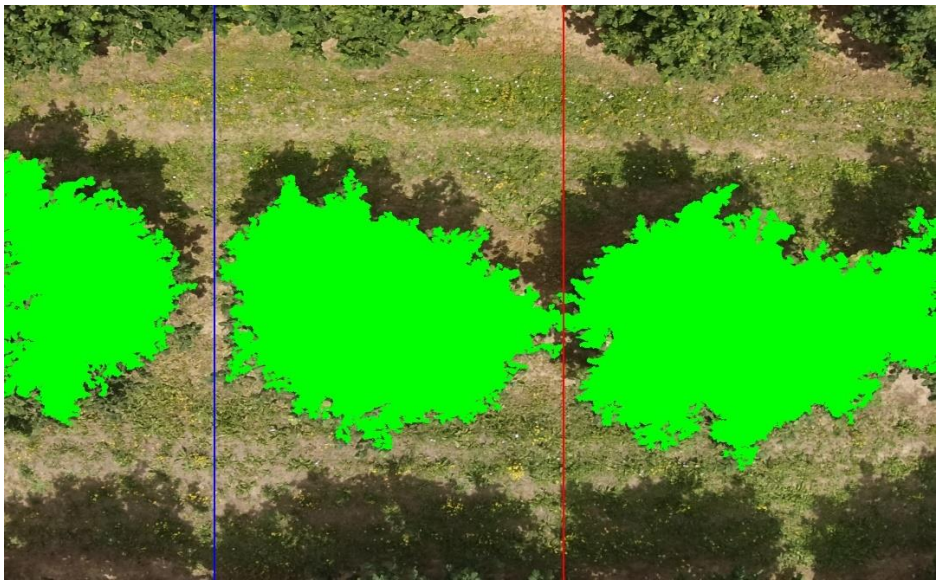


Figure 3.21: Added boundaries

The output of the vertical tree boundaries estimator will become the input of the mask extractor. In this step we will be able to isolate the subject hazelnut plant by extracting the NDVI mask defined by the vertical boundaries that were estimated in the previous step.

Figure 3.22: Extracted tree mask

In the final step, we use the NDVI mask extracted as our reference for the now isolated plant, we save the pixel coordinates of the mask in a specific file for every plant. The saved pixel data points coordinates will be used in the next chapters of our project in order to extract the mask of subject plants in all multispectral bands.

Figure 3.23: Contoured tree

3.6 Application of Semi-automated Editing

The Semi-Automated Editing can be described in 2 steps.

1 – Automatically applying vegetative Indices in order to delimit and extract the vegetation in the bounded area.

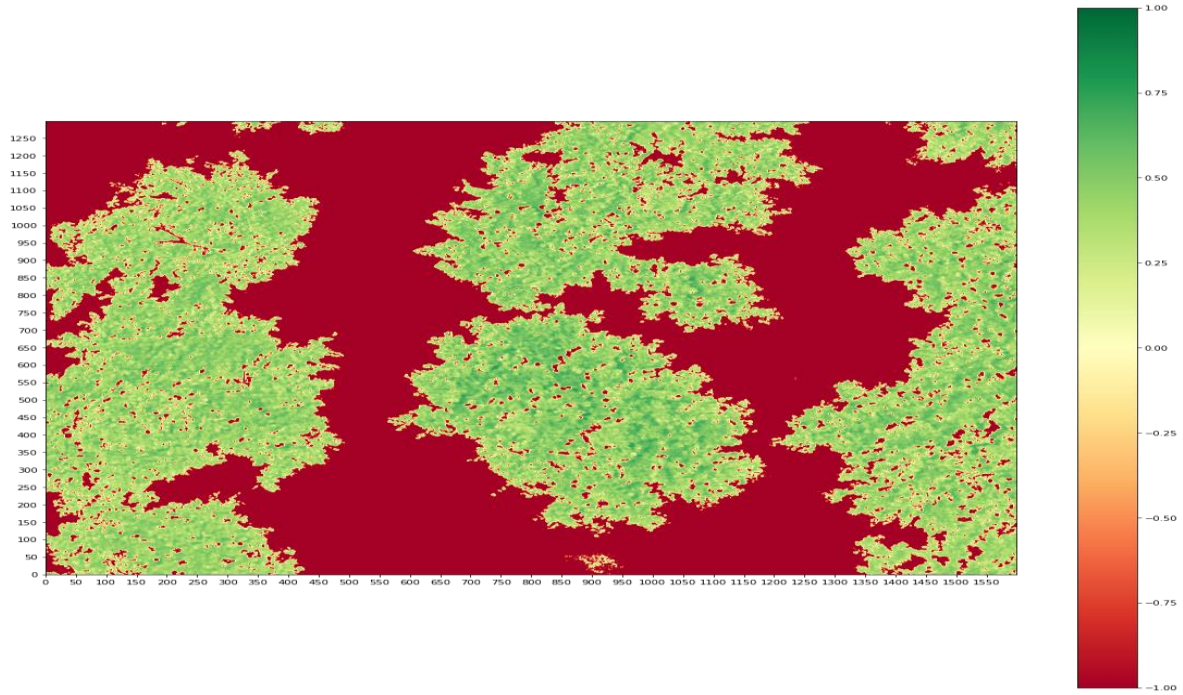
2 – Manually defining vertical and horizontal boundary lines.

In order to highlight the vegetation present in the image, in the first step, the NDVI transform is applied. The chosen threshold for the NDVI transform will assume values of 0.2 or 0.3 depending on the images. Values below that threshold will be set to -1 to enhance the contrast between vegetation and non-vegetation. This step is necessary in order to facilitate the manual definition of the boundaries. With only vegetation emphasized compared to the rest of the image we will be able to define the horizontal and vertical bounds much more accurately.

After displaying the image, we will be able to enter the minimum and maximum boundaries (the edge borders) horizontally and vertically, defining a rectangular shape around the plant under consideration. The defined boundaries will be used in the next step in order to delimit the contour of the plant present in the bounded area.

Once thing to note here is that trees can have all sort of shape geometry, and when in some images, some trees tend to overlap with one another, it will be rather difficult to well define the contour of the subject tree. As a consequence, at first glance, strictly rectangular shaped bounds around the tree will be assumed as non-ideal, however, as we will eventually see, given the nature of the datasets containing a lot of images where trees are generally massively overlapped and intertwined, will make the overall plant contour rather challenging to outline even for the human eye (**Farigliano dataset**). The best approach will be to remain conservative in delimiting the tree boundaries, so we don't mistakenly include vegetation (neighboring trees or grass) in our analysis that do not belong to the plant under consideration. That being the case, we will choose to follow the rectangular based approach for manually delimiting the boundaries of the plant under consideration.

After defined the bounds of the tree in question, we apply a contour around the tree. Then we follow by extracting the pixel coordinates points belonging to that contour and saving the obtained data points in a specific file for every plant.



```
Enter the left Bound: 550
Enter the right Bound: 1220
Enter the low Bound: 130
Enter the high Bound: 770
```

Figure 3.24: Manually entering the boundaries

The automated and semi-automated process can be illustrated in the following diagram:

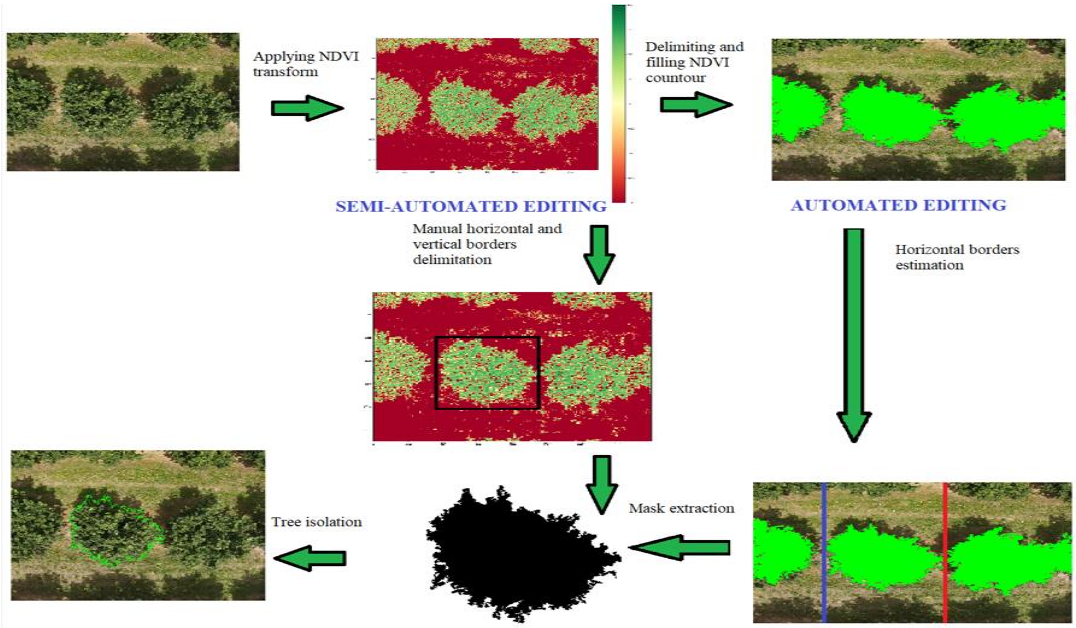


Figure 3.25: Automated and semi-automated editing processes

Having described the implementation of the different possible image editing methods that can be applied to our dataset. We proceed by presenting the different photographic contexts in which the images presented by each dataset were taken.

3.7 Processing the Carrù Dataset

As we have seen in the previous chapter, the Carrù dataset consists of 381 images taken by a DJI drone. These images do not exclusively contain only the plant under consideration as their subject, conversely the images include additional vegetation like grass or small bushes, and they additionally contain other neighboring trees (figure 3.2, 3.3).

Another apparent feature of the Carrù dataset is the discrepancy in the sun angle between the different shooting sets (figure 3.2, 3.3). While the 30/05/2022 images

appear to be taken when the sun is at zenith, the 22/06/2022 and 15/07/2022 shootings have significantly more shadows around the vegetation. The inconsistency in the exposure balance in the 22/06/2022 and 15/07/2022 images can lead to loss of details and eventually loss of information. However, regardless of the lighting conditions, the NDVI transform applied, seem to detect decently the contour of the trees.

We can observe a particular feature of the Carrù dataset that stands out, is that all the plants that appear in the images, regardless of their significance to our analysis, appear to fall horizontally in the image. For Instance, looking at figure 3.26 below, we notice that the plants are spread out from left to right. We notice in that image the absence of plants that are vertically spaced, therefore we can conclude that any vertical line drawn on the image can generally belong to a single tree or to no tree at all. This statement holds true of the entire Carrù dataset. Despite the existence of some overlap between neighboring trees present in the dataset, we can cut vertically on both sides of the plant under consideration, thus isolating it from the rest of the trees and vegetation present in the image.

Figure 3.26: Carrù dataset trees general layout

Considering the lay out of the Carrù dataset images, we can use the automated process described previously in order to isolate our subject tree from the rest of the

vegetation in the image. As illustrated above, the automated editing process will apply the NDVI transform to the image where a masking of the vegetation will be applied. Subsequently will be able to estimate the vertical boundaries at each end of the tree. The two boundaries will cut the image vertically where the NDVI transformed contour will be at a minimum (where the tree in the middle of the image ends). Finally, we will obtain the isolated and contoured plant under consideration.

