
**Honours Project - MHW225671**

# FINAL REPORT

## 2023-2024

## Department of Computing

**Submitted for the Degree of: BSc Computing**

**Project Title:** Recognition of leaf diseases of apple trees and vines using Machine Learning

**Name:** Giacomo Fadda

**Programme:** Bsc/Bsc (Hons) Computing

**Matriculation Number:** S2219926

**Project Supervisor:** Mario Mata

**Second Marker:**Maria Belk

**Words Count:** 10542

**(excluding contents pages, figures, tables, references and Appendices)**

**"Except where explicitly stated, all work in this report, including the appendices, is my own original work and has not been submitted elsewhere in fulfilment of the requirement of this or any other award"**

**Signed by Student:** **Date:**April 19, 2024

**Abstract**

This project aims to use machine learning techniques to identify some of the main pathogens of apple and grape trees, two of the most relevant Italian agricultural products. These diseases, often visible on the leaves, can be challenging to diagnose, particularly for agricultural students and early-career farmers.

With the purpose of helping them to improve their ability to diagnose these pathogens accurately and efficiently, a mobile application was developed, employing a machine learning model to identify and classify leaf diseases in apple trees and grapes.

The primary objective was to deploy a Convolutional Neural Network (CNN) specifically tailored and optimised for mobile devices, enabling users in the field to efficiently diagnose diseases directly from their devices.

The CNN model was trained using a dataset comprising thousands of labelled images of the principal apple and grape diseases. This dataset was created using images from the PlantVillage dataset and other sources to ensure robust model training. TensorFlow Lite facilitated the integration of the CNN model into the Android application, allowing it to process images captured by the device's camera, analyse them for symptoms, and classify them accurately against known disease types.

The evaluation of the application focused on its accuracy and usability on standard mobile devices. Despite some limitations, test results demonstrated that the application achieves good accuracy levels, validating its effectiveness as a tool to aid agricultural science students and early-career farmers in recognising diseases in apple and grape plants.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

This section presents an overview of the impact of some diseases that can affect two of the leading agricultural products grown in Italy, grapes and apples, their economic impact, and the importance of precise and timely recognition of the pathogen by intervening suddenly to reduce the damage caused. It will then briefly explain the project's overview and objectives through a literature review to give the basement to develop the project and the method to follow to complete the project.

## 1.1 Background of the project

Italy is a Southern European country with a solid agricultural vocation, where among the main agricultural products grown are grapes for producing wine and apples.

Looking at the charts of agricultural production of the European Union of the Eurostat update to November 2024, Italy is one of the leading apple and grapevine producers, producing 18% of apples and 31.3% of grape wine in the European Union.(Eurostat 2023)

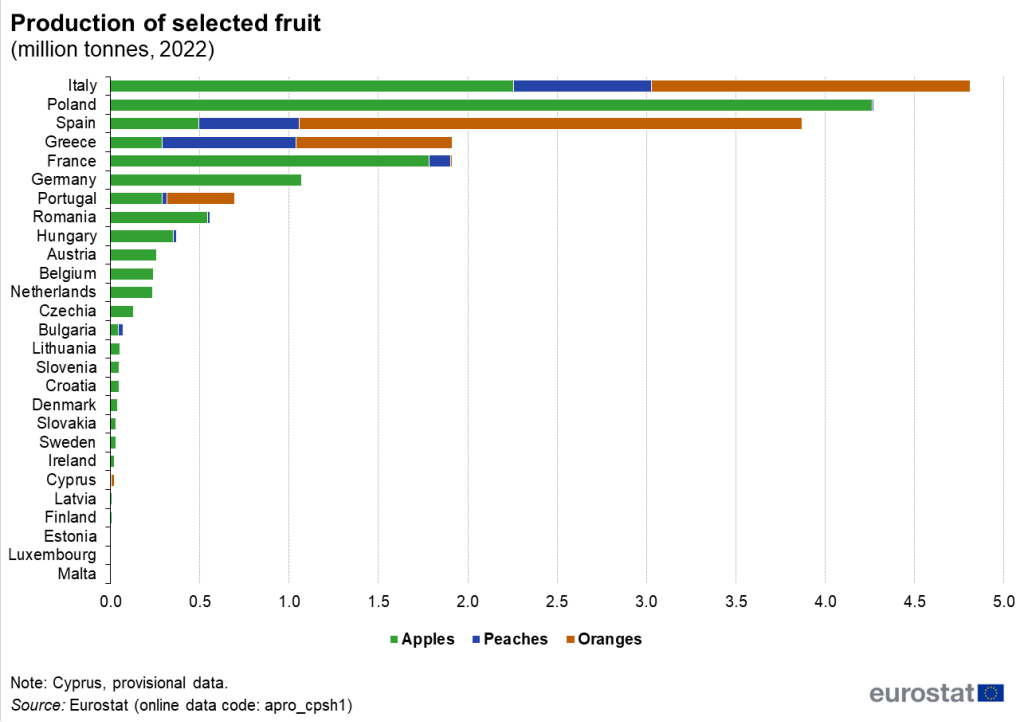



Figure 1: Production of selected fruit in the European Union(Eurostat 2023)

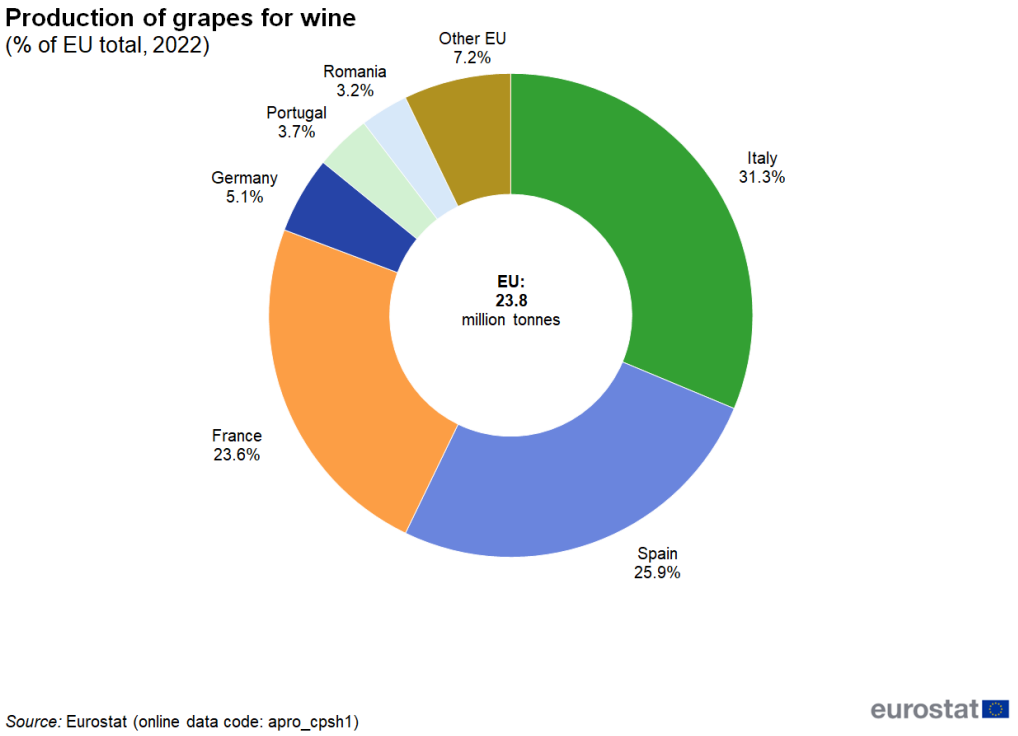



Figure 2: Production of grapewine in the European Union(Eurostat 2023)

Whereas, FAO (2023) (Food and Agriculture Organisation of the United Nations) reports the top ten producing countries worldwide of these agricultural products, where Italy turns out to be in first place for grapes production with 8,171,769.95 tons per year and sixth place for the apple production with 2,221,443.71 tons per year.

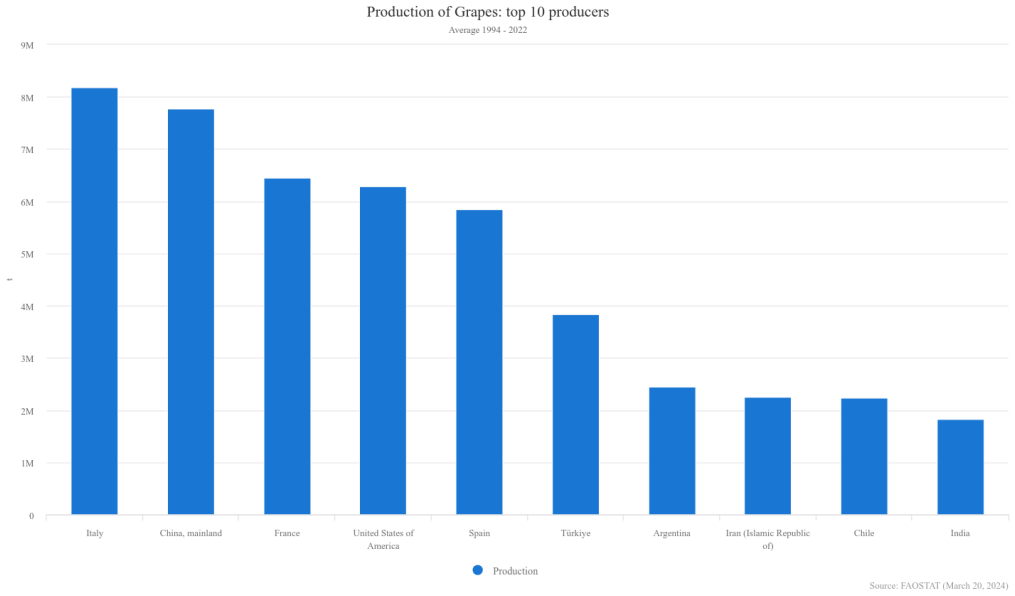



Figure 3: World top ten countries grape producers, average tons 1994 - 2022(FAO 2023)

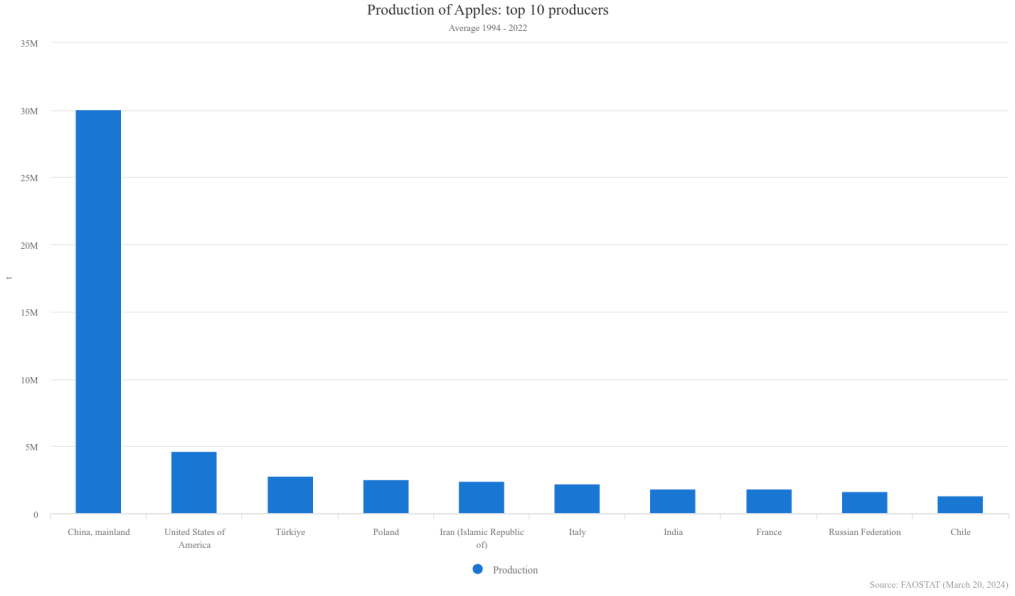



Figure 4: world top ten countries apples producers, average tons 1994 - 2022 (FAO 2023)

### 1.1.1 Impact of diseases on agricultural production

It is estimated that an increase of 60% in food production by 2050 will be needed to cover the needs of the human world population. However, Global yield losses due to crop pathogens are still significant (Ristaino et al. 2021), with an estimated cost of US$220 billion annually for the global economy("Pathogens, precipitation and produce prices" 2021).