The **automated editing** process will be applied to all the Carrù dataset:

Carrù 30/05/2022 shooting, Carrù 22/06/2022 shooting and Carrù 15/07/2022 shooting.

3.8 Processing the Farigliano Dataset

The Farigliano dataset consists of 174 images taken by a DJI drone. Compared to the Carrù dataset, the Farigliano dataset images contain much more overlapping and intertwined neighboring trees and other vegetation, making a totally automated process for isolating the subject plant in each image seem impossible, even for the human eye, the majority of the drone images taken of the plants in the Farigliano dataset will pause a challenge to properly define tree contours (figure 3.5, 3.6).

Analogous to the Carrù dataset we also notice different shadows in the different shooting sets. While the 03/06/2022 images appear to be taken when the sun is at zenith, we can observe that in the 22/06/2022 and 15/07/2022 shootings have significantly more shadows around the vegetation. However, similar to the Carrù

dataset, regardless of the lighting conditions, the NDVI transform applied, seem to detect decently the contour of the trees.

However, in contrast to the Carrù dataset, the layout of the trees in the images taken in the Farigliano dataset, do not a follow a horizontal pattern, or any particular pattern for that matter, consequently, in order to isolate the subject tree, it will be necessary to manually define horizontal and vertical boundaries.

Due to the complexity and heterogeneity of the Farigliano dataset. We will apply a semi-automated editing process, where we manually define a rectangular shape around the tree under consideration before automatically proceeding in delimiting the contour using the NDVI transform of the bounded image.

The manually applied boundaries will be strictly chosen in order to minimize the inclusion of neighboring vegetation in our NDVI transformed contour, owing to the nature of the dataset. This decision will therefore prevent the introduction of erroneous data to our defined image, even at the expense of leaving out portions of the subject tree that might not be considered, like branches or leaves that overlap from neighboring trees (figure 3.2, 3.3).

The **semi-automated** editing process will be applied to all the Farigliano dataset:

Farigliano 03/06/2022 shooting, Farigliano 24/06/2022 shooting and Farigliano 16/07/2022 shooting.

Chapter 4

4.1 Image Subdivision

In order to properly assess the health status of a plant, botanists generally do not base their evaluation on the plant as whole, rather, they try to discern and isolate specific parts of the plant that might appear to be sick. Typically, given that a portion of a plant is sick, does not necessarily imply the compromise of the whole plant. This will enable the identification and isolation of sections of the plant that might be suffering and act exclusively on them.

From this practice, we can try to apply the same evaluation process to our dataset, dividing the image of the plants into subsections on which we can apply a label either “healthy” or “sick”.

In the first step of this process, we should define the criteria for the image subdivision. Two approaches will be explored:

- **Grid subdivision**: Dividing the image of the plant into an ***N*x*N*** grid and applying labels accordingly.
- **K-means subdivision**: applying k-means clustering to the image of the plant and then dividing it correspondingly, labels will also be also applied.

We must note that the labels that will be applied to the subdivided images will be provided by a team of **expert botanists on hazelnuts trees**. These labels will be indispensable for the application of machine learning methods in the next phase of our project.

4.2 Grid Subdivision

This first explored method involves subdividing the processed images from the previous step into an ***N*x*N*** grid pattern, where each section will contain one portion of the plant. This approach allows to identify which portions of the tree appear to be sick so the labels will be applied to each sub-plane accordingly. Proceeding with the next phase, the vegetative indices will be calculated and applied for each of the boxes using the pixels that belong to the subject plant computed from the previous step. The evaluated vegetational indices for each of the sub-planes along with their respective label will serve as the input to the machine learning algorithms adopted in the following step of this project.

Our first step in the image subdivision is to decide on the criteria required to crop and cut the image. After some experimentation with the dataset, we define an **upper threshold** = 300,000 pixels and a **lower threshold** = 100,000 pixels. These defined thresholds will be used to propose quasi-optimal rules that will be based on the delimited plant pixel count in order to extrapolate the quantity **N**, we obtain the following:

- If the contoured tree contains more than the **upper threshold**, we set **N** = 3 therefore obtaining a 3x3 matrix of the image (9 boxes).
- If the pixel count of the contoured tree contains falls between the **lower threshold**, and the **upper threshold**, we set **N** = 2 therefore obtaining a 2x2 matrix of the image (4 boxes).
- If the contoured tree contains more less than the **lower threshold**, we set **N** = 1 therefore obtaining a single box containing the plant (1 box).

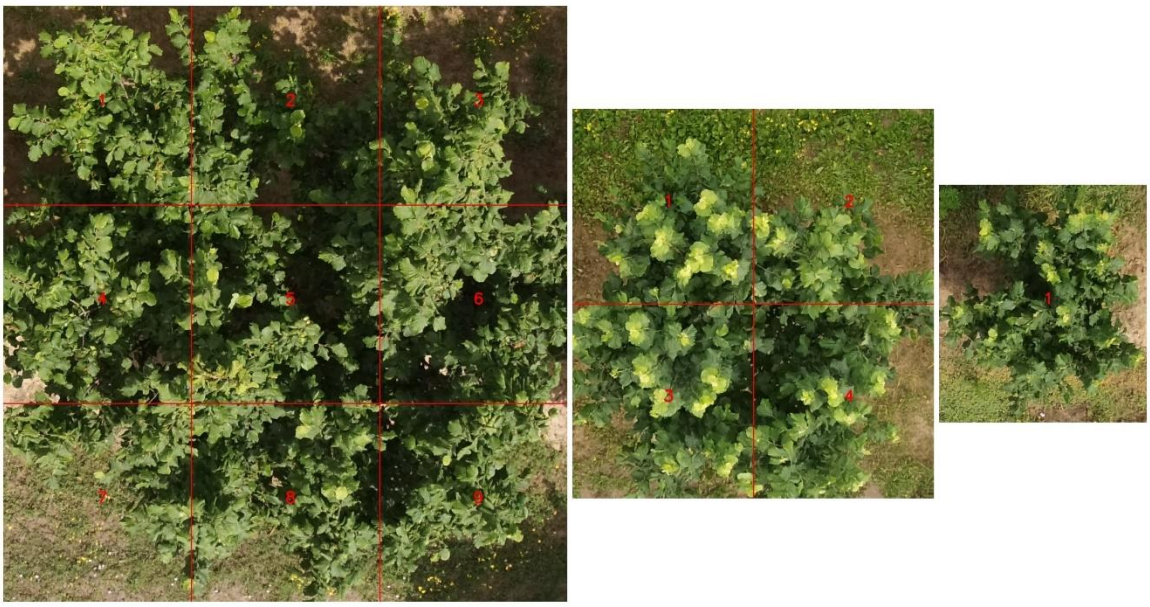


Figure 4.1: Grid subdivision for different *N*

The choice of these thresholds was based on the resolution of the obtained sub-planes. For instance, large boxes may include wider sections of the plant for which, the majority of the vegetation seems healthy, but a small part is sick, therefore defeating the purpose of the plant subdivision process. Conversely, small clippings of the plant will result in a low definition. This in turn can lead to inaccuracies and inefficiencies in the interpretation process of the images, like calculating vegetative indices and applying labels.

After having applied the grid subdivision process to the images, we proceed by numbering each sub-plane of the image starting from the upper left corner and ending at the bottom right corner as shown in figure 4.1.

Having divided the image and labeled its subsections, we crop the image around the plant, then we proceed to add a title as follows: "*Pianta*" followed by the

number of the plant in the dataset. Finally, the grid-subdivided image is saved in a folder where it can be examined and evaluated by a team of expert botanists.

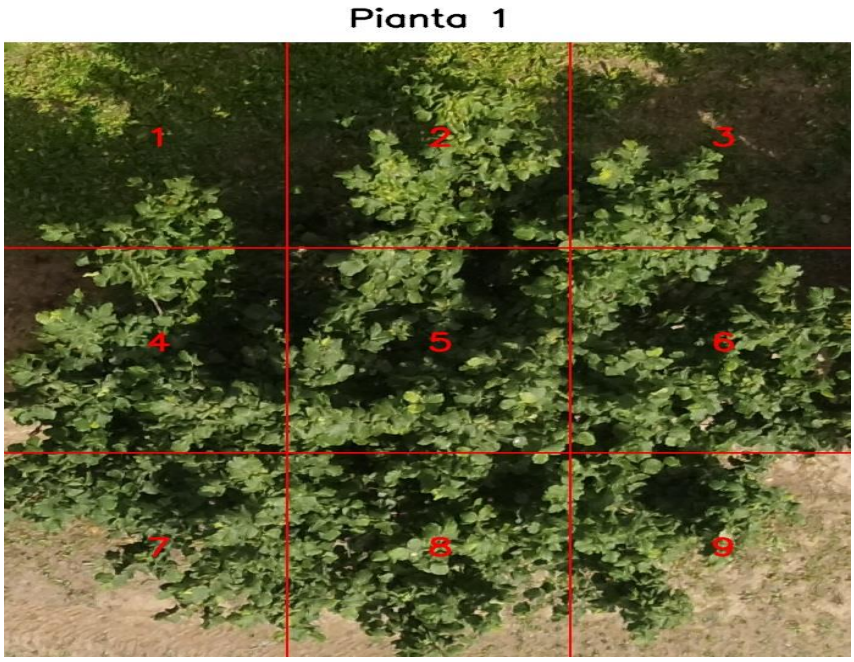


Figure 4.2: First plant of Carrù 22/06/2022 subset grid-subdivided

This procedure will be repeated for all the plants. Subsequently, we created an *.xls* sheet where the botanists can insert the “healthy” (0) or “sick” (1) labels for each sub-plane for every plant in the dataset.

	A	B	C	D	E	F	G
1			Carrù 15/07/2022 - Classificazione delle immagini delle piante				
2			Scrivi "0" per sano e "1" per malato				
3	Pianta1_1	1					
4	Pianta1_2	1					
5	Pianta1_3	0					
6	Pianta1_4	0					
7	Pianta1_5	0					
8	Pianta1_6	0					
9	Pianta1_7	1					
10	Pianta1_8	1					
11	Pianta1_9	1					
12							
13	Pianta2_1	0					
14	Pianta2_2	0					
15	Pianta2_3	1					
16	Pianta2_4	1					
17	Pianta2_5	1					
18	Pianta2_6	0					
19	Pianta2_7	1					
20	Pianta2_8	0					
21	Pianta2_9	0					
22							

Figure 4.2: First 18 entries of Carrù 15/07/2022 subset

The last step of our process is to save the pixel data for each of the sub-planes inside a specific folder belonging to every plant. This procedure will allow the computation of the average vegetational indices values for every specific subsection of the plant under consideration.

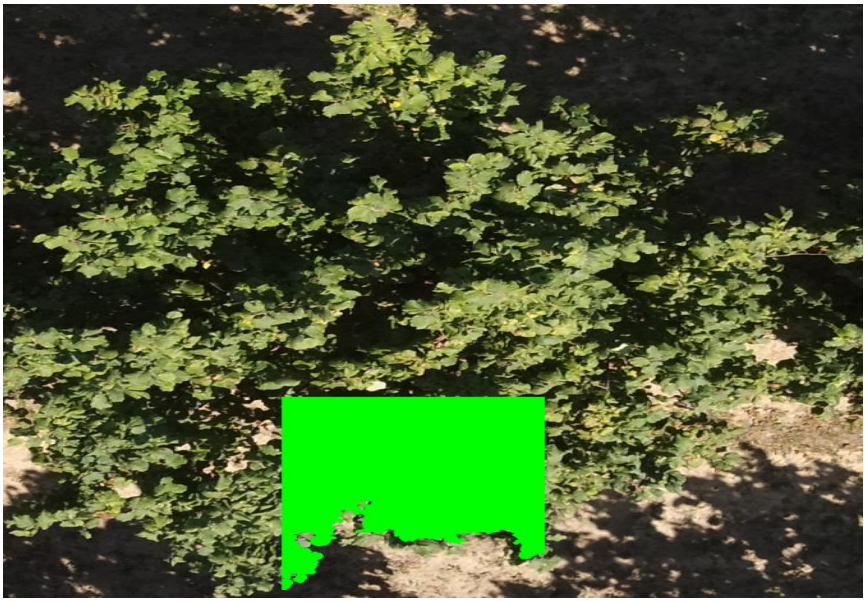


Figure 4.3: Highlighted pixels representing the subdivided region 8 in plant 33 from the Carrù 15/07/2022 subset

4.2.1 Problems with Grid Subdivision

The major drawback of a grid-based tree subdivision is the absence of conformance to the plant geometry. A strictly based rectangular subdivision will not account for the various distinct features of a tree. For instance, we can have multiple branches with different health status belonging to the same divided sub-plane and thus they must be holding the same label. This can prove inaccurate for the labeling process, and moreover, can lead to imprecise data input in the machine

learning phase of our project, as the “healthy” and “sick” parts of the subregion may cause the respective vegetative indices values to be averaged out (figure 4.4).

Figure 4.4: Example of subregion (4) containing healthy and sick leaves

On the other hand, a subsection might contain very little to even no vegetation at all. The latter can cause a major discrepancy in the pixel-to-pixel “importance”. For every subsection the vegetative indices will be computed pixel-by-pixel using the pixels that belong to the subject plant, then averaged out over the entire sub-plane. This indicates that the average vegetative indices computed for each section along with the respective labels, will be considered independently from the number of plant-pixels included in these subsections. This will lead to some pixels having more significance and influence over other ones and will generally cause an inaccurate model representation of the plant under consideration.

In order to mitigate this problem, we can for instance set a lower bound for the minimum plant-pixel count necessary to belong in a specific sub-plane so that section can be considered in our testing data in the next phase of our project (let’s say 1000 pixels belonging to that plant).

On the other hand, we can explore a more accurate tree representation model by applying the **k-means clustering** method.

4.3 k-means Clustering Subdivision

k-means clustering is a data partitioning method and a combinatorial optimization problem. It is a technique for vector quantization that performs the partitioning of *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This lead in a partitioning of the data space into **Voronoi cells**.

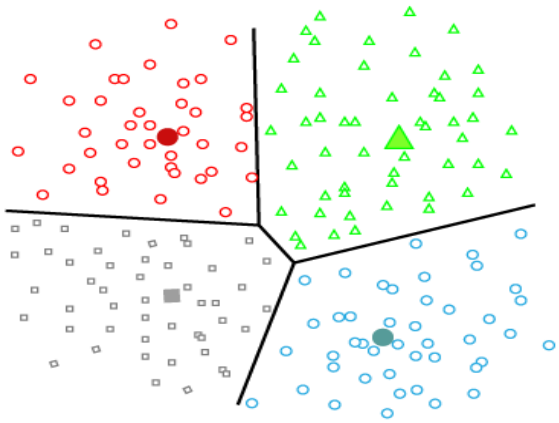


Figure 4.5: Example of k-means clustering

Voronoi cell are a region that belong to a Voronoi diagram that consists of dividing a plane into regions close to each of a given set of objects. These objects consist of many points in the plane where for each point (or seed) there is a corresponding region, called a **Voronoi cell**. These cells consist of all points of the plane closer to the seed than to any other.

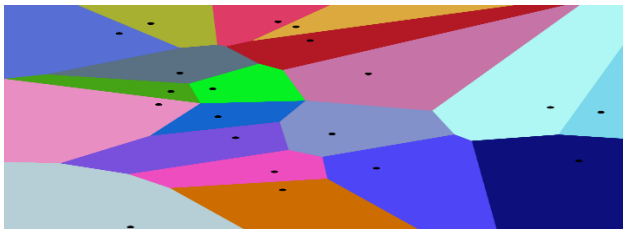


Figure 4.6: Example of Voronoi cells

The goal of the k-means partitioning method is to minimize some particular function. We consider the distance of a point to the mean of the points of its cluster, the function to be minimized is the sum of the squares of these distances. K-means clustering is used in ***unsupervised machine learning*** where it can classify unlabeled data into a predetermined number of clusters based on similarities (*k*). Despite that the k-means clustering problem is computationally difficult (NP-hard), there exist efficient heuristic algorithms converge quickly to a local optimum.

Given a set of observations (*x1, x2, ..., xn*), where each observation is a *d*-dimensional real vector, k-means clustering aims to partition the *n* observations into *k* ($\leq n$) sets S = {*S1, S2, ..., Sk*} so as to minimize the within-cluster sum of squares. The objective is to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

where $\boldsymbol{\mu}_i$ is the mean of points in S_i.

In view of the k-means clustering method that just described above, we can apply k-means clustering to our case so we can subdivide the contoured trees into *k* clusters, we obtain equivalently:

- The n observations are the pixel coordinates that belong to the tree under consideration.
- k clusters are the number of the subdivided regions to be considered for our tree.
- S_i is an area (or a tree subdivided plane in our case) that belongs to our subdivided image.

Analogous to the grid subdivision-based approach, the k-means method allows to identify which portions of the tree appear to be sick by dividing the tree into clusters (sub-planes) so labels can be applied accordingly. Vegetative indices will be then calculated and applied for each of the areas using the pixels that belong to the contoured plant computed from the previous step. The evaluated vegetational indices for each of the sub-planes along with their respective label will be used as the input to the machine learning algorithms applied in the following step.

K-means clustering will be implemented in our project using ***sklearn.cluster*** from the ***scikit-learn*** library in ***Python***

We can illustrate the k-means clustering method using the following function:

$$\boldsymbol{S} = \text{k-means}\ (k, n)$$

In the first step is to decide on the number of clusters k required for every tree. We will describe the k as the rounded ratio between the total number of pixels that belong to the contoured tree, and a predetermined cluster size, we can write it as follows:

$$k = \text{round (total number of contour pixels / cluster size)}$$

$$k \text{ belongs to } \mathbf{Z}^{+}$$

Following some experimentation with the dataset, we define the cluster size to be 100,000 pixels. Naturally the actual size of the sub-planes will not be exactly that amount, however, the more clusters we have the more the subsections size will converge close to 100,000 pixels.

Having predefined *k,* we can proceed by dividing the contoured plant into *k* clusters, we obtain the following graph:

Figure 4.7: Example of k-means subdivided tree mask

For each cluster obtained, we will draw its respective contour. Subsequently we add numbering for each of the sub-planes, and finally the mask of the k-means clusters will be applied to our original image. We also include the plant number as a title.

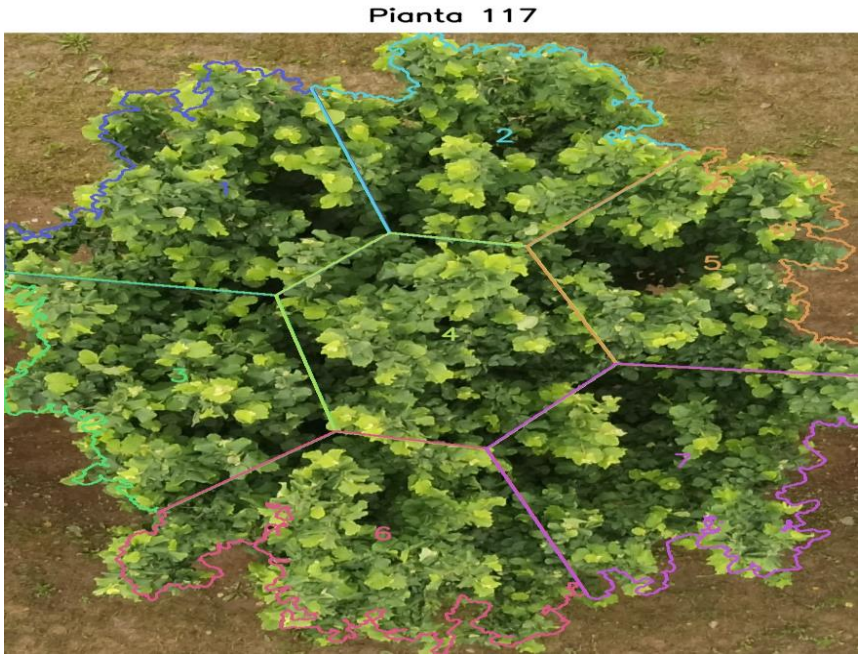


Figure 4.7: Contoured and numbered k-means divided tree

The following image is saved in a folder where it can be examined and evaluated by a team of botanists. This procedure will be applied for all the plants. An *.xls* sheet will also be created where the botanists can insert the "healthy" or "sick" labels for each sub-plane for every plant in the dataset. Finally, the pixel data will be saved for each obtained cluster inside a specific folder belonging to every plant. This procedure will consequently allow the computation of the average vegetational indices values.

4.3.1 Comparison with the Grid Subdivision method

Compared to the grid-subdivision method, the k-means clustering approach can be more accurate in creating a tree representation model that will better conform to the plant's shape. This can be attributed to the fact that using the k-means method

will result in the division of the tree into sub-regions that will not be limited to rectangular sub-planes, instead, k-means creates clusters using the pixels in the image that belong to tree as *observations*. The result will imply a subdivision that follows the tree's geometry, making it more optimal. Moreover, the clusters obtained will be very similar in size, resulting in a more balanced pixel datapoints distribution into the divided regions.

For the same image, a more balanced distribution means a lower number of subdivided areas compared to the grid-subdivision method, allowing fewer sub-planes to be evaluated by the botanists, hence making the labeling process more efficient.

Having considered both methods for image subdivision, our dataset will use a mix of both grid subdivision and k-means subdivision methods in creating subdivided images for the labeling process.

4.4 Labeling and Application of Vegetative indices

After having obtained the labels for our subdivided dataset, we proceed by applying the vegetative indices and other related metrics to the sub-regions.

We can see in the figure below the ratio between the healthy and sick labels applied to our dataset for the different shooting sets. We observe that with every shooting set the trees become more sick.