### 1.1.2 Grapes main diseases

Viticulture in Italy is a strategic activity in the Italian agro-food sector because the country is at the podium among world wine producers. In 2016, 5% of the UAA (Utilised Agricultural Area) was for the production of grapes for wine, about 622,000Ha(Corsi, Mazzarino, and Pomarici 2019)

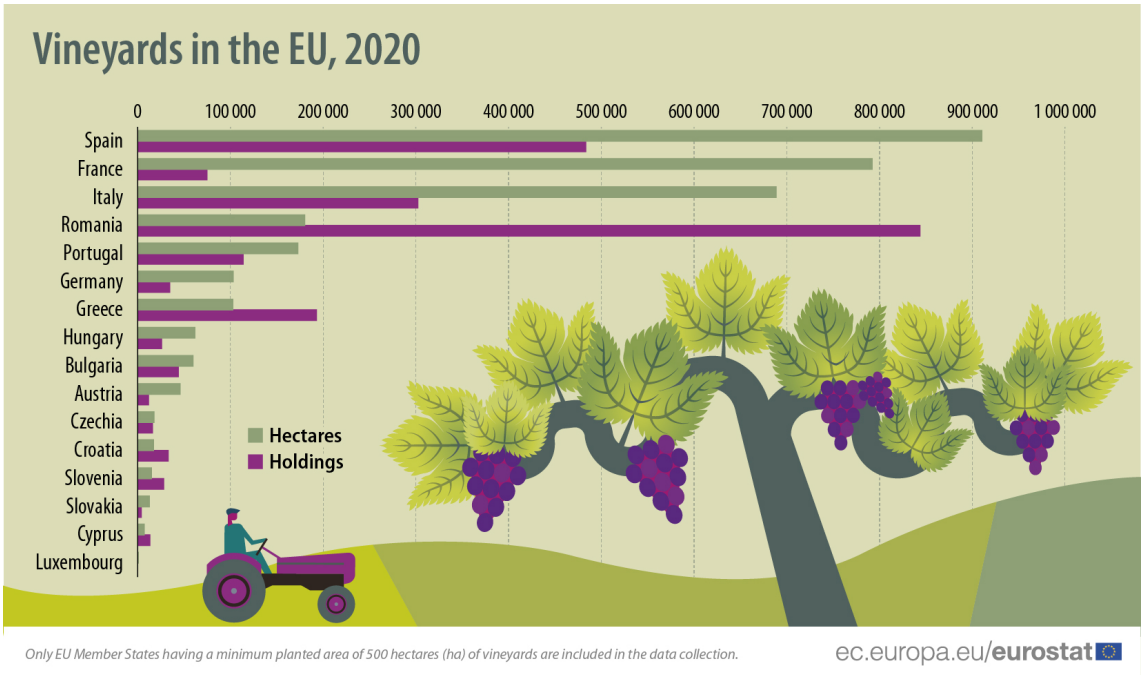



Figure 5: Vineyards in the European Union (EU) in 2020 (Eurostat 2022)

From this data, it is possible to deduce the importance of the fight against the main pathogens that can lead to a loss or reduction in product quality.
Black rot and Esca are two of the primary grapevine pathogens that can affect grapes with high economic damage.

**Black rot** caused by the fungus *Guignardia bidwellii* was the cause of a severe epidemic in Italy In 2011(Rossi et al. 2015)

Originally from North America, the European vine is particularly susceptible, and In the absence of sufficient crop protection, vineyards with severe infections can lose up to 100% of their yield.(Szabó et al. 2023)

The first symptoms of the presence of this pathogen in the leaves can be observed as light brown to reddish brown circular leaf spots and lesions.(Vogelweith and Thiéry 2017)

Figure 6: grape leave infected by black rot (Abdallah 2019)

**Esca** Caused by the fungus *Phaeoacremonium aleophilum, Phaeomoniella chlamydospora, and Fomitiporia mediterranea* is an important grape disease that can lead to the death of the grape plant itself.
Esca poses a growing threat to the world's wine industry, resulting in significant losses through yield reduction, wilting or declining vines, and shortened vineyard lifespans and, according to recent research, the illness may also have an impact on the chemical makeup of musts and wines, as well as the quality of the grapes.(Jordão, Botelho, and Miljić 2023)

The first symptoms of this infection are visible through the leaves, which take on characteristics of interveinal chlorotic discolouration, leading to leaf necrosis.(Pasquier et al. 2013)

Figure 7: grape leave infected by Esca (Abdallah 2019)

### 1.1.3 Apples main diseases

Apple is one of the most important fruit crops worldwide for its various uses, and with 55,800 hectares covered by orchards producing this fruit, Italy is among the top European producers.(Eurostat 2019)

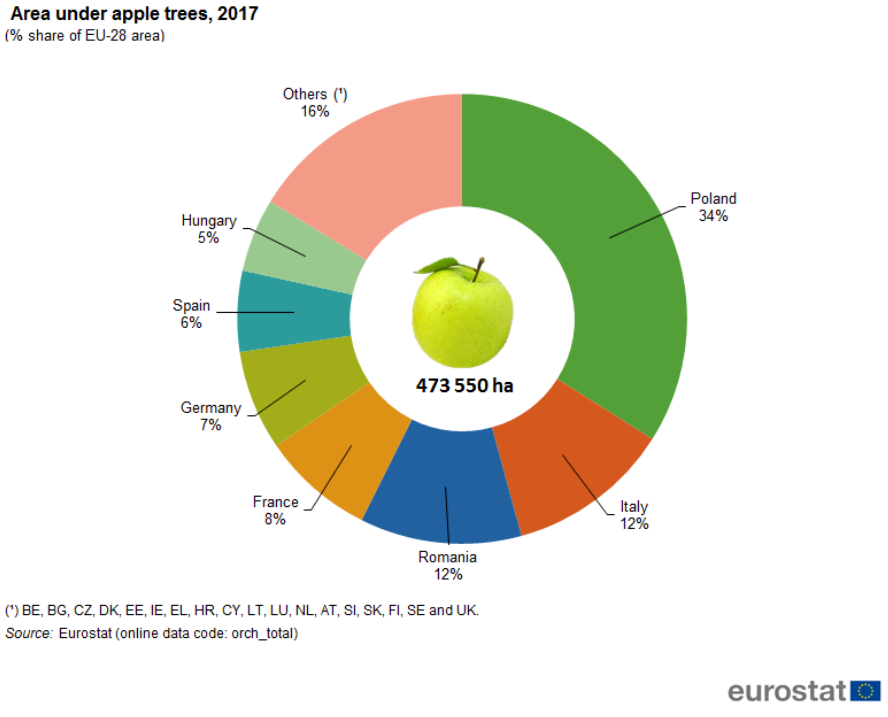



Figure 8: Area under apple trees, 2017 in the EU (Eurostat 2019)

Among the main fungal diseases that can affect apple production are Apple Scab and powdery mildew.

**Apple Scab** is among the common fungal diseases affecting this cultivar in the temperate and humid regions of the world. Caused by the fungus *Venturia Inaequalis*, the apple scab is one of the most negatively impacting apple tree diseases from an economic point of view because it causes a reduction in the quantity and quality of the fruit.(Belete and Boyraz 2017). In the early stages of the infection, the first symptoms of the apple scab infection are mainly observed on leaves as small lesions on the underside.

Figure 9: Apple leave infected by scab (Abdallah 2019)

To underscore that, in the first stages of infection, the symptoms of this disease on the leaves can be mistaken for sooty mould (another fungal disease) or leaf "fuzz"(Ciancio and Mukerji 2008). Belete and Boyraz (2017) say that if the spread of this disease is not well controlled, it can cause economic losses of up to 70% of the product value.

**The apple powdery mildew**, caused by the fungus *Podosphaera leucotricha* is another economically impactful disease that may cause the fruit to be russeted, discoloured and dwarfed. Like the other pathogens, the first symptoms appear in the leaves, where small, whitish, felt-like patches of fungal growth appear and quickly cover the entire leaf(Ellis 2008).

Figure 10: Apple leave infected by powdery mildew (Turechek 2023)

### 1.1.4 Impact of climate change on the increase in the incidence of plant diseases

The adverse effects of climate change should also be highlighted among the factors contributing to increasing the severity of most of plant diseases.

Bregaglio, Donatelli, and Confalonieri (2013) argue that even though there is still scarce literature on the changes in the future distribution of plant diseases, there is evidence that climate changes can profoundly affect the effect of plant diseases.

However, the argument about the impact of climate change and the interaction with plant pathogens is more complex than expected because climate change may significantly alter plant disease development patterns, with combinations of driving factors influencing crops and pathogens at varying rates.(Caffarra et al. 2012)

Remaining in the context of vine and apple tree diseases due to the CO2 enrichment, due to the CO2 enrichment, N (Nitrogen) deposition, and changes in temperature and rainfall regimes can lead to an increase in fungal diseases(Bregaglio, Donatelli, and Confalonieri 2013)

For instance, the results of simulations of future scenarios of downy mildew epidemics on grapes under climate changes indicate a significant increase in infections(Francesca et al. 2006)

### 1.1.5 Importance of the pathogens' prompt detection

Having established the importance of the production of grapes (especially for wine-making) and apples for the Italian agricultural economy, how the diseases that afflict

these cultivars can lead to the loss of production or its loss of quality and how the climate changes can affect the incidence of plant pathogens, it is therefore essential to try to reduce or eradicate this problem through the detection, monitoring and management of potential pathogens in the early stages(Khaled et al. 2018), that can often manifest themselves on the leaves and can be done with different methodologies.

| **Detection Methods** | | |
|---|---|---|
| **Visual Identification** | **Laboratory Tests** | **Advanced Techniques** |
| • individual monitoring<br>• Scouts Monitoring | • Physiological Test<br>• Biological Test<br>• Seriological Test<br>• Molecular Test | • Spectroscopy Techniques<br>• Image Processing<br>• Electronic Nose<br>• Sonic Tomography<br>• Microfocus X-ray<br>• Flourecence (uXRF)<br>• Terrestrial Laser Scanning |

Table 1: Different method of plant diseases detection (Khaled et al. 2018)

From the methodologies listed in the table above, there is the visual identification of plant pathogens that, although it could prove expensive and time-consuming for large farms due to the need for continuous monitoring by experts in large areas(Al-Hiary et al. 2011), for the average size of the Italian agricultural holdings that have a regional average of between 2.2 and 19 hectares(Eurostat 2018) it is a methodology that can be easily adopted.

However, the difficulty of proceeding in this way for an untrained eye of a student and an early career farming entrepreneur can be high, and to fill this gap, it is necessary to acquire experiences in the field and adopt the suitable instruments and resources that often are not easy to find.

## 1.2 Project Overview

The project overview has the purpose of the project outline.

### 1.2.1 Project Outline

Nowadays, learning also occurs through the use of technologies and the high presence of portable devices, and the current level of maturity of Machine Learning technologies can be a starting point for using these elements to help solve the problem. Thus, the research question of the project is:

**Can a mobile application that implements Machine learning be precise enough to help agricultural science students and early-career farming entrepreneurs to train to recognise the apples and grapes diseases?**

### 1.2.2 Project Aims and Objectives

The project aims to develop a mobile application that can run on Android devices and will use machine learning to detect the presence of a pathogen in grapevines and apple trees by recognising the first symptoms of the disease in the leaves and comparing the result with other examples to help the user to recognise the diseases. The first symptoms are usually visible in the leaves of these plants, and the device's camera will be used to capture images to be analysed by the software.

The software will be developed as a tool for agricultural science students and early career farming entrepreneurs to study and reduce the margin of error in recognising those diseases. The main objective of this project is to reduce the possibility of confusing the symptoms of the different diseases present in the foliar system of the plants examined (apple tree, vine), which often represents a challenge for beginners in the topic.

### 1.2.3 Secondary Objectives

- **Investigate the use of machine learning and Deep Learning**
  Performing research into Machine Learning Concepts and its sub-field deep learning to understand which part of this technology to focus on for the project's development.
- **Investigate the use of machine learning to recognise plants' diseases**
  Performing research into machine learning to recognise the plants' diseases to analyse the work done by other researchers and understand the project's feasibility and usefulness.
- **Investigate into Machine learning in mobile apps**
  Performing research into the technology used to develop Android Apps which implement machine learning and possible issues.

### 1.2.4 Primary Objectives

- **Requirement**
  look for everything necessary to develop the project and prepare the environment.
- **Planning**
  Create a project plan to follow during all project phases and write documentation to test it.
- **Design**
  Design the software interface and relate use case and activity diagrams.
- **Development stage 1: Develop a Convolutional Neural Network for image recognition tasks**
  Develop the Convolutional Neural Network core using the knowledge gained through the literature review.
- **Development stage 2: the Android application**
  Develop the Android Application to implement the previously created CNN core and use the device's camera to detect images to predict.
- **Testing**
  Install the application on a mobile and test its performance using a set of images as a tester to check if the app meets the initial requirements and expectations.

- **Review**
  It involves analysing all data collected through the application test and stored in the test documentation to determine if the software achieves the set objectives and satisfies them.

### 1.2.5 Hypothesis

To make sure that the application can be a valid instrument to help recognise the apple and grape cultivars diseases, it will be essential that the software can achieve the following results during the test phase:

- The Machine learning model that will be used will achieve a high accuracy score and a low level of false positives.
- The accuracy achieved during model training can be preserved when the model is integrated into the mobile app without degrading the device's performance.

## 1.3 Structure of the rest of the project report

The rest of the project chapters are structured as follows:

1. **Literature and Technology Review**: This chapter discusses all relevant findings from existing research related to the project topic, focusing on advancements in machine learning and their applications in agriculture.
2. **Problem Analysis**: This chapter analyses the problem, describes the research methodology, and outlines the project lifecycle to develop the solution.
3. **projects Requirements**: This chapter details all the requirements necessary to develop the project, including technical specifications and user needs.
4. **projects planning**: This chapter provides a comprehensive plan for the project, including timelines, resources needed, and stages of development.
5. **Design the wireframe and diagrams**: This chapter describes the project's wireframe and related diagrams, detailing the design process and the rationale behind the interface and architecture choices.
6. **Developing stage 1: Developing Convolutional Neural Network (CNN)**: This chapter covers the development of the Convolutional Neural Network models.
7. **Developing stage 2: Developing the Android application**: This chapter details the development of the Android application, from initial setup to integration of the CNN models.

8. **Application testing** : This chapter provides information on the application testing process, including test cases, testing methods, and initial findings.
9. **Evaluation**: This chapter describes all tests made to evaluate the application.
10. **Legal, Social, Ethical and Professional Issues**: This chapter addresses the project's legal, social, ethical, and professional considerations.
11. **Conclusions** : This chapter summarises the project's outcomes, reflecting on the objectives met, lessons learned, and potential areas for future research.

# 2 Literature and Technology Review

The literature review is an essential component of the project as it provides the basic knowledge necessary for understanding the project through research of similar projects and other relevant literature to address the aspects of the research question/problem statement.

The acquisition and analysis of literature and similar projects made by others will help accredit the project's usefulness, understand its feasibility and give a guideline for its development. In order to cover the objectives found in the project outline, the literature review will contain the following points:

- Investigate the main concepts and the use of machine learning and Deep Learning.
- Investigate the use of machine learning to recognise plants' diseases.
- Investigate Machine learning in Android mobile apps.