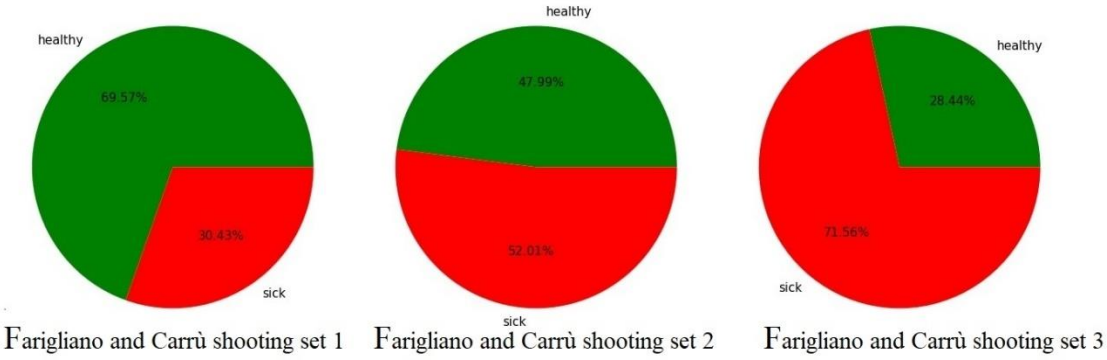


Figure 4.8: Healthy and sick labels distribution for every shooting set

However, we notice that the presented dataset across all shootings sets is fairly balanced.

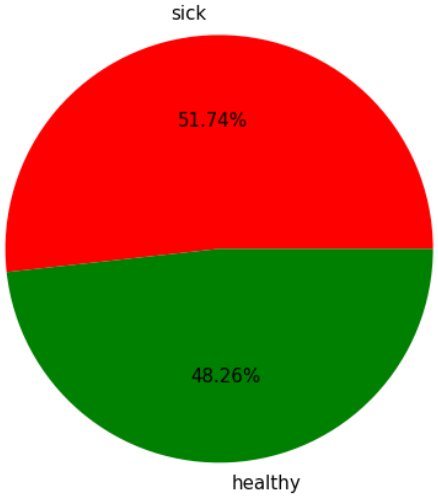


Figure 4.9: Healthy and sick labels distribution across all the dataset

The last phase required before the implementation of the machine learning algorithms is the calculation of the vegetative indices of our dataset.

We calculate pixel-by-pixel the vegetative indices for every subdivided region for each plant in all the datasets. The computed values will be stored in a ***.csv*** file, the respective label for every subdivided region will also be added.

We will also add for the dataset thresholds for the **NDVI, SAVI** and **GCI** metrics that will be used to differentiate healthy from unhealthy vegetation. Three more metrics will be created for each of the vegetative indices:

- ***Healthy mean***: is the mean of the total number of pixels having a value above the threshold
- ***Unhealthy mean***: is the mean of the total number of pixels having a value below the threshold
- ***Healthy Ratio***: the ratio between the number of pixels having a value above the threshold and the number of pixels having a value below the threshold

After some experimentation we add the following thresholds: $Th_{NDVI} = 0.5$, $Th_{SAVI} = 0.25$ and $Th_{GCI} = 0.6$

These added metrics will help highlight the status health of the vegetation and the amount of it present in the images compared to other objects.

We proceed by calculating the correlation matrix between the different attributes of the image.

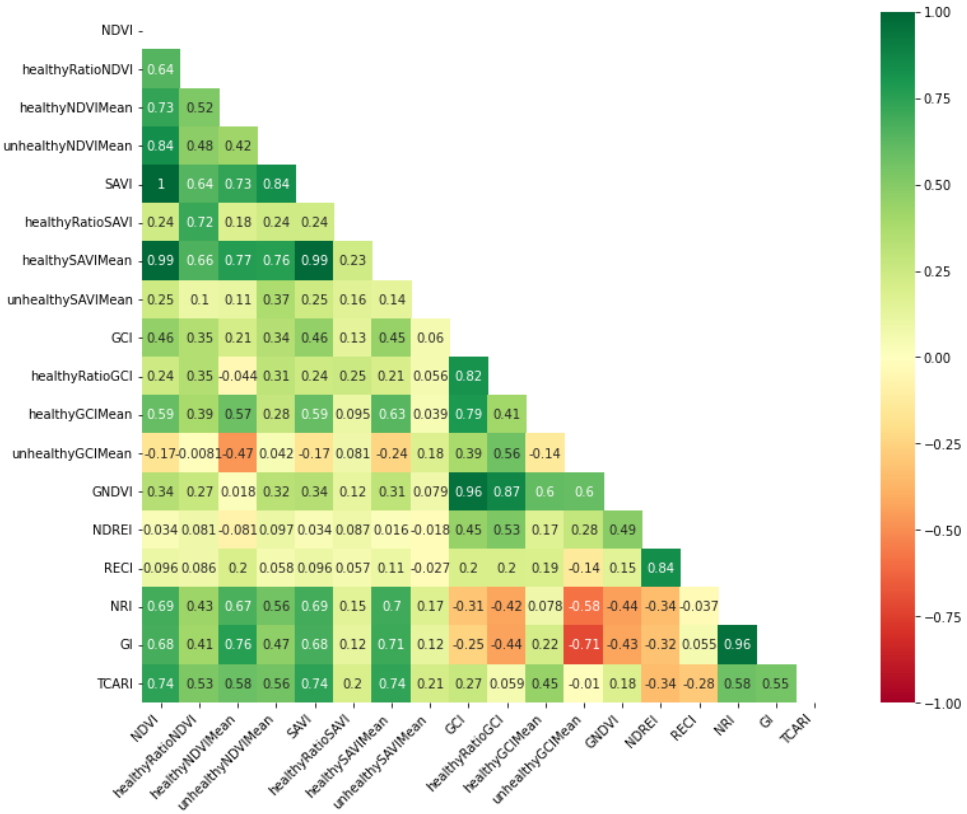


Figure 4.10: Vegetative indices correlation matrix

We can notice from the table above (**figure 4.9**) that some of the vegetative indices attributes have a strong correlation between them. For instance, **GCI** and **GNDVI** seem to be strongly correlated with a correlation of 96%. On the other hand, we can see different attributes showing a low correlation, like for instance the **NRI** and **RECI**, having only -3.7% correlation.

Below we can see the distribution between the different calculated vegetative indices attributes of the image relative to their labels.

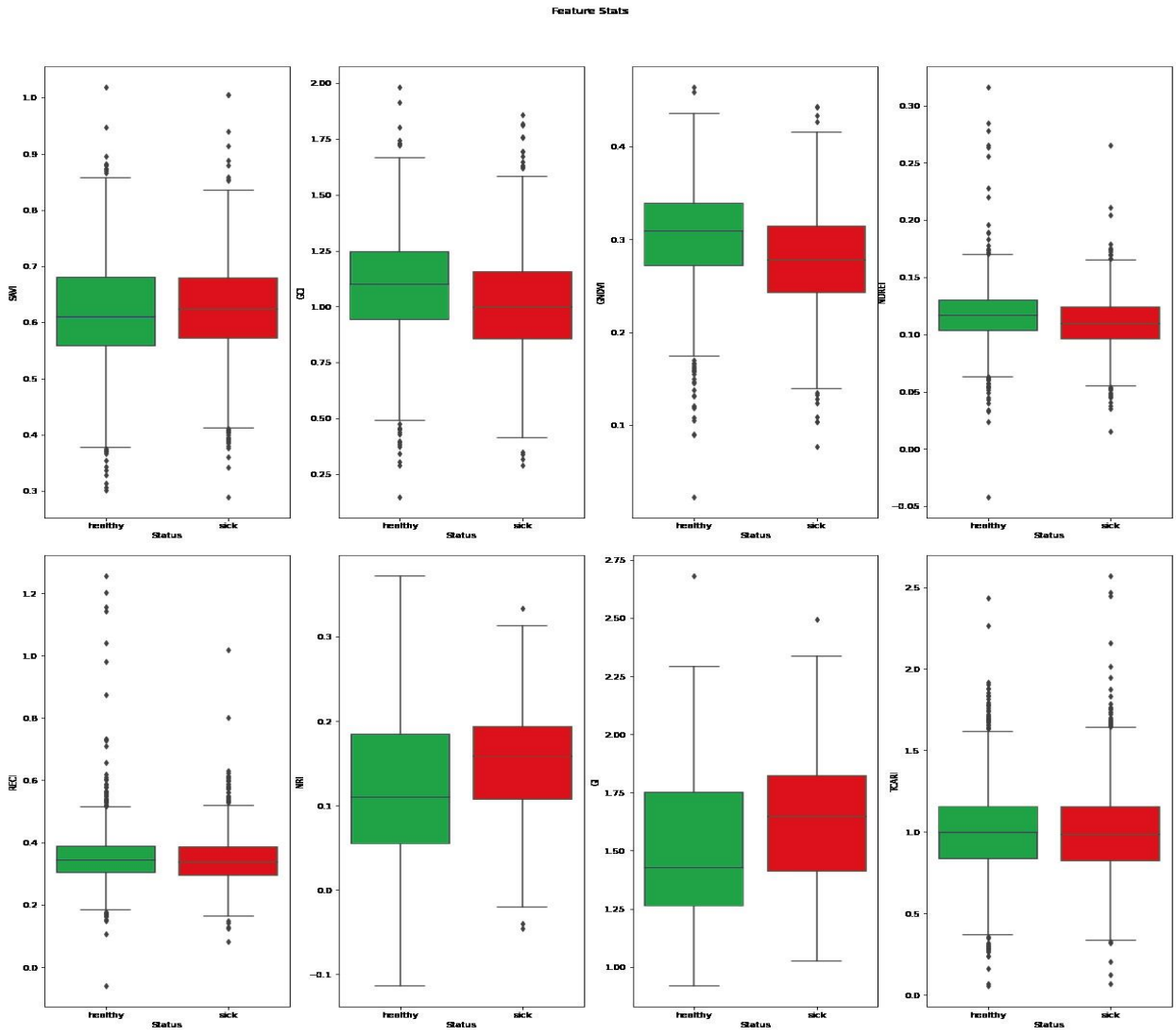


Figure 4.11: Vegetative indices values distribution for healthy and sick labels

For the **GCI, GNVI** and **NDREI** metrics, we can clearly observe a general trend that shows higher values for healthy plants and lower values for sick ones. In the contrary, for the **NRI** and **GI** metrics the sick plants have higher values, this can be related to the fact that the lower the values are in **NRI** and **GI**, the higher the reflectance of the leaves of the plants (**2.1.6, 2.1.7**). However, the **SAVI, RECI** and **TCARI** indices, the distribution is the same for both healthy and sick plants

for the following reasons: **SAVI** is not a good indicator of the plants health, rather its main purpose is to highlight vegetation in images compared to the ground or soil. **RECI** and **TCARI** are the only vegetative indices used that include the Red-Edge in the calculation of their formulas. Both having the same distribution for healthy and sick plants signify that the Red-Edge band for our case holds little to no valuable information regarding the health status of the plants.

It was decided to remove the **SAVI, RECI** and **TCARI** from our calculation while keeping the rest, as their number is not very excessive for the applied machine learning algorithms.

Chapter 5

5.1 Application of Classical Machine Learning Models

Machine learning is a branch of computer science which uses methods and algorithms in order to mimic the way that the human brain learns, gradually improving its accuracy. The used methods can use data to learn and improve its performance on a set of tasks. Machine learning is seen as part of artificial intelligence.