## 2.1 Investigate the main concepts and the use of machine learning and Deep Learning.

The project's central core consist of the use of machine learning to recognise the plant's diseases through images. As a result, it would be helpful to investigate the main Machine learning and Deep learning concepts and the challenges in classifying the images using this technology.

### 2.1.1 Definition of Machine Learning

The challenges encountered by systems that are dependent on pre-programmed knowledge underscore the necessity for Artificial Intelligence (AI) systems to possess the capability to amass their knowledge independently. This capability is achieved through the identification of patterns within unprocessed data, a process commonly referred to as machine learning.(Goodfellow, Bengio, and Courville 2016)

Machine learning technology is a subfield of AI (Artificial intelligence) where a machine shows learning concerning a specific task, performance metric and type of experience if there is a constant improvement in its performance at that task after experience is acquired.(Chowdhary 2020a)

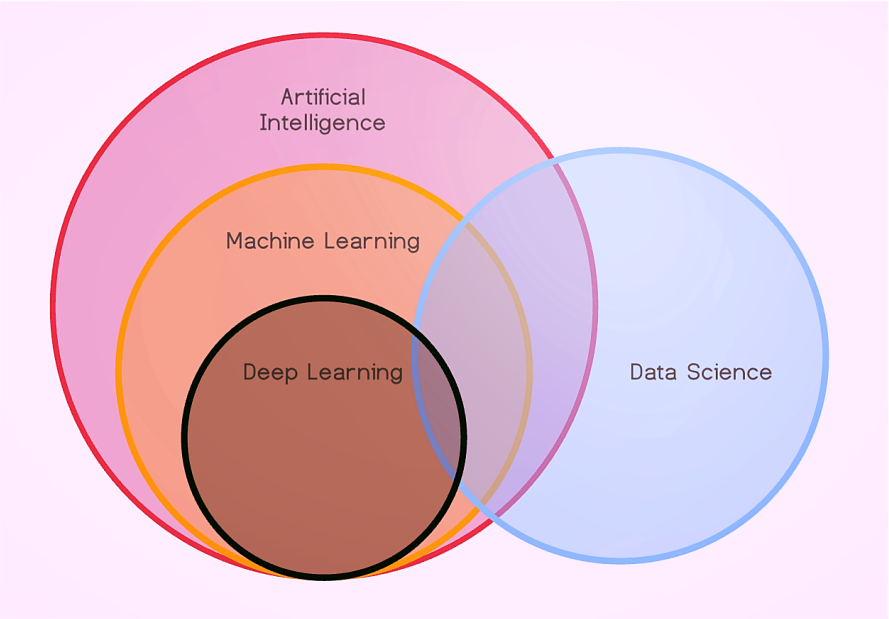



Figure 11: Ven Diagram with the the AI subfields.(Team 2019)

### 2.1.2 The main types of Machine Learning

The main forms of learning used by Machine learning are:

1. **Supervised learning:** It is a learning approach where the parameters are fine-tuned to minimise discrepancies and errors between the desired output and the output produced.(Jo 2021)

2. **Unsupervised Learning:** Unsupervised learning is identified as the procedure of enhancing the prototypes of clusters, which is contingent on the degree of similarities among the instances in the training set.(ibid.)

3. **Semi-supervised learning:** The concept of semi-supervised learning is designed to incorporate both labelled examples, which are typically more expensive to gather, and unlabelled examples, which are relatively inexpensive to acquire, for training learning algorithms.(ibid.)

4. **Reinforcement learning:** It is characterised as a type of learning where the parameters are adjusted to optimise the reward and minimise the penalty.(ibid.)

### 2.1.3 Deep Learning and Convolutional Neural Network for image recognition tasks

Deep learning is a subfield of Machine learning that, with its neural net-based approach inspired by brain science, involves the simulation of Artificial Neural Networks (ANNs) that learn gradually over time. It results in being s particularly well suited for tasks such as image processing, speech recognition, and decision-making.(Chowdhary 2020b)

The benefits of Deep Learning algorithms compared to traditional Machine learning stay in creating more accurate models from large datasets and the possibility of speeding up the pipeline significantly by eliminating the need for expert domain knowledge, although training times may be longer. (Fergus and Chalmers 2022)These benefits will significantly increase the performance of the algorithm.

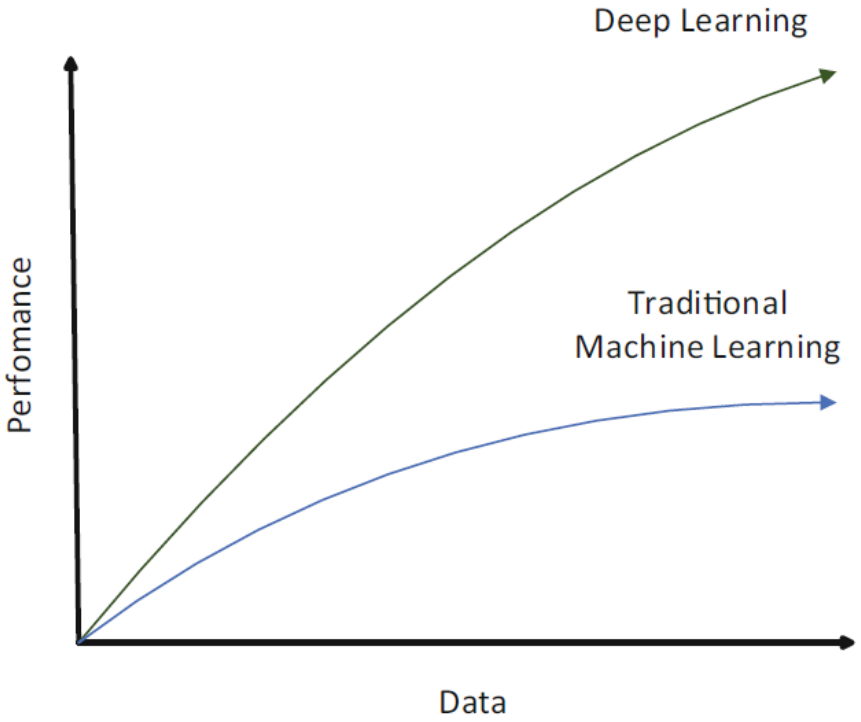



Figure 12: Deep learning performance(Fergus and Chalmers 2022)

Once the definition of deep learning is explained, it is essential to speak about the Convolutional Neural Network (CNN) as will be implemented in the project development.

CNNs are specifically designed to recognise visual patterns from images with minimal processing. They are a multi-layer neural network that can process 2D array input data, such as images, and learn features from the input images to make predictions.(Sewak, Karim, and Pujari 2018)

A typical Convolutional Neural Network (CNN) architecture comprises the following components:

1. **Convolutional Layers:** filter the input data to extract relevant features. These layers are the building blocks of CNNs, and multiple layers are stacked to learn hierarchical representations.
2. **Activation Function:** allows the CNN to learn complex patterns effectively by introducing non-linearity into the model.
3. **Pooling Layers:**Reduce the spatial dimensions of the feature maps and downsample the learned representations.
4. **Fully Connected Layers:**Perform classification or regression based on the learned representations.
5. **Output Layer:**This provides the final prediction based on the learned features.

(Wei 2023)

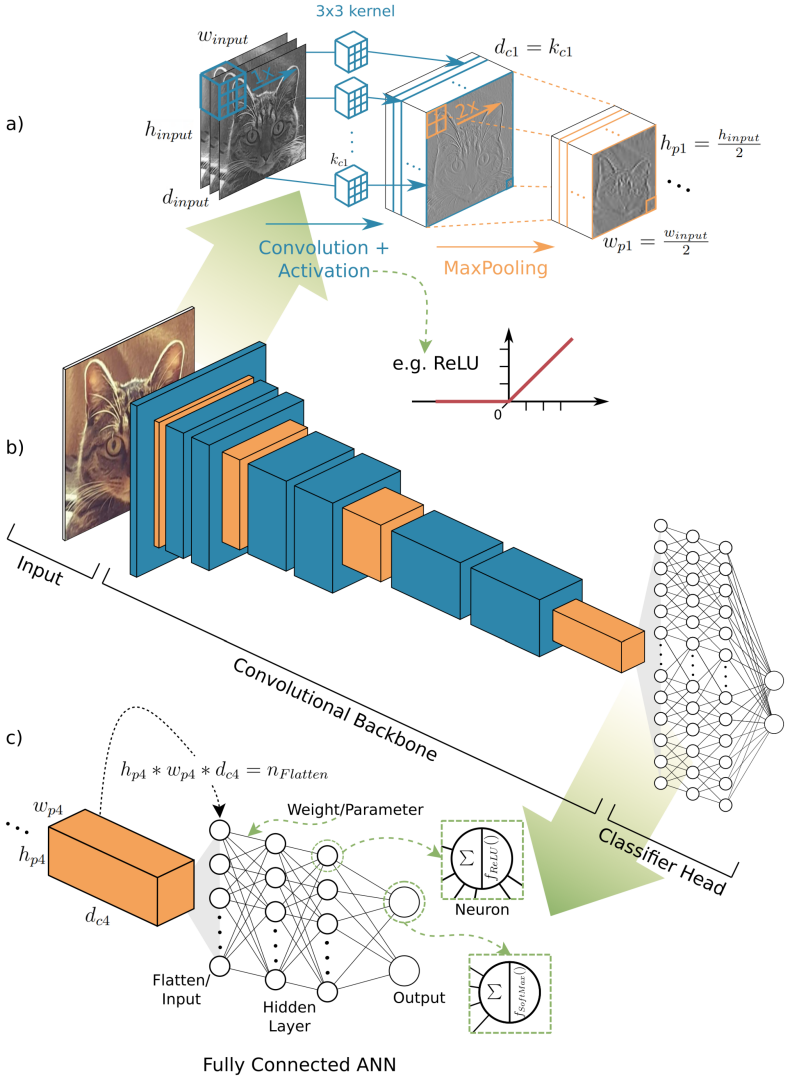



Figure 13: CNN feed-forward network(Hoeser and Kuenzer 2020)

The figure below shows that the CNN architecture can be split into Input, Convolutional Backbone Classifier. The 2D array input undergoes a series of convolutions, activations, and max pooling operations, known as the convolutional backbone, to extract high-level features. The classifier, located at the end of the backbone, is a sequence of stacked, fully connected layers, similar to an Artificial Neural Network (ANN). It uses the extracted features from the convolutional backbone to classify them into output classes and provide their probability.(Hoeser and Kuenzer 2020)

### 2.1.4 Potential challenges in the development of Convolutional Neural Networks.

As mentioned, the Convolutional Neural Network (CNN) is a widespread technique in computer vision tasks. However, training a CNN model requires a large amount of data, demonstrating remarkable success where sufficient labelled data is available for training, such as in traffic sign identification, medical picture segmentation, face detection, and object identification in natural photos.(Bhatt et al. 2021)

Moreover,Bhatt et al. (ibid.)argue other reasons why the CNN models can be challenging:

- Deep CNNs can be challenging to comprehend and explain because they are typically black boxes.
- Adding a small amount of random noise to the input image can cause the network to incorrectly classify the original and slightly agitated variant, increasing the misclassification error when training a CNN.
- Deep CNNs use need of robust hardware resources.
- The CNN performance is significantly influenced by the choice of hyperparameters. Even a minor change in the hyper-parameter values can affect the overall performance of the CNN. Therefore, it is crucial to select hyper-parameters carefully, which is a critical design issue that requires an appropriate optimisation technique.

## 2.2 Investigate the use of machine learning to recognise plant diseases.

Modern computer vision techniques based on machine learning can help to reduce the time of recognising plants' diseases at the early stages with the main advantage of increasing productivity by going through an appropriate management statement of the issue, such as a more correct and focalised use of the appropriate agrochemical. (Abbaspour-Gilandeh et al. 2022).

Thapa et al. (2020) argues that advancements in digital imaging and machine learning have shown significant promise in enhancing the speed and accuracy of plant disease recognition, made possible through the use of smartphones, which are now equipped with high-quality cameras capable of capturing detailed images of disease symptoms.

To develop the software to classify foliar diseases of apples were collected 3651 images of the primary apple foliar diseases: apple scab, cedar apple rust, and leaves with more than one disease in the same leaf training a CNN ResNet50 pre-trained on ImageNet accuracy, achieving the result of over 97% of accuracy on most diseases category except multiple diseases category where was achieve a 51% of accuracy.(ibid.).

An analogous study has been conducted with the aim of identifying the main grape diseases, namely Black rot, Esca, and Isariopsis leaf spot. This research utilised the integrated architectures of GGNet, GoogLeNet, and DenseNet, which were trained in line with the control experiments of UnitedMode. (Ji, Zhang, and Wu 2020) The achieved results achieving are the following:

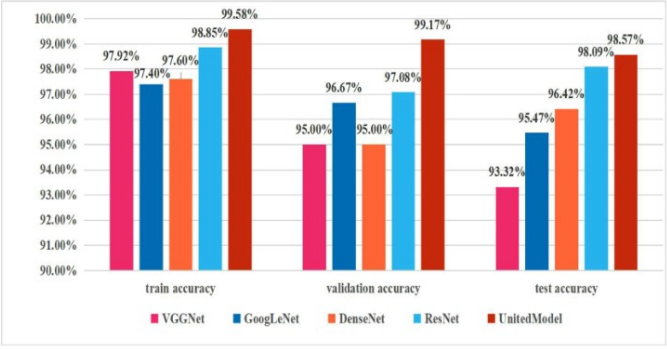



Figure 14: Comparison of the average accuracy for the different architectures.(Ji, Zhang, and Wu 2020)

With a similar method, Inception, MobileNet, ResNet, VGG, and Xception DenseNet (previously used in the above test) architectures were also tested and compared, returning the following F1-score, which is an overall estimation of the precision and recall of the test subject (Kabir, Ohi, and Mridha 2021):

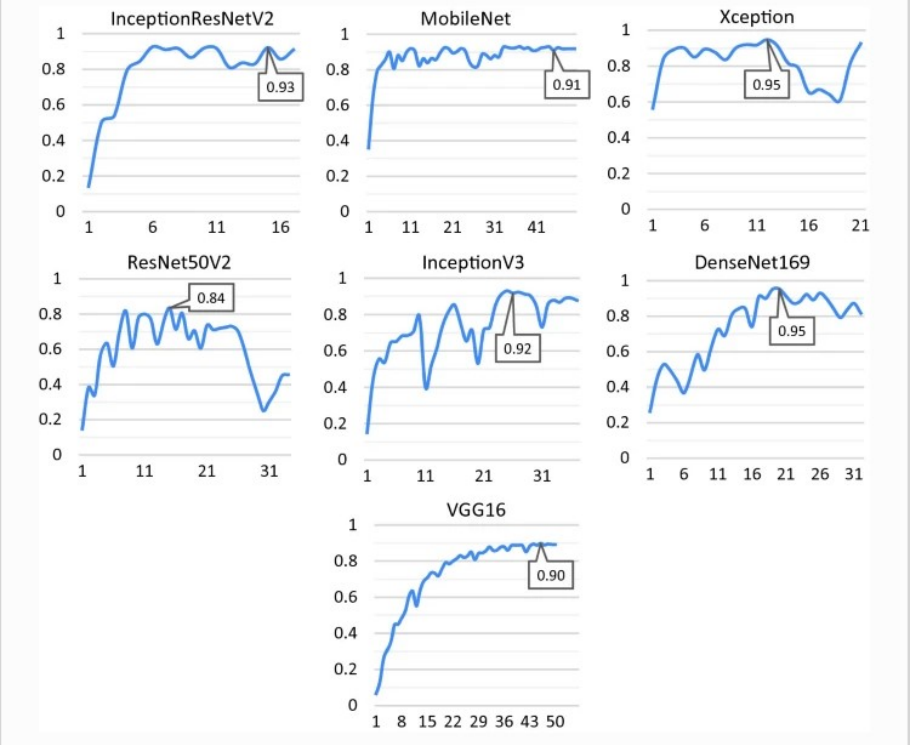



Figure 15: F1-score calculated on the validation dataset.(Kabir, Ohi, and Mridha 2021)

Militante, Gerardo, and Dionisio (2019) outline the process and findings of a software developed for identifying plant leaves and diagnosing diseases. They used the Plant Village repository dataset to train their Convolutional Neural Network (CNN) model. The images used were colour images, resized to a 96x96 resolution for processing. The model employed fully connected layers for classification and convolutional and pooling layers for feature extraction.

The model demonstrated an impressive accuracy rate of 96.5% after 75 training epochs. Furthermore, it achieved a 100% accuracy score when tested with random images of various plant species and diseases.


Figure 16: Accuracy and loss against.(Militante, Gerardo, and Dionisio 2019)

The Plants Village is a dataset currently consisting of 54303 images of healthy and unhealthy leaves of plants of agricultural interest, divided into 38 categories per plant species and distributed under an open source licence(Hughes and Salathé 2015), which, at the time of publication of the article, consisted of approximately 35,000 images with 32 different classes of plant varieties and diseases.

## 2.3 Investigate into Machine learning in mobile apps.

As mentioned in the project overview and background, the project will consist of developing an Android application. For this reason, it can be helpful to understand better what technologies are used, challenges, results achieved and existing similar apps.

Several Open-Source license technologies can be helpful for the CNN development and its implementation to an Android mobile app; Reda et al. (2022) argued the development of the mobile app System testing the mobile-optimised CNNs MobileNet, MobileNetV2, NasNetMobile, and EfficientNetB0, the open-source license Dataset Plant village and TensorFlow lite to the implementation of the machine learning model into the mobile app. The tests carried out on the various CNNs gave

preferences in using EfficientNetB0 as best-performing model obtaining the highest accuracy and F1-scores of 99.65%.

Introduced in 2019 by Tan and Le, EfficientNet-B0 is the smallest CNN model in the EfficientNet family and is suitable for mobile and embedded devices with limited computational resources. The EfficientNet-B0 architecture has 5.3 million parameters and is based on a compound scaling method that uniformly scales the network's depth, width, and resolution.(Tan and Le 2019) It is available in popular deep learning frameworks such as TensorFlow and Keras.

```
keras.applications.EfficientNetB0(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
    **kwargs
)
```

Figure 17: EfficientNet B0 model, Keras code(T. Keras n.d.)

Used to implement the CNN model in the mobile app, TensorFlow Lite is a set of multi-platform tools that enable on-device machine learning by helping developers run their models on mobile, embedded, and edge devices. It is optimised for on-device machine learning by addressing 5 key constraints: latency, privacy, connectivity, size, and power consumption.(Google 2022b)

### 2.3.1 Issues to consider when implementing machine learning in a mobile app.

Before implementing a Machine learning model into an Android app, it is critical to consider that the devices where the application will be installed can have limited memory, computational power and battery capability, all parameters that could affect the final user experience.

McIntosh, Hassan, and Hindle (2019) argue the impact that machine-learning functionality can have on Android devices with limited battery capability by testing different Machine-learning algorithms and datasets using the GreenMiner energy-measurement framework.

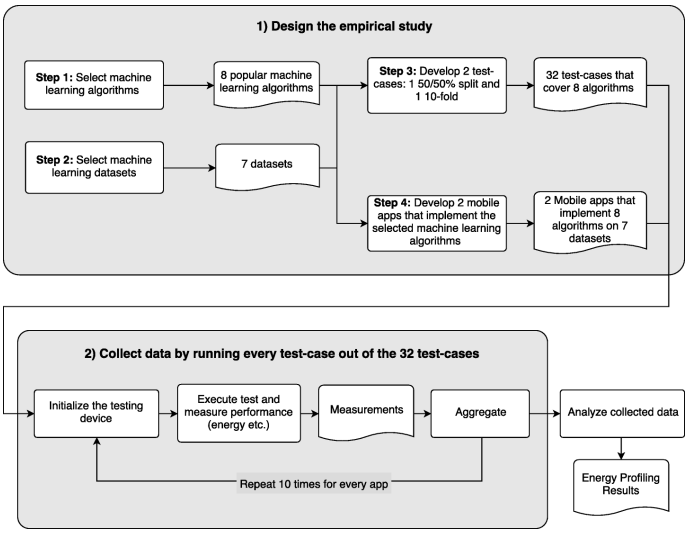



Figure 18: The overall process used to study and collect energy consumption measurements and power measurements(McIntosh, Hassan, and Hindle 2019)

The test's results indicate that certain algorithms outperform others in both energy consumption and accuracy. Additionally, energy consumption is highly correlated with algorithmic complexity. However, to achieve optimal results, developers must consider their specific application, including factors such as the size of the dataset, the number of data attributes, and whether the model will require updating.(McIntosh, Hassan, and Hindle 2019)

According to the TensorFlow Lite Overview Guide, optimising models is important for application development, especially regarding size reduction, latency reduction, and compatibility with hardware accelerators. A mobile device can benefit from an optimised model by reducing the storage space needed and download times and decreasing memory usage for improved performance. Latency reduction through optimisation positively impacts speed and power consumption during model inference. Compatibility with hardware accelerators, such as the Edge TPU, underscores the necessity of correct model optimisation.Google (2022a)

### 2.3.2 Other apps to recognise plant diseases

There are many apps for mobile devices which make use of deep learning to detect objects, animals, or plants. For the project's interest, it is necessary to mention the Agrio application.(Saillog 2023) Available for the main mobile platforms, Agrio is an app that allows the forecast, identification and treatment of plants' diseases, pests and nutrient deficits by scanning or taking a picture of the plant.

### 2.3.3 Applying Literature Insights to Project Strategy

The literature review has allowed to identify ways through which machine learning and, more specifically, Deep Learning technologies can be harnessed to progress into the project of developing an android application for recognising plant diseases through leaf images. The insights garnered, ranging from exploring the possible use of Convolutional Neural Networks (CNNs) for image recognition tasks to exploring various machine learning models in mobile app development, can provide a foundation for this project. The challenges and successes delineated in existing research underscore optimising algorithmic efficiency and ensuring model compatibility with mobile devices. These points will initially guide the project development, focusing on achieving the best content accuracy possible while maintaining the application's usability on standard mobile devices.

# 3 Problem Analysis

The project aims to develop a mobile application that can run on Android devices and will use machine learning to detect the presence of a pathogen in grapevines and apple trees, processing images captured through the device camera or already stored in it and compare the result with other examples so that to help the agricultural science students and early career farming entrepreneurs to study and reduce the margin of error in recognising those diseases.

The research method selected for this project is the "Develop and Test" methodology. This approach aligns with an Agile development model, emphasising continuous development and testing throughout the software development lifecycle.

The following section aims to illustrate what must be done to complete the project and satisfy the arguments explained in the problem statement. It will also show the method used to develop and test the application and define the main objectives to complete it.

## 3.1 project lifecycle

In order to develop the project, the agile methodology was used.


Figure 19: Agile methodology(Sahu 2021)

Agile methods prioritise continuous design, flexible scope, late freezing of design features, customer interaction, and modified project team organisation, promoting

an evolutionary, iterative, and incremental process for better efficiency.(Serrador and Pinto 2015)

The agile method approach of emphasises continuous development and testing throughout the software development lifecycle, providing frequent opportunities to review the product and make decisions and changes to the project. It also makes bug fixing easier during the project development phase. In agile, each iteration has its testing phase, allowing regression testing every time new functionalities or logic are released. Sahu (2021)

This methodology was helpful for project development as it was divided into two stages characterised by continuous research and test phases, which produced continuous changes to achieve the best result possible:

1. Stage 1: The Dataset and the Convolutional Neural Network (CNN) will be structured, developed and tested.

2. Stage 2: The Android application will be structured, developed and tested.

## 3.2 Project Development steps

**Project requirement:** The project's first stage involved setting up the computer with the software and data needed.
Following the previous research in this stage, the best software solutions to work with Python and related libraries for data science were identified and installed to provide the environment for developing the convolutional neural network, and following the knowledge gathering during the research phase, it was found the best dataset inherent the subject of the project to train and test the convolutional Neural Network tested.

The same approach was used to prepare the environment for developing the Android application using Java as the programming language. The project's progress was stored in the open source, distributed control system, and git.

**Planning:** At this stage, the test documentation to test the application was created to be sure that all features planned in the project were well integrated. In order to manage all phases of the project and respect the project deadlines, a Gantt chart was created using Microsoft Project.

**Design the interface and diagrams:** This is the stage in which the software wireframe and appropriate UML (Unified Modeling Language) diagrams were designed to simplify the Android application's development stage. Figma was used to design the software wireframe, while Visual Paradigm will be used for the app UML diagrams.

**Develop stage 1: Developed Convolutional Neural Network (CNN):** During the first development stage, The Convolutional Neural Network(CNN) models were developed and trained using a dataset structured for holding apple and grape tree leaf diseases. The trained model was then converted into a CNN model suitable for mobile applications.

**Develop stage 2: Develop the Android application:** Developed the Android application: This project development stage was focused on the Android application where the Convolution Neural Network was implemented.

**Application testing:** The application's features were tested to ensure they met the associated requirements, and the user's feedback about the application's usability, facility, learnability, and satisfaction was reported.paragraph

**Testing and evaluation:** The convolutional neural network models were tested at this stage to find the best model to use in the final application.

The convolutional neural network models were tested at this stage to find the best model based on the image classification reliability to use in the final application, and the app's possible power consumption was determined using the tool Battery Historian to see if there are different power consumption based on the CNN model implemented.

All tests were made to evaluate the software developed against the hypothesis formulated in the project outline.

**Conclusions:** The conclusions provided a final analysis of the project starting from the results of the tests carried out and the resulting evaluations.

# 4 projects Requirements

## 4.1 Requirements to develop the Convolutional Neural Network model

The project's core was developing a Convolutional Neural Network model that could detect the presence of a pathogen in grapevines and apple trees, classifying image processes.

### 4.1.1 Data acquisition

The first essential requirement for the project's feasibility consisted of identifying the best dataset of images that could contain the most significant number of pathogens on grape leaves and apple trees.

From the data collected during the literature and technology review, it was identified that, for the majority of the experiments developed on the development of Convolutional Neural network models in the field of phytopathology in the agricultural sector, the PlantVillage Dataset was the most widely used.

The dataset named "Plant_leaf_diseases_dataset_without_argumentation dataset"(J and Gopal 2019), a PlantVillage dataset version, previously used for an experiment on plant leaf disease identification, explained in the article "Identification of plant leaf diseases using a nine-layer deep convolutional neural network" (Geetharamani and Pandian 2019), is the basement of the project dataset. However, because of the limited number of images in the categories of apple and grape and because it had 30 other classes with images of other plans pathogens not necessary for the project purpose, a new dataset version was created using two augmented versions of the original PlantVillage dataset:

- **Plant_leaf_diseases_dataset_with_argumentation dataset**(J and Gopal 2019):Version of "Plant_leaf_diseases_dataset_without_argumentation dataset" dataset augmented using the techniques of image flipping, Gamma correction, noise injection, PCA colour augmentation, rotation, and Scaling and previously used for an experiment of plant leave diseases identification.
  This also implements an additional class called background_without_leaf, where images of backgrounds without leaves were collected.
- **New Plant Diseases Dataset**:It is an augmented version of the Original plantVillage dataset(Hughes and Salathé 2015) available on the Kaggle website(Bhattarai 2018) that improves the image number amount for each class

compared to the "Plant_leafe_diseases_dataset_with_argumentation" dataset but does not provide the class called background_without_leaf.

| **Plant leaf diseases dataset with argumentation** | **N.img** | **Plant leaf diseases dataset without argumentation** | **N.img** | **New Plant Diseases Dataset** | **N.img** |
|---|---|---|---|---|---|
| Apple scab | 630 | Apple scab | 1000 | Apple scab | 2016 |
| Apple black rot | 621 | Apple black rot | 1000 | Apple black rot | 1987 |
| Cedar apple rust | 275 | Cedar apple rust | 1000 | Cedar apple rust | 1760 |
| Healthy apple | 1645 | Healthy apple | 1645 | Healthy apple | 2008 |
| Backgrounds | 1143 | Backgrounds | 1143 | Grape black rot | 1888 |
| Grape black rot | 1180 | Grape black rot | 1180 | Grape esca | 1920 |
| Grape esca | 1383 | Grape esca | 1383 | Grape leaf blight | 1722 |
| Grape leaf blight | 423 | Grape leaf blight | 1076 | Healthy grape | 1692 |
| Healthy grape | 1076 | Healthy grape | 1000 | | |

Table 2: comparison of datasets used to create the final project dataset version

A new dataset using data from the above mentioned datasets was created containing only the apple and grape pathogens classes with the most significant number of images in each class. In addition, images from 2 other datasets were added to some of the existing classes of the new dataset to diversify the image variety further. In this regard, the following image datasets were used:

- **ESCA-dataset**(Alessandrini et al. 2021): This dataset provides images of leaves acquired from plants affected by Grape Esca disease.
- **Apple Tree Leaf Disease Segmentation Dataset**(Feng and Chao 2022): This dataset collects apple disease images from four different apple experimental demonstration stations of the Northwest University of Agriculture and Forestry Science and Technology in Northwest China.
  From this dataset was used, images about Cedar apple rust and apple health to improve the related pathologies classes.