In the past decades, technological advances in storage and processing power have led to breakthroughs machine learning research. These advancements allowed the development of a variety of new tools that can be applied to solve data classification problems.

The baseline of our project is to be able to classify a tree into two categories: “healthy” or “sick”. This classification will be decided by data provided based on multispectral images shot in multispectral bands. Given that the classified images can fall into two categories, the problem to solve, is a ***binary classification*** problem.

We recall the steps of our project in the following diagram.

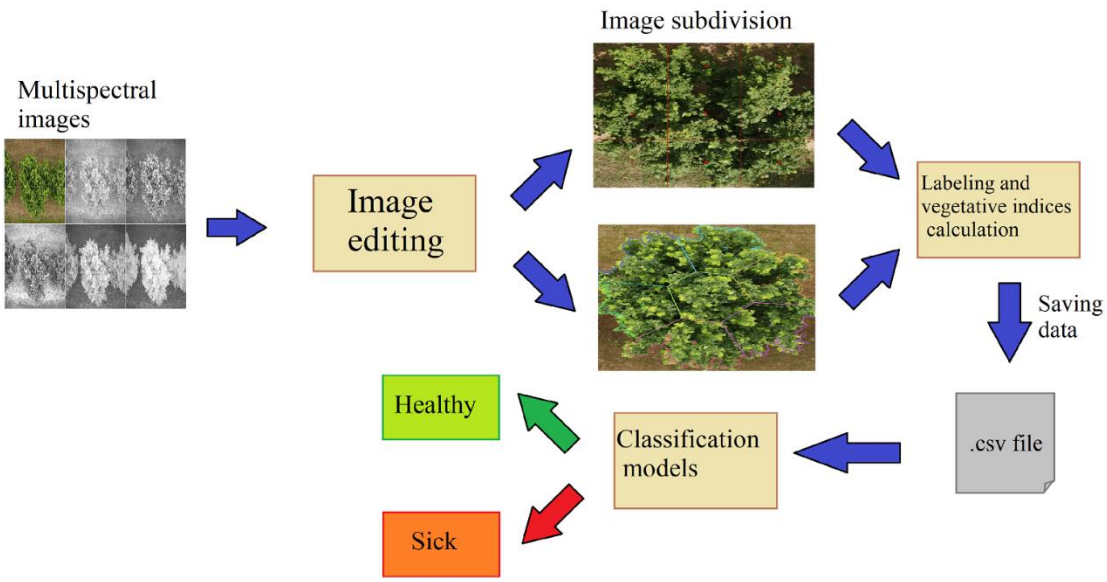


Figure 5.1: Classification process diagram

For this chapter we will consider classical machine learning algorithms, of which, many can be applied to solve our problem. However, in this chapter we will consider three ***supervised leaning*** algorithms that fit well our case.

1 - Random Forest
2 - K-Nearest Neighbors
3 - Logistic Regression

The first step of our process is to divide the data into training set and a testing set. We have decided that given the size of our data, an 80% of the data will for training and 20% will be for testing. The partitioning of the data will be achieved using ***stratified random sampling***. For each of the used algorithms, the best combinations of ***hyperparameters*** will be achieved through parameter optimization

methods such as ***Random Search*** and ***Grid Search***. Moreover, each model will utilize a ***k-fold cross validation*** process in order to validate the data.

It is necessary to define a confusion matrix table for the evaluation of the machine learning algorithms implemented. The latter means that it will be required to choose model performance criteria that will create a clearer understanding about the machine learning models performance.

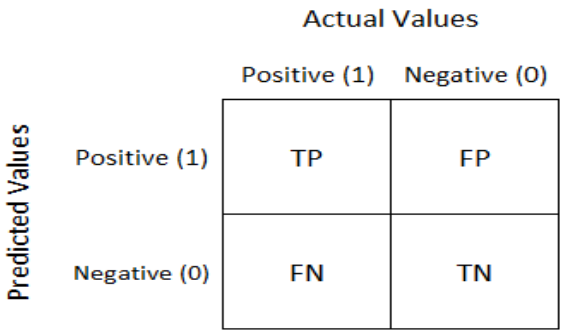


Figure 5.2: Confusion matrix

The criteria required for evaluation will be:

- Accuracy
- Precision
- Recall
- F1 score

Accuracy:

Accuracy is defined as the following:

$$Accuracy = \frac{Total\ correct\ guesses}{Total\ guesses} = \frac{TP + TN}{TP+TN+FP+FN}$$

Accuracy is simply the number of right guesses expressed as a portion of all the guesses. The main shortcoming of accuracy in measuring performance and that will become obvious when working with biased datasets. For instance, in our case, we have all the labels being "healthy" labels and only say 1% of them are actually "sick" if the model then decides to only output "healthy" predictions and absolutely zero positive guesses the model will achieve 99% accuracy.

Precision:

We can express precision as the following:

$$Precision = \frac{Correct\ positive\ guesses}{Total\ positive\ guesses} = \frac{TP}{TP + FP}$$

Precision is a measure of how well the model guessed the label. It is calculated by dividing the number of *correct positive* guesses by all the *positive guesses*. The objective of a model that optimizes for this metric would be to make as few mistakes as possible when guessing the positive labels. Moreover, the precision score penalizes the model with the score of 0 if it fails to guess any positive labels. However, the main issue regarding the precision score is that it doesn't consider any of the negative labels.

Recall:

Recall is expressed in the following equation:

$$Recall = \frac{Correct\ positive\ guesses}{All\ positive\ labels} = \frac{TP}{TP + FN}$$

Recall measures how many positive labels are guessed correctly out of the total number of positive labels available in the dataset. The objective is therefore to try and find *every positive*. However, in some extreme cases recall can label everything as positive and this will result in no *false negatives*.

F1-Score:

F1-Score is defined in terms of *Precision* and *Recall*:

$$F_1 = \frac{2 * precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

F1- Score is defined as the harmonic mean between precision and recall. F1-score can check on how good the quality of the predictions is and how completely and accurately the labels were predicted. The F1-score serves as a good indicator of how the performance of model as it tries to maximize both precision and recall.

5.2 Random Forest

The first model used in this project will be based on the Random Forest algorithm. Random Forest is a supervised machine learning classification algorithm that combines the output of multiple **decision trees** to achieve a single result.

Decision trees is a learning algorithm that make use of tree-like model of decisions. This tree structure has lead nodes that represent different classifications.

Each node can be seen as a class that can be connected to a parent node and possibly having multiple child subclasses. The dataset is split recursively using the decision nodes unless the node is a leaf node. The decision tree then finds the optimal split by maximizing the entropy gain. If a data sample satisfies the condition at a decision node, then it moves to the child node that holds that condition. Finally, it reaches a leaf node where a class label will be assigned to it.

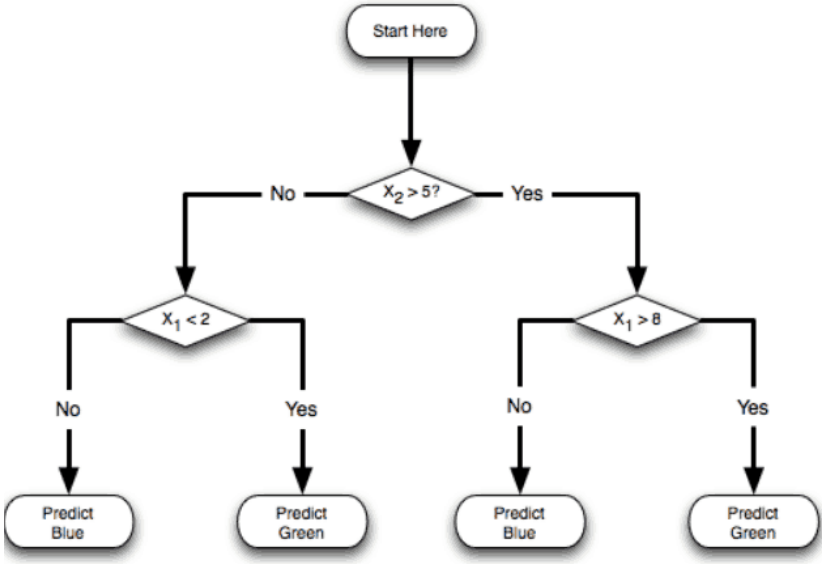


Figure 5.3: Decision Tree example

Random forests are a collection of multiple random decision trees. In order to create a random forest, the first step required to build new datasets from the original data where every dataset will contain the same number of features as the original one. The process adopted to create new data is called ***Bootstrapping***. The following step will be training the decision trees on each of the bootstrapped datasets independently. For every tree, a random subset of features for will be used for training. After evaluating each tree independently, the predictions of all the trees will be combined. This process of combining results from the multiple

decision trees is called ***Aggregation***. ***Bootstrapping*** helps the model to be less sensitive to the original training data as different parts of the data are used for every decision tree. This method will help reduce the correlation between the trees.

Regarding the ideal size of the feature subset, a good rule of thumb will be a number of features per decision tree which is close to the square root of the total number of features. $m \cong \sqrt{p}$

5.2.1 Implementation and Results

In order to optimize our model, it is necessary to fine tune the ***hyperparameters***. Due to the high number of hyperparameters, we used hyperparameter optimization through ***Random Search***. Random search is a process that utilizes random combinations of the hyperparameters in order to obtain the best solution for the built model. For each combination of the chosen hyperparameters, a model will be built, tested and evaluated using classification performance metrics (**precision**, **recall**, **f1 score**). Finally, the model with the highest performance will be adopted.

The results of the Random Forest model are presented below after the tuning of the hyperparameters (*k-fold* = 5*)*.

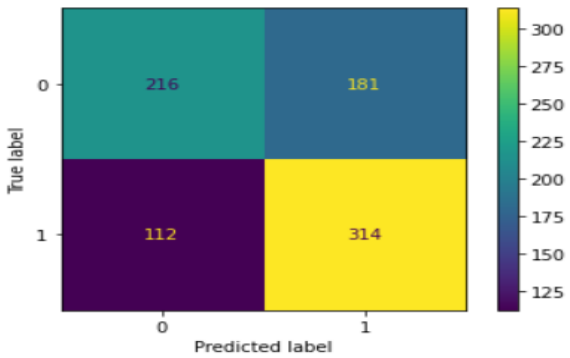


Figure 5.4: Random Forest confusion matrix

	precision	recall	f1-score	support
0	0.67	0.56	0.61	397
1	0.64	0.74	0.69	426
accuracy			0.65	823
macro avg	0.66	0.65	0.65	823
weighted avg	0.66	0.65	0.65	823

Figure 5.5: Random Forest results

5.3 Logistic Regression

Our third machine learning algorithm to be applied to our dataset is Logistic regression. Logistic regression is a special case of regression analysis, it is used to find the probability of an instance belonging to a particular class. For our binary classification problem, dichotomous variables with 0 and 1 or "healthy" and "sick" in our case, can be predicted by using a Logistic Regression.

Logistic regression creates a model based on the ***Logistic Function***. A logistic function allows only values between 0 and 1 to be possible. The Logistic function, like linear regression models are based on the weighted sum of the input variables. A logistic function will be applied on these features: The sigmoid function:

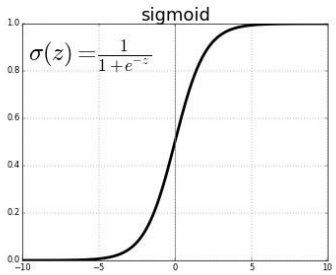


Figure 5.6: Sigmoid function

The training phase of the model will try to find the probability that an instance belongs to a certain class. Looking at the sigmoid curve in (figure 1.23) if the result of the logistic function is less than 0.5 than the instance belongs to the class '0' or if it is greater or equal to 0.5 than the instance belongs to '1'. Particularly, given a parameter vector β and considering N samples with labels 0 or 1:

- For samples with label 1: Estimate β such that *p(X)* is as close to 1 as possible.
- For samples with label 0: Estimate β such that *p(X)* is as close to 0 as possible.

Where *p(X)* € [0,1]

5.3.1 Implementation and Results

Like the previous model, it is necessary to fine tune the ***hyperparameters*** in order to optimize our model. Similarly, ***Random Search*** was used for this model.

The results of the Logistic Regression model are presented below after the tuning of the hyperparameters (*k-fold* = 5*)*.

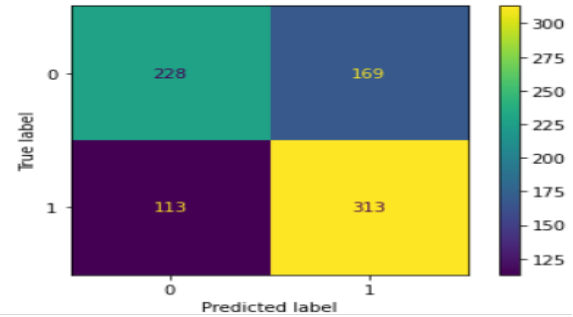


Figure 5.7: Logistic Regression confusion matrix

	precision	recall	f1-score	support
0	0.67	0.57	0.62	397
1	0.65	0.73	0.69	426
accuracy			0.66	823
macro avg	0.66	0.65	0.65	823
weighted avg	0.66	0.66	0.65	823

Figure 5.8: Logistic Regression results

5.4 K-Nearest Neighbors

K-nearest neighbors algorithm or KNN is a supervised machine learning method is used for classification and regression. In our case the k-nearest neighbors algorithm will be applied for classification.

For classification in k-nearest neighbors, an instance is classified by a vote of its neighbors, making the output is a class membership. The object is assigned to the class with the most common among its k nearest neighbors (k belongs to $\mathbf{Z}^{+}$). For instance, if k = 1, the instance will be assigned to the class of the closest neighbor.

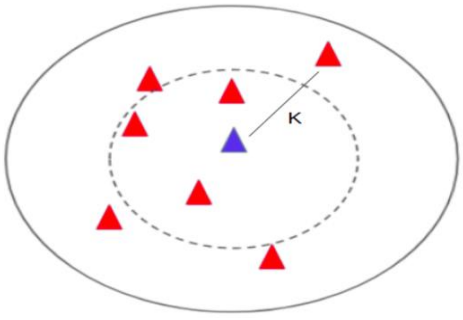


Figure 5.9: K nearest neighbor example

Weights can be assigned to the contributions of the neighbors, so that the nearer neighbors will contribute more to the average than the more distant ones. For example, a common weighting method consists in giving each neighbor a weight of 1 over the distance to the neighbor (1/d), making the weighing effect of more distant neighbors less prominent.

The training set of the k-nearest neighbor algorithm is taken from a set of objects for which the class is known. The classes in our binary classification can fall into "healthy" or "sick".

5.4.1 Implementation and Results

In view of a smaller number of hyperparameters ***Grid Search*** was used to optimize our model. While Random Search chooses randomly from the defined search space, Grid Search tests for all the possible combinations available in the hyperspace. Although Grid Search is more computationally expensive, it usually yields the best results.

The results of the k-nearest neighbors model are presented below after the tuning of the hyperparameters (*k-fold* = 5*)*.

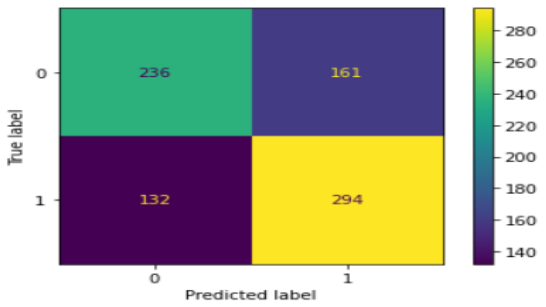


Figure 5.10: K-Nearest Neighbors confusion matrix

	precision	recall	f1-score	support
0	0.64	0.58	0.61	397
1	0.64	0.70	0.67	426
accuracy			0.64	823
macro avg	0.64	0.64	0.64	823
weighted avg	0.64	0.64	0.64	823

Figure 5.8: K-Nearest Neighbors results

5.5 Models Comparison

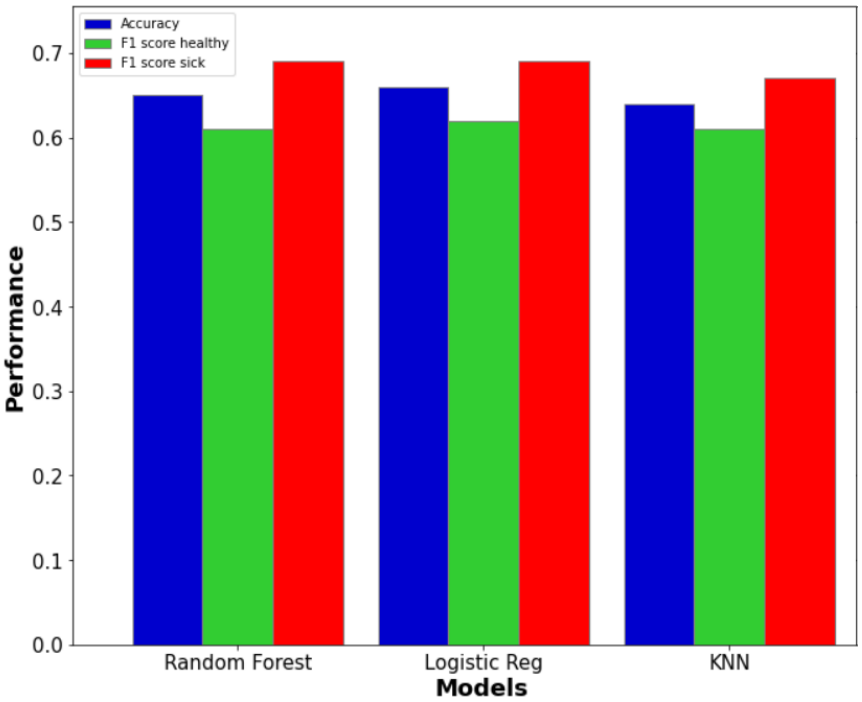


Figure 5.9: Model performance comparison

Looking at the implemented models results (figure 5.9), we can see that each of the models presented results that slightly differ to each other. While **Logistic**

Regression performed best with an accuracy of 0.66 and f1-scores of 0.62 and 0.69 respectively for “healthy” and “sick”, it took the most amount of time to run despite the use of the faster *Random Search* method for hyperparameter fine tuning. **Random Forest** follows directly next with an accuracy of 0.65 and f1-scores of 0.61 and 0.69 for “healthy” and “sick” respectively using also *Random Search* method for hyperparameter fine tuning. And Finally **k-nearest neighbors** came in third achieving an accuracy of 0.64 with f1-scores 0.61 and 0.67 for “healthy” and “sick” classes using *Grid Search* in order to tune the hyperparameters.

Chapter 6

6.1 Application of Convolutional Neural Networks

In this chapter we explore a new approach that moves away from traditional vegetative indices-based evaluation towards the application of artificial neural networks

The main problem with a classification based vegetative indices is the reliance on multispectral or hyperspectral imagery in order to calculate the necessary features required for the evaluation. This latter requires the use of expensive equipment and will result in the creation of a large amount of data for each photo taken. Moreover, misalignment between different multispectral shots can create inaccuracies in the calculation of the vegetative indices and will therefore introduce errors to our data.

Given our case, the labeling process was accomplished by a team of experienced botanists relying solely on the RGB photo of the plant. We can therefore assume for this step that all the necessary information required to evaluate and classify the images of the plants can be reliably found in the RGB photos. As a result, in this chapter we will apply artificial neural networks to our case that will be solely based on the information found in RGB photos. Particularly, the types of the networks that will be applied are ***Convolutional Neural Networks.***

Convolutional Neural Networks (or CNNs for short) are a subset of machine learning and a class of Artificial Neural Networks (ANNs). Convolutional Neural Networks is a specific type of network architecture for ***deep learning*** algorithms. CNNs are applied for image recognition tasks that make use of the processing of pixel data.

6.2 Deep Learning

Deep learning is a subset of machine learning. Neural networks are considered to be the backbone of deep learning algorithms. Compared to classical machine learning algorithms, deep neural networks consistently improve as the amount of data to be processed increases, making them better performers for large amounts of data. Moreover, the application of deep learning algorithms does not require the manual extraction of features from the data. In classical machine learning algorithms, the relevant features are extracted in a first step, then based of these features, the model will carry out the predictions. On the other hand, with deep learning, this step is skipped and instead the data is directly fed into the deep learning algorithm.

However, deep learning can have some disadvantages compared to classic machine learning algorithms. In particular, deep learning is generally requires a large amount of data to perform well. For instance, a typical deep learning neural networks needs at least a few thousand images to get reliable results. Moreover, deep neural networks are heavy on computational resources. They often require a high-performance GPUs and a good amount of RAM resources. Compared to classical machine learning algorithms, they take a considerable amount of time to run. In order to mitigate the aforementioned drawbacks, ***transfer learning*** and ***data augmentation*** can be employed, these two techniques will be explored in the application of the CNNs to our dataset.

6.3 Architecture of Convolutional Neural Networks

The general aim of CNNs is to process the information present in images. The architecture of the CNNs is generally based on biological neural networks that represent the cellular structure of the brain. This architecture tries to imitate the way a biological brain processes visual information.

CNNs follow an architecture similar to artificial neural networks, they are comprised of a node layers. The first layer is called ***input layer***, it is followed by one or more ***hidden layers***, and finally an ***output layer***. The input layer is used to insert the pixel data of the images, and the output layer will return the final result computed by the neural network.

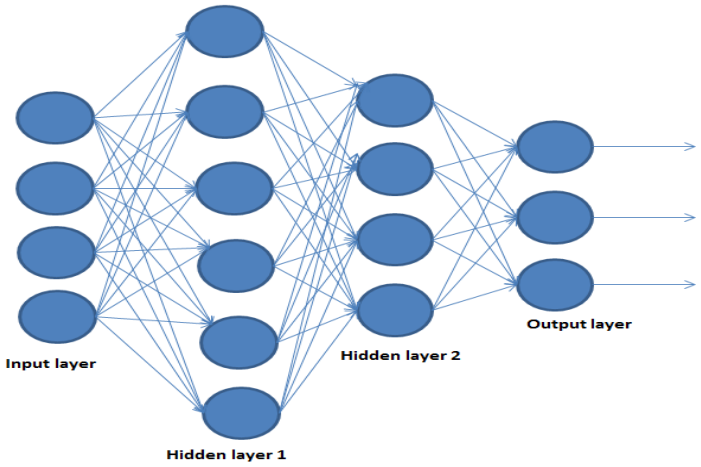


Figure 6.1: Example of a neural network with 2 hidden layers

In each layer, every node is connected to the nodes from the previous and the following layers and has an associated weight and a given threshold. If the output of any individual node is above the specified threshold, that node is activated, and in turn, will forward the information to the next layer of the network. The node architecture can be represented in the following diagram.