The final dataset created had the following class structure and number of images for each class:

| Final dataset classes | N.images |
|---|---|
| Apple scab | 2016 |
| Apple black rot | 1983 |
| Cedar apple rust | 2102 |
| Healthy apple | 2050 |
| Backgrounds | 1143 |
| Grape black rot | 1888 |
| Grape esca | 2271 |
| Grape leaf blight | 1692 |
| Healthy grape | 1722 |

Table 3: Final dataset classes name and number of images for class.

### 4.1.2 Software Deep Learning Framework and Environment required to develop the Convolutional Neural Network model.

The Python code provided by the deep learning framework TensorFlow, which is a comprehensive, open-source deep learning framework that integrates Keras to provide a high-level API for building and training deep learning models, was used to develop the project's Convolutional Neural network model using Jupyter Notebook an open-source web application for creating and sharing computational documents.

All the software was installed on WSL2 (Windows Subsystem for Linux) to use the Linux software, utilities and command line tools on the Microsoft Windows environment with minimal impact on the system differently from traditional virtual machine or dual-boot setup(Loewen 2023).
It allowed TensorFlow 2.15.0 to use the laptop's NVIDIA's GPU card, reducing the CPU workload and decreasing the model training time.

```
[5]: import tensorflow as tf

     # Check if the system recognizes the GPU
     gpus = tf.config.list_physical_devices('GPU')
     print("Number GPUs Available: ", len(gpus))

     Number GPUs Available:  1
```

Figure 20: NVIDIA GPU card recognised to train the CNN model

## 4.2 Requirements to develop the Android application

The second step of the project consists of developing an application for Android devices that implements the Convolutional Neural Network model to allow the stakeholders to classify the leaves' images taken by the device camera or stored in it.

### 4.2.1 Software required to develop the Android application

Java and the Integrated Development Environment(IDE) Android Studio was installed to develop the Android application.

### 4.2.2 Identification of Functional and not-functional Android application requirements

Functional requirements identified to achieve the project's requirements for the Android application are:

- **Image Capture and use the image stored on the device:**The app has to allow the users to capture images using the device's camera and allow the user to access the image stored in it.
- **Pathogen Detection Using the implemented CNN model:**The application must use the Convolutional Neural Network model to detect apple and grape pathogens.
- **Comparative image Analysis:**The application has to allow the user to compare the image classified with an example image to allow the user to understand the confidence level of the classification.
- **User Interface:**The app has to provide a user-friendly interface to allow users to navigate the app easily.

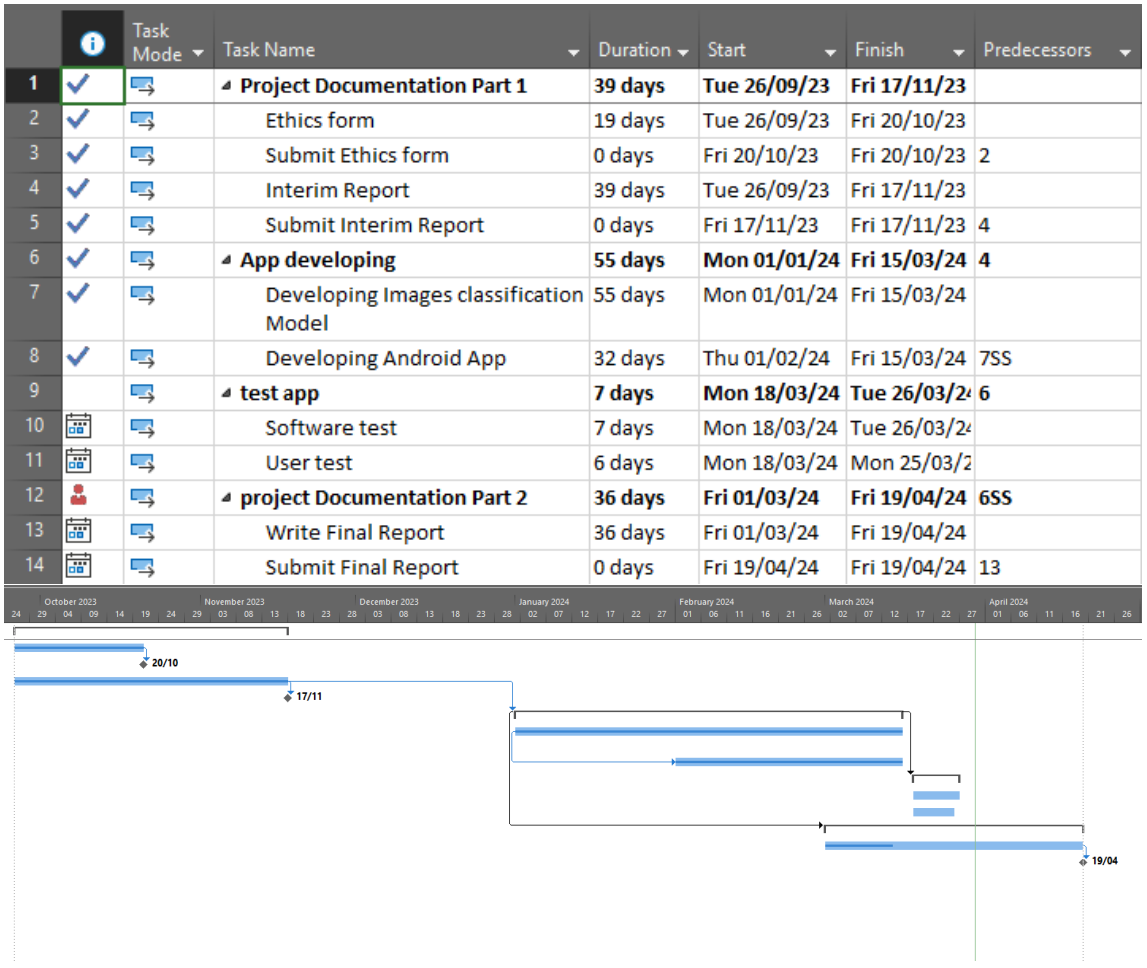
- **Reporting classification accuracy:**The app had to report the accuracy of the detection results to allow the user to understand the confidence level of the classification.

Non-Functional requirements identified to achieve the project's requirements are:

- **Performance:**The app has to process images and provide analysis results in the shortest time possible, guaranteeing optimal energy efficiency through reduced use of hardware resources.
- **Usability:**It has to have an intuitive and easy-to-navigate user interface suitable for users with different levels of tech-savviness.
- **Privacy:**The app must ensure compliance with privacy.
- **Compatibility:**The app has to be compatible across different Android devices.

# 5 projects planning

In order to manage all phases of the project and respect the project deadlines, a Gantt chart was created using Microsoft Project.



| | Task Mode | Task Name | Duration | Start | Finish | Predecessors |
|---|---|---|---|---|---|---|
| 1 | | **Project Documentation Part 1** | **39 days** | **Tue 26/09/23** | **Fri 17/11/23** | |
| 2 | | Ethics form | 19 days | Tue 26/09/23 | Fri 20/10/23 | |
| 3 | | Submit Ethics form | 0 days | Fri 20/10/23 | Fri 20/10/23 | 2 |
| 4 | | Interim Report | 39 days | Tue 26/09/23 | Fri 17/11/23 | |
| 5 | | Submit Interim Report | 0 days | Fri 17/11/23 | Fri 17/11/23 | 4 |
| 6 | | **App developing** | **55 days** | **Mon 01/01/24** | **Fri 15/03/24** | **4** |
| 7 | | Developing Images classification Model | 55 days | Mon 01/01/24 | Fri 15/03/24 | |
| 8 | | Developing Android App | 32 days | Thu 01/02/24 | Fri 15/03/24 | 7SS |
| 9 | | **test app** | **7 days** | **Mon 18/03/24** | **Tue 26/03/2** | **6** |
| 10 | | Software test | 7 days | Mon 18/03/24 | Tue 26/03/2 | |
| 11 | | User test | 6 days | Mon 18/03/24 | Mon 25/03/2 | |
| 12 | | **project Documentation Part 2** | **36 days** | **Fri 01/03/24** | **Fri 19/04/24** | **6SS** |
| 13 | | Write Final Report | 36 days | Fri 01/03/24 | Fri 19/04/24 | |
| 14 | | Submit Final Report | 0 days | Fri 19/04/24 | Fri 19/04/24 | 13 |



Figure 21: Gantt chart created to manage all phases of the project

Test documentation was also created to test the application and ensure that all features planned for the project were integrated.

| List of test cases | | | |
|---|---|---|---|
| **Test n.** | **Purpose of the test** | | |
| 1 | Testing the view of the "permission dialog." for the camera's Device access. | | |
| 3 | Testing the "Don't Allow" button in the "permission dialog." | | |
| 4 | Testing the "Only this time" button in the "permission dialog." | | |
| 5 | Testing the "While Using the app" button in the "permission dialog." | | |
| 6 | Testing all elements on the main page are visualised. | | |
| 7 | Testing the menu bar elements. | | |
| 8 | Testing the access to the photo gallery. | | |
| 9 | Testing the image classification capturing an image. | | |
| 10 | Testing the image classification selecting an image from the gallery. | | |
| 11 | Testing the correct visualisation of the image classification page result. | | |
| **date** | 01/04/24 | **System** | Android 13 go edition on Motorola Moto E 13 |

Figure 22: list of the Android Application test cases

# 6 Design the wireframe and diagrams

Following the project structure, the stage design of the project can be divided into:

1. **Convolutional Neural Network design:** Structured and analysed the design architecture of the Convolutional Neural Networks model chosen as potential candidates for the final model to be implemented in the Android application.

2. **Android application design:** Created a wireframe as an essential step to a visual representation of the app structure, layout, and functionality to be implemented, as well as the design diagrams used to facilitate the application developments.
   It was created in order:

   (a) Use Case Diagram.

   (b) Activity Diagram.

   (c) CRC card (Class Responsibility collaborator).

   (d) Initial Class Diagram.

   (e) Detailed Class Diagram.

## 6.1 Convolutional Neural Network design

To achieve a lightweight Convolutional Neural Network model without compromising performance, two custom Keras sequential models were created to develop a model that minimises resource requirements, particularly for implementation on mobile devices with limited hardware resources.

One model underwent standalone training, while the other employed transfer learning, utilising pre-trained EfficientNet B0 and EfficientNet B1 models. The following images show the design structure of the models developed:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional  (None, 7, 7, 1280)        4049571
 )

 conv2d (Conv2D)             (None, 7, 7, 32)          368672

 batch_normalization (Batch  (None, 7, 7, 32)          128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 3, 3, 32)          0
 D)

 conv2d_1 (Conv2D)           (None, 3, 3, 64)          18496

 batch_normalization_1 (Bat  (None, 3, 3, 64)          256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 1, 1, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 1, 1, 128)         73856

 batch_normalization_2 (Bat  (None, 1, 1, 128)         512
 chNormalization)

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 64)                8256

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 9)                 585

=================================================================
Total params: 4520332 (17.24 MB)
Trainable params: 470313 (1.79 MB)
Non-trainable params: 4050019 (15.45 MB)
```

Figure 23: Keras Sequential model pre-trained EfficientNet B0

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb1 (Functional  (None, 8, 8, 1280)        6575239
 )

 conv2d (Conv2D)             (None, 8, 8, 32)          368672

 batch_normalization (Batch  (None, 8, 8, 32)          128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 4, 4, 32)          0
 D)

 conv2d_1 (Conv2D)           (None, 4, 4, 64)          18496

 batch_normalization_1 (Bat  (None, 4, 4, 64)          256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 2, 2, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 2, 2, 128)         73856

 batch_normalization_2 (Bat  (None, 2, 2, 128)         512
 chNormalization)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 9)                 585

=================================================================
Total params: 7070576 (26.97 MB)
Trainable params: 494889 (1.89 MB)
Non-trainable params: 6575687 (25.08 MB)
```

Figure 24: Keras Sequential mode pre-trained EfficientNet B1

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2  (None, 90, 90, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 flatten (Flatten)           (None, 64800)             0

 dense (Dense)               (None, 128)               8294528

 dense_1 (Dense)             (None, 9)                 1161

=================================================================
Total params: 8300777 (31.66 MB)
Trainable params: 8300777 (31.66 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 25: Keras Sequential model standalone training

## 6.2 Android Application wireframe and Design diagrams

### 6.2.1 Application wireframe

The purpose of the application is to classify the photos captured by the user and show the result of the classification. In this regard, the application is reduced to a single action on the part of the user that concerns only the methodology of how the image that the Convolutional Neural Network model must process is selected and that involves capturing an image using the camera of the device or selecting the image from the device's storage.

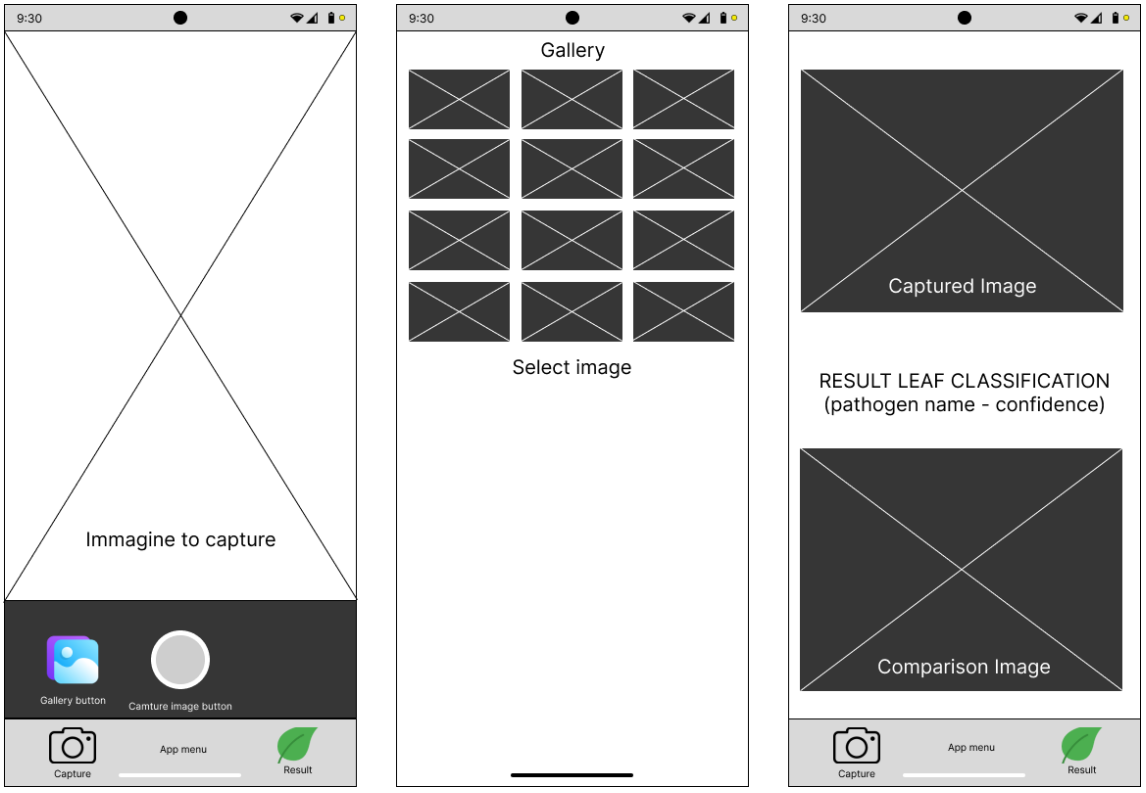



Figure 26: Android App wireframe.

The figure with the wireframe created shows, in order from the left:

- **The camera display page:** It is the application's main page for capturing an image. At the same time, a button connected to the gallery allows the user to access the images on the device.
- **The gallery page:** It is the standard one for all Android devices.

- **The result page:** Shows the classified image and the classification result, which includes the name of the recognised disease and the model confidence percentage. Another image with the pathogen detected is then displayed to allow the user to compare it with the image classified.

### 6.2.2 Use case diagram

The first diagram created involves a UML Use case diagram that specifies the expected behaviour of the application developed, showing the iteration between use cases, actor and system.

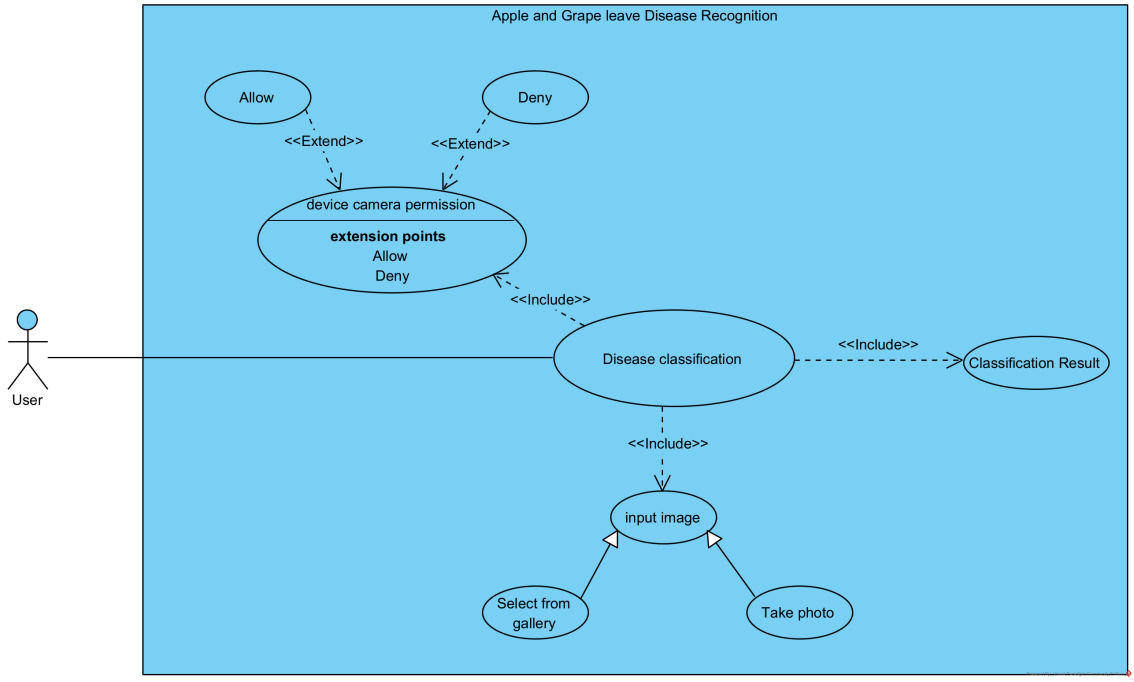



Figure 27: Android App Use case Diagram.

Following the use case diagram in the figure above, the first step is that once the user interacts with the application, the system asks for permission to access the device camera. Thus, the user can decide to allow or deny permission to the system to access the camera. The user can then capture an image through the camera or select an image from the gallery to receive the image classification result from the system.

### 6.2.3 Activity diagram

The activity diagram was created to provide a much more detailed view of the app functionality, with a step-by-step vision of what happens in a function to allow the app to work.

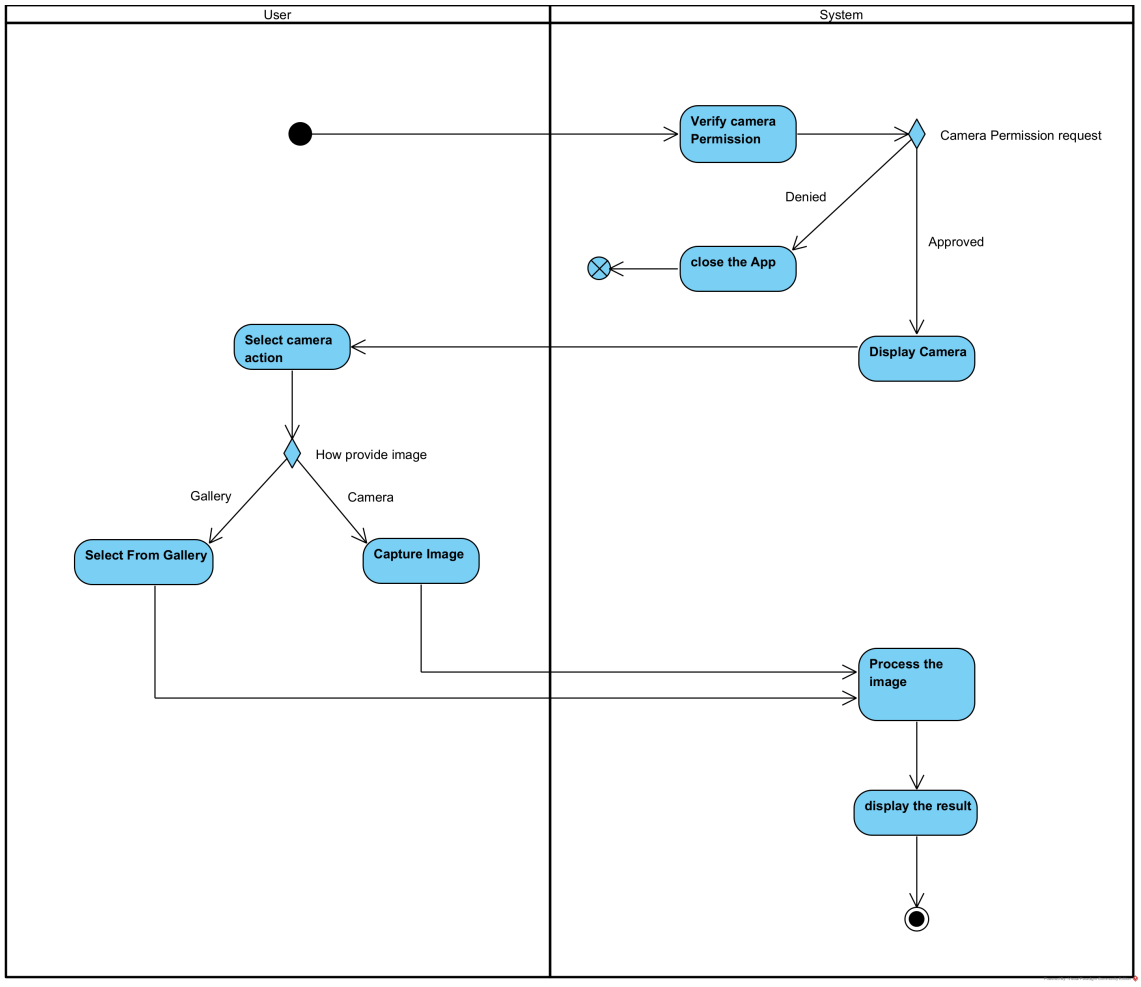



Figure 28: Android App Activity Diagram.

Following the steps in the figure above, once the user opens the app, the system asks, as the first action, permission to access the device's camera. If access to the camera is guaranteed, the system will allow the user to use the application; otherwise, it will close that.

The next step, if access to the camera is guaranteed, consists of providing the user with a way to obtain an image to be classified, choosing between selecting from the

gallery and capturing a photo. If an image is provided to the system, it will process that, returning the result of the c classification.

### 6.2.4 CRC card (Class Responsibility collaborator)

As a first step for developing the application class diagram, a CRC card (class responsibility collaborator) was created to roughly hash out the system's basic class design to facilitate the class diagram's composition.

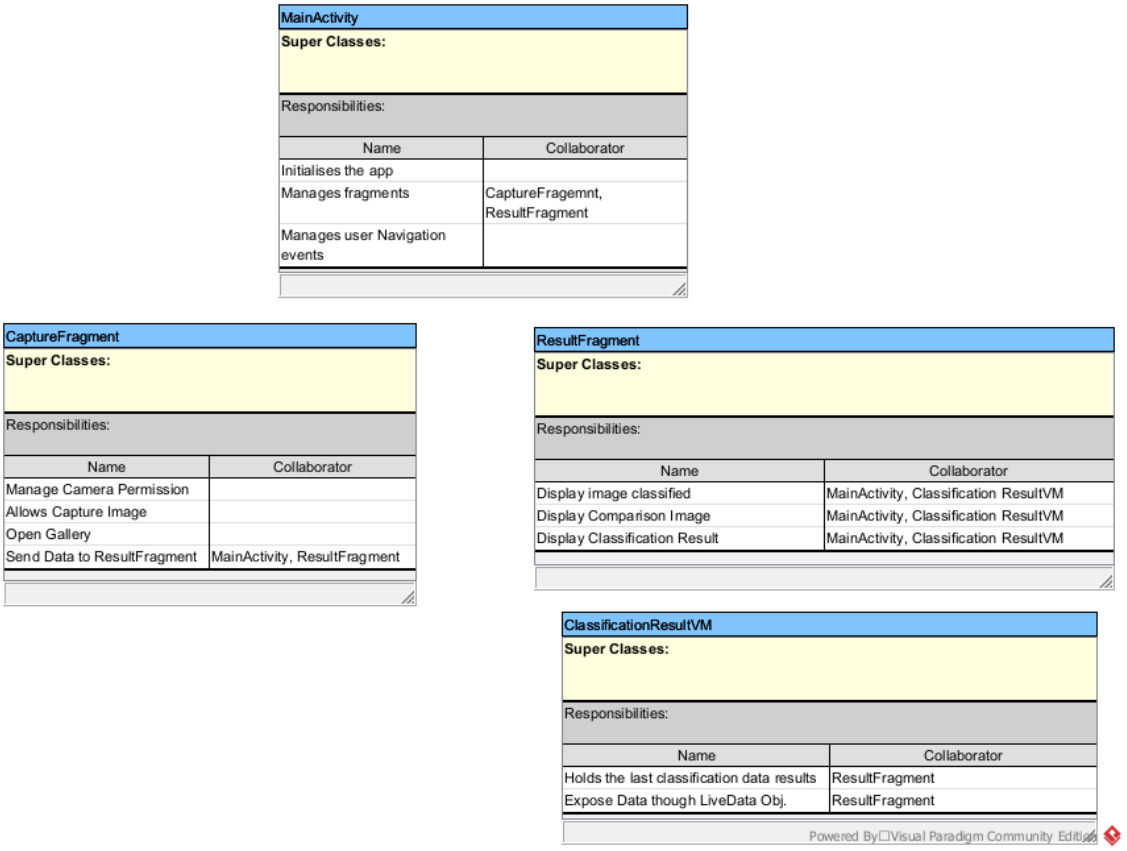



Figure 29: Android App Class Responsibility collaborator card

Because the android created uses the approach of the fragments method, the cards created are:

- **MainActvity:** initialises the app and manages the fragments. CaptureFragment: the fragment that has the purpose of managing the camera gallery use.
- **CaptureFragment:** the fragment that has the purpose of managing the camera gallery use.