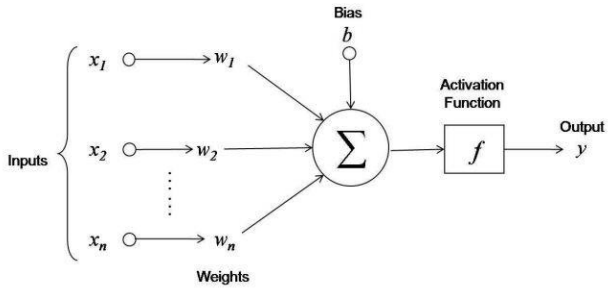


Figure 6.2: Neural network node

A neuron's activation status is determined by an activation function. The latter will determine whether the neuron's input to the network is significant during the prediction process. The Activation Function's main objective is to convert the node's weighted input sum into an output value that will be fed into the following hidden layer or used as output. One of the most used activation functions is the ***Sigmoid activation function*** shown in the formula below.

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

Convolutional neural networks and artificial neural networks differ primarily in the type of hidden layers used in the network. While the standard artificial neural network uses fully connected layers, CNNs, additionally, make use of partially connected layers. CNNs generally make use of 3 types of layers: ***Convolutional layers, pooling layers*** and ***fully connected layers.***

A convolutional layer is considered as the major building block of a CNN. It is comprised of filters (or convolution kernels) with parameters that will be learned throughout the training process. Each filter convolves with the image and creates

an activation map. This map can highlight the locations and strength of the perceived features in an input.

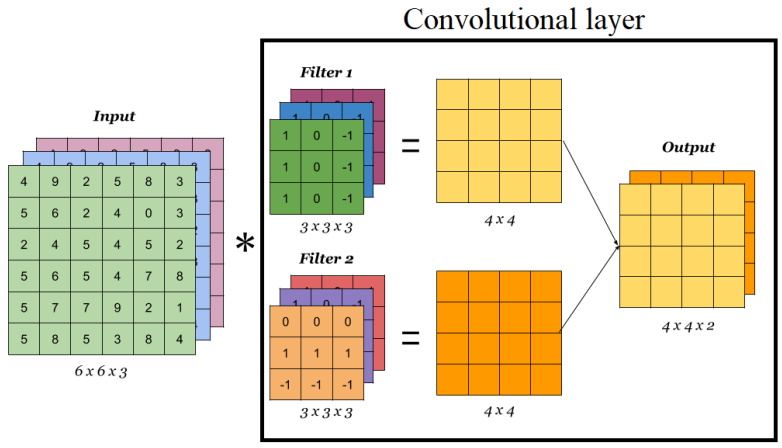


Figure 6.3: Example of convolutional layer

Pooling layers are used alongside the convolution process. Pooling layers compresses images summarizing the existence of features in individual feature map patches, pooling layers offer a method for down sampling feature maps. ***Average pooling*** and ***Max pooling*** are two popular pooling techniques that summarize a feature's average presence and its most active presence respectively.

6.4 Training process

The process of training a neural network can be characterize as an optimization problem. The basic idea is to optimize the weights of connections between the nodes in every layer of the network. Each connection between neurons starts with an arbitrary weight assigned to it during the beginning of the training phase. These weights would be constantly updated in order to reach their optimal values. There

exist a multitude of methods to update and optimize weights, the most widely used is called ***stochastic gradient descent*** (SGD).

The main objective of stochastic gradient descent is to minimize a given ***loss function***. SGD would be updating the weights with the aim of making the loss function as close to zero as possible. The loss function could be defined for instance as the ***mean squared error*** (MSE), which calculates the difference between the predictions made by the neural network and the true actual label value of the tested sample, averaged it out across the whole dataset.

SGD will try to minimize this error to maximize the model's accuracy in its predictions. Through a Cross-validation resampling process of repeatedly sending the same data into the model, is when the model will actually learn with SGD, updating the weights of the model accordingly until the loss function is at a minimum.

6.5 Transfer Learning and Data Augmentation

Before implementing the CNN the model, it is crucial to choose the correct architectures to be used. As stated previously, one of the main drawbacks of CNNs is the large amount of data required in order to achieve good performance and results. Consequently, it is necessary to apply CNNs architectures that can mitigate this problem. Given the case of our project, it was therefore decided to implement architectures that supports the prior mentioned techniques: ***transfer learning*** and ***data augmentation.***

Transfer learning allows the transfer the information learned from models already extensively trained, in order to solve a different problem that belongs to the same

domain. The main objective of transfer learning is therefore to adapt a pre-trained CNN that will be able to generally extract features within an image (object detection). This technique will be applicable to our case regardless of whether the used transferred model was trained on trees or not. This can be attributed to the fact that transfer learning is based on features that are commonly present in all images in general like horizontal and vertical edges, object contours, etc. allowing the usage of fewer computational resources and less training data required compared to the case where the CNN is trained from scratch.

Data augmentation is used to artificially increase the amount of training data. This includes making minor adjustments to the data or creating new data points using deep learning models. This technique will create additional datapoints to be used by a CNN, therefore increasing the size of the dataset and helping reduce overfitting. Different methods of data augmentation can be applied to the images such as: random rotation, flipping, padding, translating, etc.

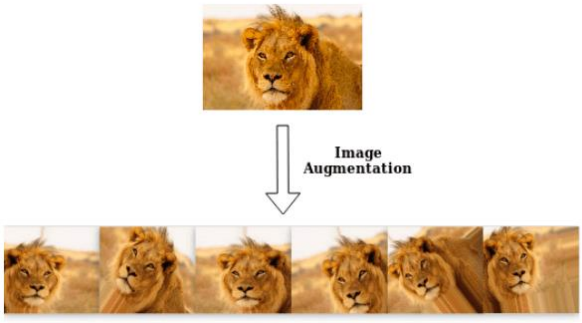


Figure 6.4: Example of data augmentation

6.6 Choice of Architecture

Considering the mentioned techniques above, architectures that provide both transfer learning and data augmentation will be implemented. These architectures will be adopted from the ***Keras*** library in Python.

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3

Figure 6.5: Keras architectures

Given the limited available resources in both computation terms and terms of time. We decided to implement *ResNet50* and *DenseNet121*.

We should also bear in mind that the optimization of the hyperparameters (batch size, epochs, learning rate, etc.) will require tremendous computational power therefore much longer running time, consequently, random search and grid search will not be applicated in this case. As for the transfer learning adopted in our model, we can apply ***fine tuning*** which allows CNN to learn on some or on all levels of the adopted basic model. This can be achieved by using a low learning rate in order to better optimize the model.

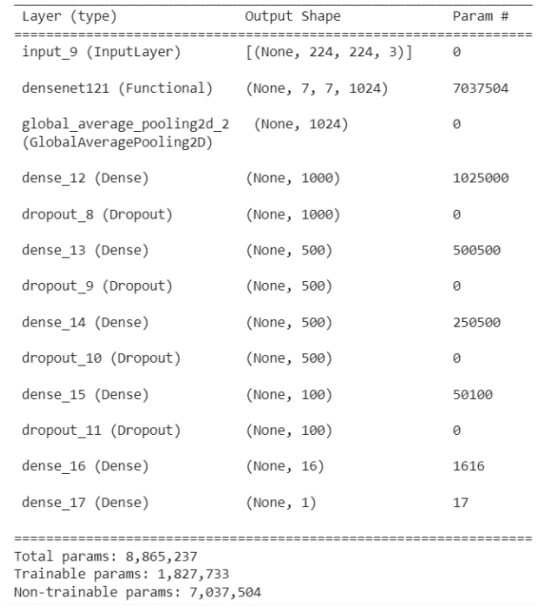
6.7 Implementation and Results

Before we proceed with the execution of our models, it is necessary to divide our dataset into different sets: ***training set***, ***test set*** and ***validation set*** accordingly:

- Training set: 70%
- Test set: 15%
- Validation set: 15%

We then proceed by defining the hyperparameters of our network (***Network architecture, Batch size, learning rate, epochs, etc.)***. Due to the complexity of this step due to hardware and time limitations, the hyperparameters will be manually tuned and optimized for every architecture considered.

We therefore implement the following architectures:



Layer (type)	Output Shape	Param #
input_9 (InputLayer)	[(None, 224, 224, 3)]	0
densenet121 (Functional)	(None, 7, 7, 1024)	7037504
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1024)	0
dense_12 (Dense)	(None, 1000)	1025000
dropout_8 (Dropout)	(None, 1000)	0
dense_13 (Dense)	(None, 500)	500500
dropout_9 (Dropout)	(None, 500)	0
dense_14 (Dense)	(None, 500)	250500
dropout_10 (Dropout)	(None, 500)	0
dense_15 (Dense)	(None, 100)	50100
dropout_11 (Dropout)	(None, 100)	0
dense_16 (Dense)	(None, 16)	1616
dense_17 (Dense)	(None, 1)	17

Total params: 8,865,237
Trainable params: 1,827,733
Non-trainable params: 7,037,504

Figure 6.6: DenseNet121 architecture

Layer (type)	Output Shape	Param #
input_12 (InputLayer)	[(None, 224, 224, 3)]	0
resnet50 (Functional)	(None, 7, 7, 2048)	23587712
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 2048)	0
dense_18 (Dense)	(None, 1000)	2049000
dropout (Dropout)	(None, 1000)	0
dense_19 (Dense)	(None, 500)	500500
dropout_1 (Dropout)	(None, 500)	0
dense_20 (Dense)	(None, 500)	250500
dropout_2 (Dropout)	(None, 500)	0
dense_21 (Dense)	(None, 100)	50100
dense_22 (Dense)	(None, 16)	1616
dense_23 (Dense)	(None, 1)	17

Total params: 26,439,445
Trainable params: 3,906,453
Non-trainable params: 22,532,992

Figure 6.7: ResNet50 architecture

After training our network, we will proceed to our next phase: ***fine tuning*.** This step described previously can also improve the performance of our network further by allowing the activation of the learning of all or some levels of the basic model.