- **ResultFragment:** the fragment that has the purpose of managing the classification result.
- **ClassificationResultVM:** it is the class linked to the ResultFragment that allows temporary memorisation of the data of the last image classification results until it is not made another image detection or the user does not end the app.

### 6.2.5 Class Diagram

Using the Class Responsibility collaborator cards as basement, two class diagrams were created:

- **Initial class diagram:**Without any attribute or method within each class but showing their association. For the developed app, CaptureFragment and ResultFragment are associated with the Class MainActivity that manages them and allows communication between them. ClassificationResultVM is instead associated with ResultFragment for the temporary data storage.

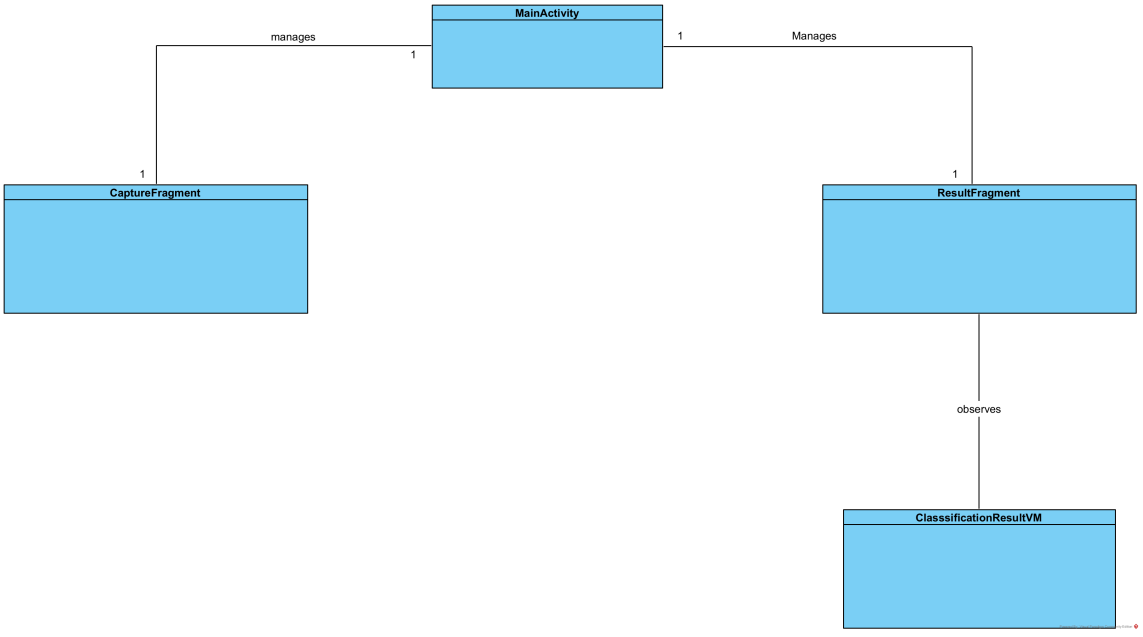



Figure 30: Android App Initial Class Diagram

- **Detailed Class Diagram:** This is the evolution of the initial class diagram, which provides each class with attributes and methods required to provide the app with all the theorised functions.

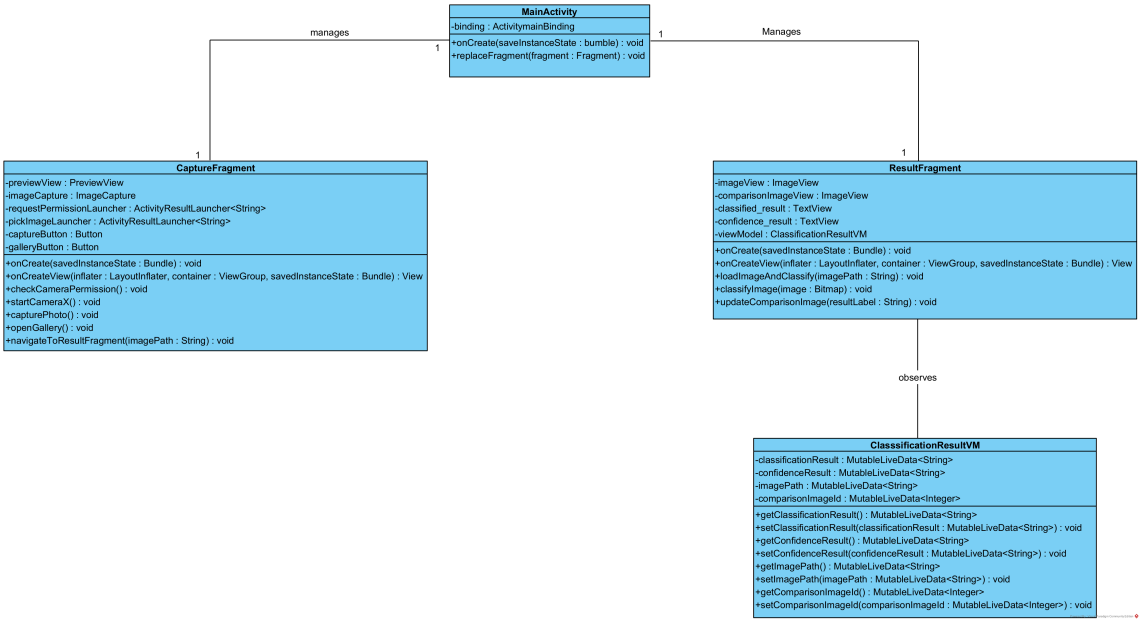



Figure 31: Android App Detailed Class Diagram

# 7 Developing stage 1: Developing Convolutional Neural Network (CNN)

The first stage of the project code implementation, consisted of developing the three convolutional neural network models chosen as potential candidates for implementation into the final Android app to classify the images of apples and grapes tree leaves with pathogens' symptoms. The stages of the models' development can be listed as follows:

1. Data Collection.
2. Data Preprocessing.
3. Model structuring.
4. Model Training.
5. Model Evaluation.
6. Model conversion for mobile devices.

## 7.1 Data collection and preprocessing

### 7.1.1 Data collection

As described in the requirement chapter, the database created was composed of images of 8 different apple and grape tree diseases and images of casual backgrounds without leaves divided into 9 classes labelled with the name of the pathogen and the reference plant (apple of grape) for a total of 16867 images.

```
#Folders(classes) in 'Dataset' directory
class_names = dataset.class_names
class_names
```

```
['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Background_without_leaves',
 'Grape___Black_rot',
 'Grape___Esca_(Black_Measles)',
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 'Grape___healthy']
```

Figure 32: Dataset classes

```
Found 16867 files belonging to 9 classes.
```

Figure 33: Dataset images count

The dataset was organised to ensure that each class contains a similar number of images.

| Final dataset classes | N.images |
|---|---|
| Apple scab | 2016 |
| Apple black rot | 1983 |
| Cedar apple rust | 2102 |
| Healthy apple | 2050 |
| Backgrounds | 1143 |
| Grape black rot | 1888 |
| Grape esca | 2271 |
| Grape leaf blight | 1692 |
| Healthy grape | 1722 |

Table 4: Final dataset classes name and number of images for class.

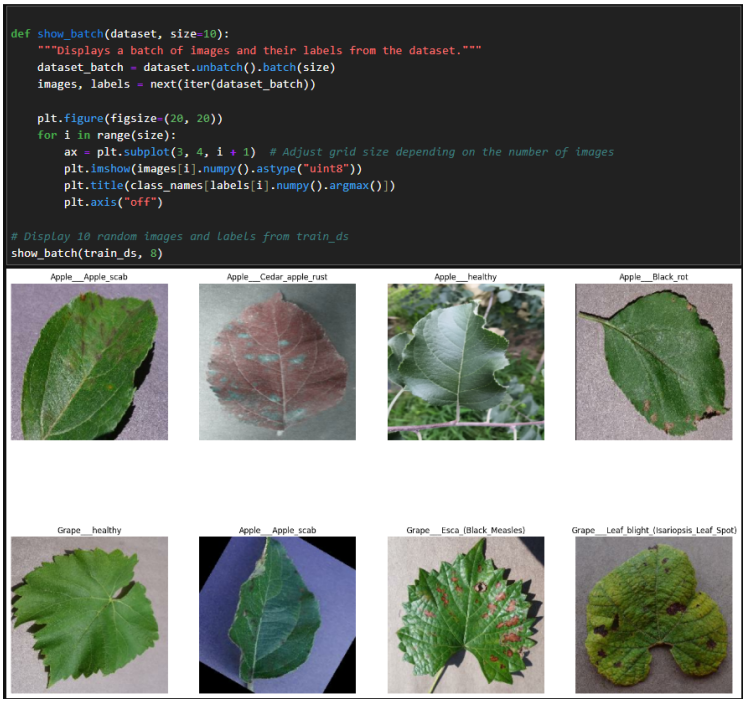



Figure 34: Batch of images from the dataset with related label

Having a similar number of images for each class allows for mitigating the risk of the model disproportionately classifying instances into the majority group due to its higher prior probability, which leads to a higher misclassification rate for minority classes.(Johnson and Khoshgoftaar 2019).

### 7.1.2 Data Preprocessing

The first step of data preprocessing consisted of dataset standardisation, which gave a default image size based on the model architecture requirement.
The model tests have the following dataset standardisation using *'tf.keras.preprocessing.image_dataset_from_directory'*:

- **Images standardisation for Keras Sequential model standalone training:** In the trial and error studies, it was decided to set the image sizes chosen as input size for this Keras sequential model as 180x180 to avoid having a large size final trained model and to reduce the impact on the computer's hardware resources.
  It was then set to 16 as the bath size of the image samples processing at once and set shuffle as True to randomise the input data order to ensure each batch had a diversified data input.(TensorFlow 2024b).

```
Image_Size = 180
Batch_Size = 16
Channels = 3

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    batch_size = Batch_Size,
    image_size = (Image_Size, Image_Size),
    shuffle = True,
    )
```

Figure 35: Images standardisation for Keras Sequential model standalone training

- **Images standardisation for Keras Sequential model pre-trained EfficientNet B0 and EfficientNet B1:** Because it was tested a keras sequential model that used EfficientNet B0 and EfficientNet B1 pre-trained models for transfer learning, it was required to resise the images based on the default input image sizes expected for these models, respectively 224x224 for EfficientNet B0 and 240x240 for EfficientNet B1.

| Model name | Input Size |
|---|---|
| EfficientNet B0 | 224x224 |
| EfficientNet B1 | 240x240 |
| EfficientNet B2 | 260x260 |
| EfficientNet B3 | 300x300 |
| EfficientNet B4 | 380x380 |
| EfficientNet B5 | 456x456 |
| EfficientNet B6 | 528x528 |
| EfficientNet B7 | 600x600 |

Table 5: EfficientNet models Family default input image sizes (Atila et al. 2021)

The batch size set for this model was 64, maintaining the shuffle to True.

```
# Standardise the images in the dataset
Image_Size = 224
Batch_Size = 64
Channels = 3

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_directory,
    batch_size = Batch_Size,
    image_size = (Image_Size, Image_Size),
    shuffle = True,
    label_mode = 'categorical'
    )
```

Figure 36: Images standardisation for Keras Sequential model pre-trained EfficientNet B0

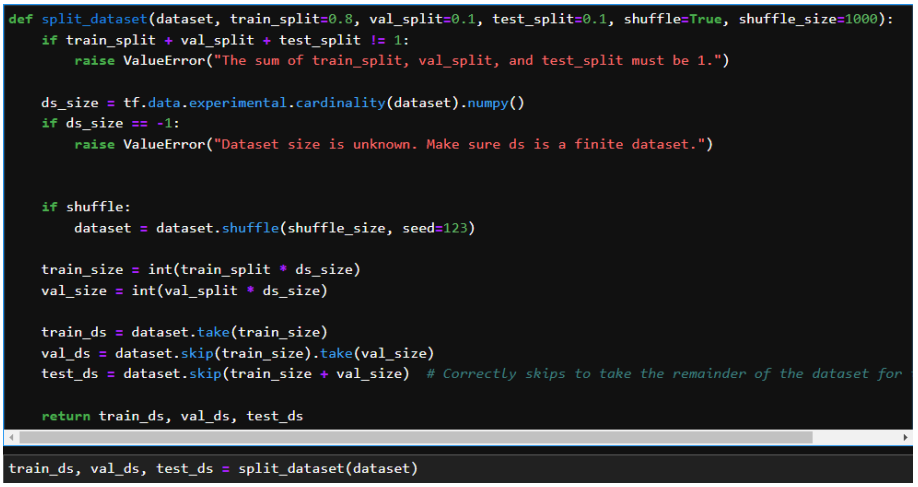
```
# Standardise the images in the dataset
Image_Size = 240
Batch_Size = 64
Channels = 3

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_directory,
    batch_size = Batch_Size,
    image_size = (Image_Size, Image_Size),
    shuffle = True,
    label_mode = 'categorical'
    )
```

Figure 37: Images standardisation for Keras Sequential model pre-trained EfficientNet B1

Once the images on the dataset were standardised, the next step involved dividing the dataset into training, validation, and testing as a fundamental practice to prepare the data for the Convolutional Neural Network model training process.
Due to the moderate amount of images contained in the dataset, corresponding to 16867 belonging to 9 classes, it was divided with the following image distribution percentage:

- training dataset (train_ds) = 80% of the dataset.
- validation dataset (val_ds) = 10% of the dataset.
- test dataset (test_ds) = 10% of the dataset.