We obtain the following results for the *ResNet50* and *DenseNet121* applied to tour dataset:

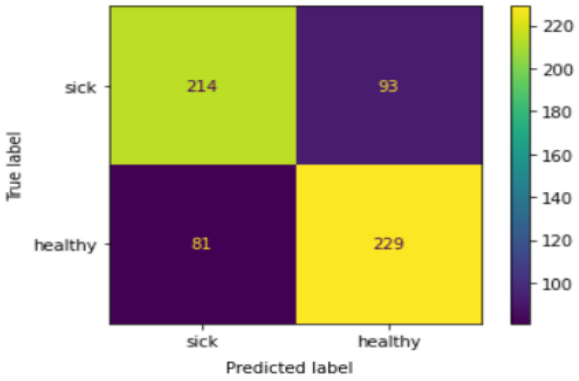


Figure 6.8: ResNet50 confusion matrix

	precision	recall	f1-score	support
sick	0.73	0.70	0.71	307
healthy	0.71	0.74	0.72	310
accuracy			0.72	617
macro avg	0.72	0.72	0.72	617
weighted avg	0.72	0.72	0.72	617

Figure 6.9: ResNet50 results

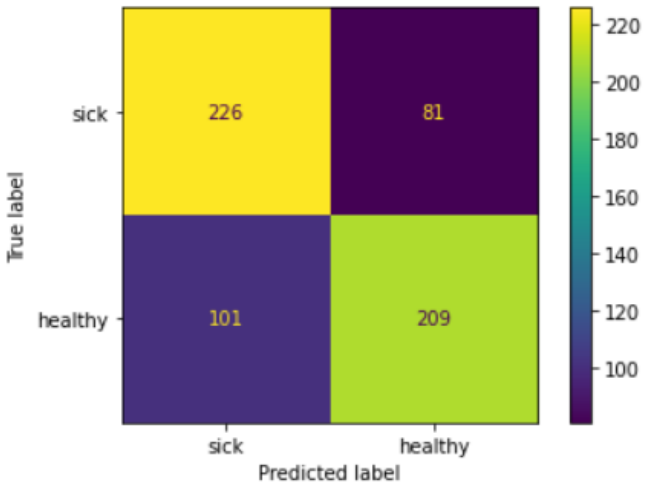


Figure 6.10: DenseNet121 confusion matrix

	precision	recall	f1-score	support
sick	0.69	0.74	0.71	307
healthy	0.72	0.67	0.70	310
accuracy			0.71	617
macro avg	0.71	0.71	0.70	617
weighted avg	0.71	0.71	0.70	617

Figure 6.11: DenseNet121 results

From the results presented in figure 6.8 and figure 6.9 we can clearly see that the *ResNet50* model achieved the best performance overall in terms of accuracy and f1 scores compared to the *DenseNet121* model.

Finally the performance between all the models implemented in chapters 5 and 6 is shown in the following histogram:

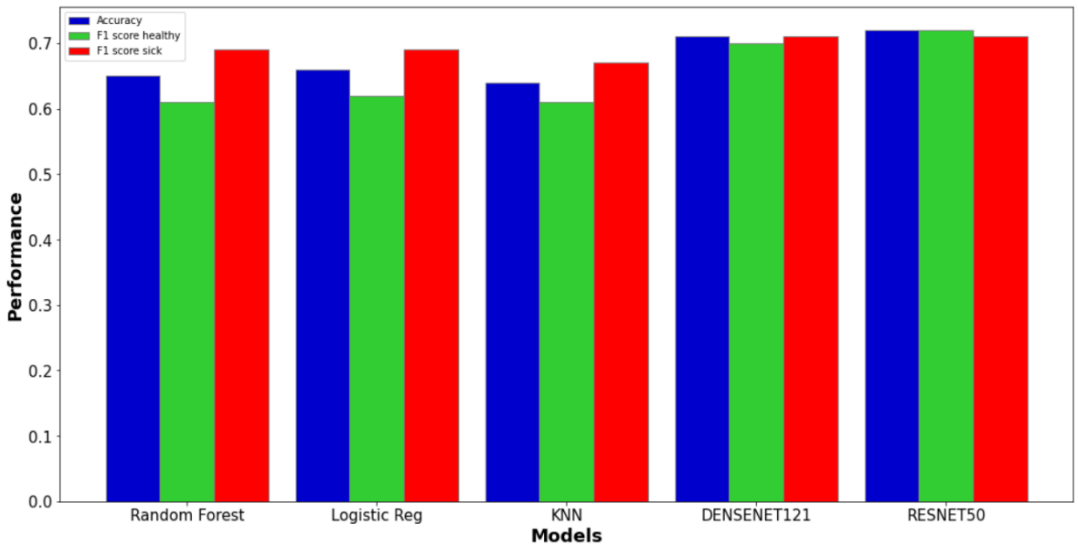


Figure 6.12: Performance comparison of all implemented models

Conclusion and Final Remarks

Based on the results obtained in figure 6.10, from the classical machine learning classification models we notice that Logistic Regression model performed best with a 66% overall accuracy followed by Random Forest with 65% and lastly k-nearest neighbors achieving 64% overall accuracy. Compared to CNNs, we observe a greater performance overall with ResNet50 performing the best with an accuracy of 72%.

Comparing the classical machine learning models to CNN it is necessary to clarify that while the classical models can work well on small datasets, neural networks require generally a large amount of data in order to perform well, the more data they have to learn from, the more they will be able to generalize. For smaller datasets several techniques can be employed on the architectural level of CNNs such as applying *data augmentation* on the dataset and adding *dropout* layers in the network to prevent overfitting. Conversely, classical machine learning algorithms that were applied can be more efficient in computational resources and more accurate in classifying smaller datasets with a specific set of features based on the well-defined and selected vegetative indices. Applying smaller datasets will also make these models tend to be less prone to overfitting. Additionally classical machine learning algorithms can be more extensible, meaning that features can be easily added and integrated. For instance, given a wider range of multispectral bands, it may be possible to add new metrics that make use of a wider set vegetative indices that can quantify and represent more information of the vegetation.

However, a noticeable contrast between classical machine learning algorithms and CNNs, is the required amount of data employed for each of the methods applied.

While CNNs require only the RGB photo of the plant as input, traditional machine learning algorithms rely on a set of features that were pre-computed in advance. In our case the features used were a set of vegetational indices computed from five multiple spectral bands. For comparison, for each plant tested, the single RGB image is around 1 Megabytes while the combined size of the multispectral images is around 20 Megabytes. Given this observation, we can deduce that for every plant, classical machine learning algorithms make use of around 20 times the amount of data required compared to CNNs.

Considering that CNNs algorithms performed best in terms of accuracy and f1 scores while requiring less data to be processed and pre-computed, we can conclude that Convolutional Neural Networks, despite requiring more computational resources, are the optimal performing algorithm for our project.

Nevertheless, even if we only employed CNNs as our solution, we cannot solely rely on RGB images to achieve the desired results, given that multispectral images were necessary for achieving our goal in this project, that being the automated plant extraction and subdivision. Being one of the crucial steps of our project, the automated plant extraction and sectioning phase relies solely on multispectral bands in order to detect vegetation of the image before isolating the plant under consideration. Although in this step, it is important to note that only the Red and NIR bands were used in order to compute the NDVI index. However there exist in the literature a number of solutions that only utilize RGB photos as their input in order to highlight vegetation in a specific image. One of the proposed solutions is the use of suitable augmentations on images so the defined model learns semantic representations that are invariant to photometric and geometric shifts and employ an adaptive sampling technique, which determines the training images based on a pixel-wise distribution of classes measurement and real network confidence

(Tavera, Antonio, et al., 2022), thus providing an approach that moves away from classical handcrafted vegetative indices towards more robust computer vision methods.

A final remark should be made on the application of the labels on the provided dataset. Being conducted by the human eye, the labeling process can be prone to the element of subjectivity. For instance, for every shooting set, we can notice that the images were evaluated and labeled relative to their surroundings (other subsections of the same tree or other trees belonging to the same subset of images). For example, given that in May and June trees appear to be generally healthier than in July, an image labeled “sick” in May/June may be considered “healthy” if it belonged to a shooting set in July. Moreover, the labeling process was solely based on the RGB photos, thus lacking the additional information provided on the NIR and Red-Edge bands and also making the evaluation process more susceptible to lighting conditions. We must also add that it is essential for the labeling process to be cross-checked by a team of experts in order to avoid discrepancy in the data and reduce the human error attributed to subjectivity.

In conclusion, the best implemented solution was able to achieve an accuracy of 72% allowing the identification and classification of healthy and sick subregions of each plant tested. A higher accuracy proves impossible to achieve due to the fuzziness of the labels applied.

Therefore, any future developments should certainly include:

- A finer and more accurate labeling process from a team of expert botanists that should be thoroughly cross-checked to avoid any bias or subjectivity.
- More precise shots taking into consideration perfect lighting conditions and overall good exposure.

- The capture of higher resolution images for more detail, and better framing of trees under consideration.
- Use of better sensors that can capture more bands therefore providing the possibility to exploit more vegetative indices (for instance hyperspectral sensors).
- A larger and more diverse dataset in order to enhance the performance of CNNs.
- More computational power that will allow the application of a greater number of hyperparameters in order to achieve better optimization of the machine learning algorithms.

Bibliography

[1] **The Nature of Geographic Information - Spectral Response Patterns:**

https://www.e-education.psu.edu/natureofgeoinfo/c8_p5.html

[2] **DJI P4 Multispectral:**

https://www.dji.com/it/p4-multispectral/specs

[3] **L3Harris – Broadband Greenness:**

https://www.l3harrisgeospatial.com/docs/broadbandgreenness.html

[4] **Agricolus site:**

https://www.agricolus.com/en/vegetation-indices-ndvi-ndmi/

[5] **Auravant site:**

https://www.auravant.com/en/blog/precision-agriculture

[6] **EOS Data Analytics – Precision Agriculture:**

https://eos.com/industries/agriculture/

[7] **Koch Agronomic - Knowledge Center:**

https://kochagronomicservices.com/knowledge-center/

[8] **IBM- Machine Learning:**

https://www.ibm.com/cloud/learn/machine-learning/

[9] **IBM – Random Forest:**

https://www.ibm.com/cloud/learn/random-forest

[10] **Scikit-learn – KMeans:**

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[11] **TechTarget – Logistic Regression:**

https://www.techtarget.com/searchbusinessanalytics/definition/logistic-regression

[12] **IBM – K-Nearest Neighbors Algorithm:**

https://www.ibm.com/topics/knn

[13] **EXSILIO - Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures:**

https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

[14] **V7labs - Activation Functions in Neural Networks:**

https://www.v7labs.com/blog/neural-networks-activation-functions

[15] **AI Multiple – Data Augmentation:**

https://research.aimultiple.com/data-augmentation/

[16] **Keras – Keras Applications:**

https://keras.io/api/applications/

[17] **Mobasheri, M. R., M. Chahardouli, and M. Farajzadeh. "*Introducing PASAVI and PANDVI methods for sugarcane physiological date estimation, using ASTER images*." (2010): 309-320.**

[18] **Bausch, Walter C., and Jorge A. Delgado. "*Impact of residual soil nitrate on in-season nitrogen applications to irrigated corn based on remotely sensed assessments of crop nitrogen status.*" Precision Agriculture 6.6 (2005): 509-519.**

[19] **Liao, Kuo, et al. "*Detection of Eucalyptus Leaf Disease with UAV Multispectral Imagery.*" Forests 13.8 (2022): 1322.**

[20] **Tavera, Antonio, et al. "*Augmentation Invariance and Adaptive Sampling in Semantic Segmentation of Agricultural Aerial Images.*" Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022.**