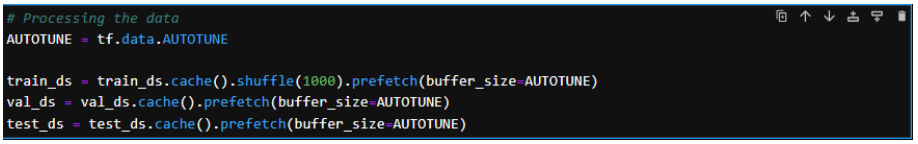
Distributing the majority of data to the training allowed the model to train in the widest amount of images possible. The same data distribution was used to train all models tested.



```python
def split_dataset(dataset, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=1000):
    if train_split + val_split + test_split != 1:
        raise ValueError("The sum of train_split, val_split, and test_split must be 1.")

    ds_size = tf.data.experimental.cardinality(dataset).numpy()
    if ds_size == -1:
        raise ValueError("Dataset size is unknown. Make sure ds is a finite dataset.")


    if shuffle:
        dataset = dataset.shuffle(shuffle_size, seed=123)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = dataset.take(train_size)
    val_ds = dataset.skip(train_size).take(val_size)
    test_ds = dataset.skip(train_size + val_size)  # Correctly skips to take the remainder of the dataset for

    return train_ds, val_ds, test_ds

train_ds, val_ds, test_ds = split_dataset(dataset)
```

Figure 38: Code used to divide the dataset into train_ds - val_ds - test_ds

In order to improve training performance, the dataset was configured using buffered prefetching, which delivers data from the disc without obstructing I/O. The methods for loading data were "dataset.cache", which stores images in memory after loading off the disc, and "dataset.prefetch", which overlaps data preprocessing and model execution during training to prevent dataset bottlenecks and create an efficient on-disk cache.(TensorFlow 2024a)



```python
# Processing the data
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Figure 39: training performance improvement using dataset.cache and dataset.prefetch

This step completed the data preprocessing process before running the model training.

## 7.2 Model Structuring and training

Two Keras sequential models were structured; one underwent standalone training, while the other employed transfer learning, utilising pre-trained EfficientNet B0 and EfficientNet B1 models, as shown in the figures below.

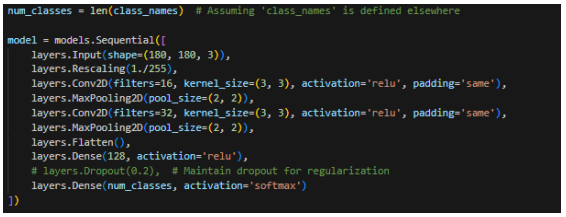



Figure 40: Keras Sequential model standalone training


Figure 41: Keras Sequential model pre-trained EfficientNet B0

```
num_classes = len(class_names)

# Load EfficientNetB0 as a base model, with pretrained weights and without the top classification layer
base_model = EfficientNetB1(
    include_top=False,
    weights="imagenet",
    input_shape=(240, 240, 3)
)

base_model.trainable = False

# Define the top layer
top_layers = [
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(num_classes, activation='softmax')
]
```

Figure 42: Keras Sequential model pre-trained EfficientNet B1

The hyperparameters learning_rate were introduced, and the number of epochs (100 epochs for the model without transferring learning and 50 epochs for the model with transfer learning) was set to achieve better model learning results.

Thus, to reduce the risk of overfitting and minimise the loss, the Early-stopping callback function was added, which stopped the model training if the model had 10 epochs without any improvement by setting the patient parameter to 10(TensorFlow 2024a), as shown in the figure below.

```python
# Define initial hyperparameters
initial_learning_rate = 0.001
epochs = 100

# Learning rate scheduler function
def scheduler(epoch, lr):
    if epoch < 10:
        return lr
    elif epoch >= 10 and epoch < 20:
        return lr * tf.math.exp(-0.1)
    else:
        return lr * tf.math.exp(-0.2)

# Define early stopping callback
early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    min_delta=1e-3,
    verbose=1,
)

# Compiled the model with the specified initial learning rate
model.compile(
    optimizer=optimizers.Adam(learning_rate=initial_learning_rate),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),  # Adjusted to match your original loss
    metrics=['accuracy']
)
```

Figure 43: Keras Sequential model standalone training Hyperparameters and epochs used

```python
model = models.Sequential([base_model] + top_layers)

# Set hyperparameters
learning_rate = 0.001
epochs = 50

# Compile the model with the specified learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

# Train the model
model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',  # Suitable for one-hot encoded labels
    metrics=['accuracy']
)
```

Figure 44: Keras Sequential model pre-trained Hyperparameters and epochs used

In the tested model, where the pre-trained models EfficientNet B0 and EfficientNet B1 were utilised for transfer learning, the process employed was fine-tuning. In the first step of the training process, the model was initialised with the pre-trained ImageNet dataset weights to leverage previously learned features for improved performance.

Additionally, all layers of the model were frozen to prevent any adverse effects during the initial training phase on a new dataset. Thus, in the second training step, the top 20 layers of the EfficientNet model were unfrozen. It allowed these layers to learn more task-specific features, improving the model's accuracy(Keras 2020).

Secont train fine-tuning

```
def unfreeze_model(base_model):
    #unfreeze the top 20 layers while leaving BatchNorm layers frozen
    for layer in base_model.layers[-20:]:
        if not isinstance(layer, layers.BatchNormalization):
            layer.trainable = True

# Compile the model with the specified learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)

model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])


# Train the model with fine-tuning
fine_history = model.fit(
    train_ds,
    validation_data= val_ds,
    epochs=epochs,
    callbacks=[early_stopping],
    validation_steps = None
)
```

Figure 45: EfficientNet B0 and B1 transfer learn Fine-tuning

Once the models were trained and their performances were tested using the test data created at this scope, they were converted to a light version optimised for mobile devices, TensorFlow lite, to be implemented into the Android application.

## 7.3 Problems encountered

Due to the limited hardware resources, the main issues encountered were the correct set of images and batch size to avoid breaking the kernel during the training phase. Many tests were performed to find the correct setting, particularly for the model with the EfficientNet B0 and B1 transfer learn test, which required the image sizes 224x224 and 240x240.

# 8 Developing stage 2: Developing the Android application

The second stage of the project code implementation consisted of developing the Android app to contain the pre-trained model chosen and implementing all the functions to allow the project stakeholders to use the image classification provided by it.
The application development can be divided into two stages:

1. **Frontend development**, involves developing the app's visual aspect and creating the application's UI (User Interface).
2. **Backend Development**, involves developing the application features that are required by the project.

The application can be run with Android 8 and later to guarantee full compatibility with a wide range of devices, and following the diagrams created during the design stage, it was easily structured the app in fragments, as shown in the figure below:

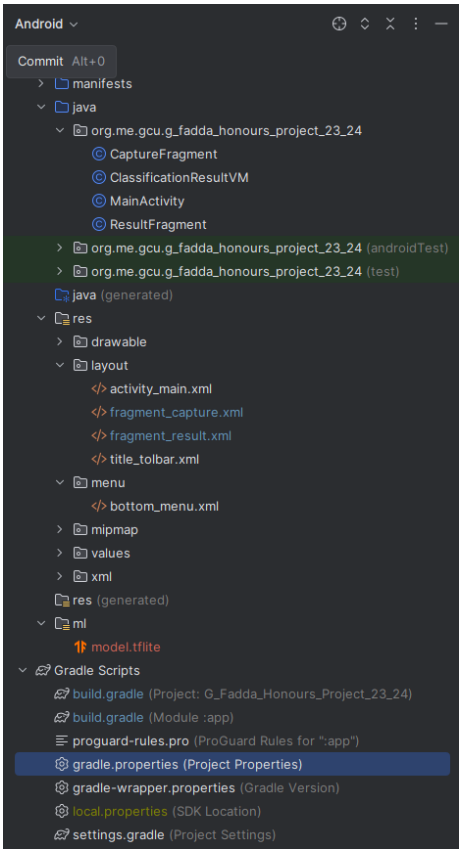



Figure 46: Application structures in Android Studio

From the figure above, it is possible to visualise, in the Java folder, the classes that compound the application backend, while in the folder layout, the XML files that provide the app a user interface. The folder named "ML" holds the TensorFlow model file that gives the app the feature of image classification.

## 8.1 App Frontend development

At this stage, the application UI was created following the wireframe created during the design stage of the project lifecycle, making it possible to create a clean and intuitive interface.

The application was developed using the method of fragments to allow modularity and reusability to the software, and following the class diagram it was created the following fragments:

### 8.1.1 Fragment_capture

It is the UI of the fragment that implements all functionality to allow the user to provide an image for the classification. It consists of a camera display with a menu with buttons to capture images and access the gallery.

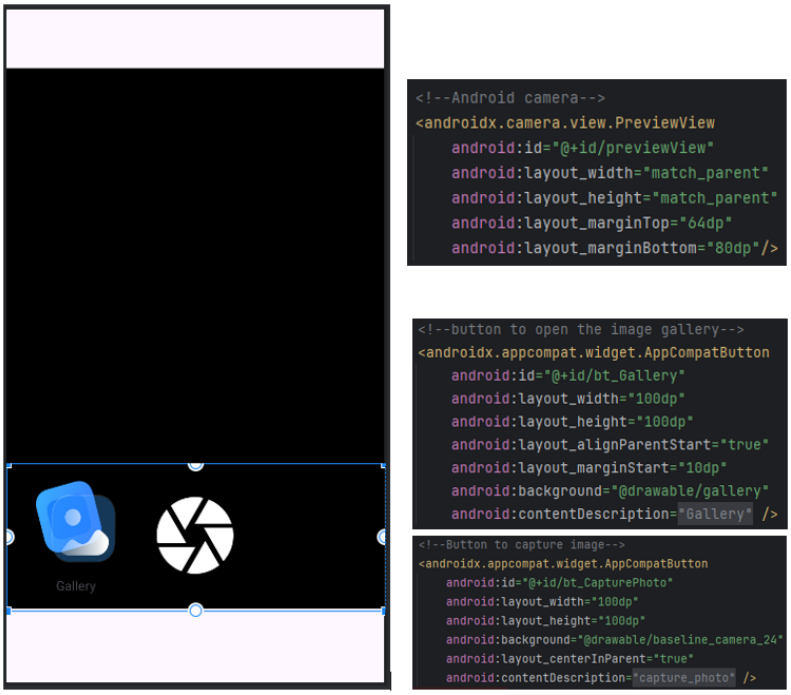



Figure 47: fragment_capture and code related to its components

### 8.1.2 Fragment_result

It is the UI of the fragment that implements the functionality to display the classification result, consisting of two cards that display the image captured and classified and an image to compare the classification result with an external example and a label text with the classification info.

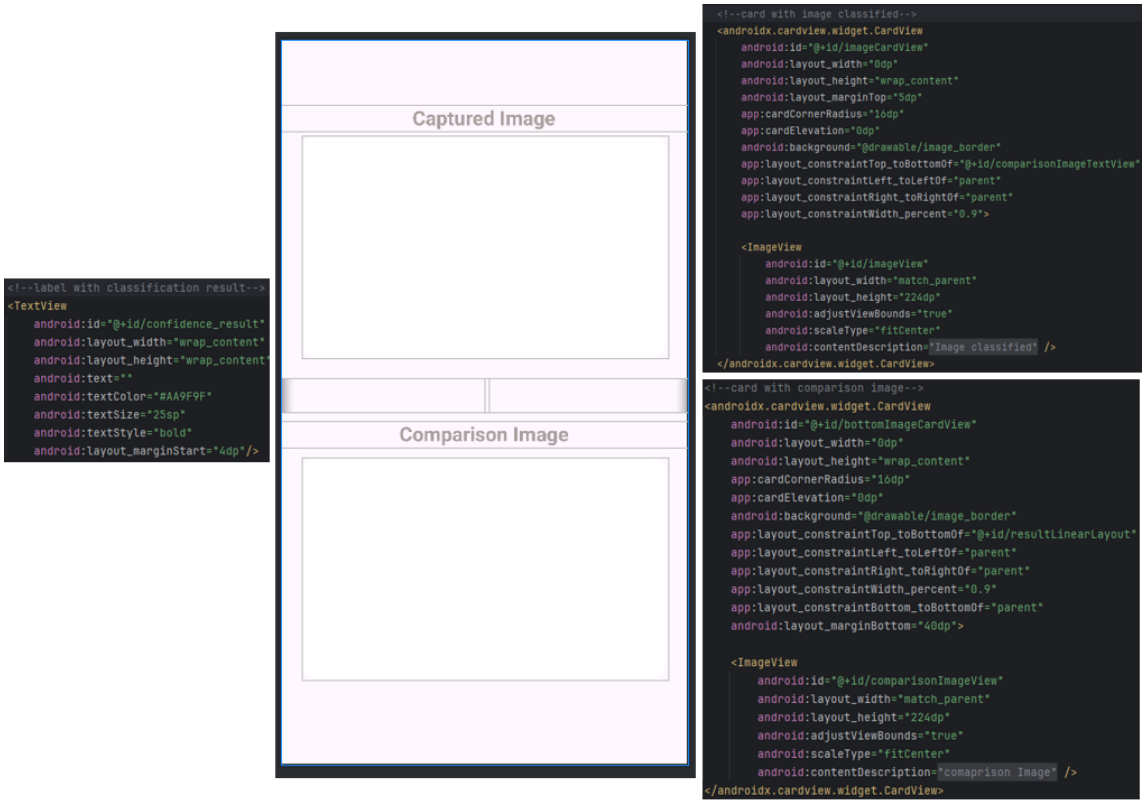



Figure 48: fragment_result and code related to its components

### 8.1.3 Fragment menu toolbar

This is the UI fragment created to allow the user to navigate around the application; located at the bottom of the app, it provides the links to the fragment capture and the fragment result.

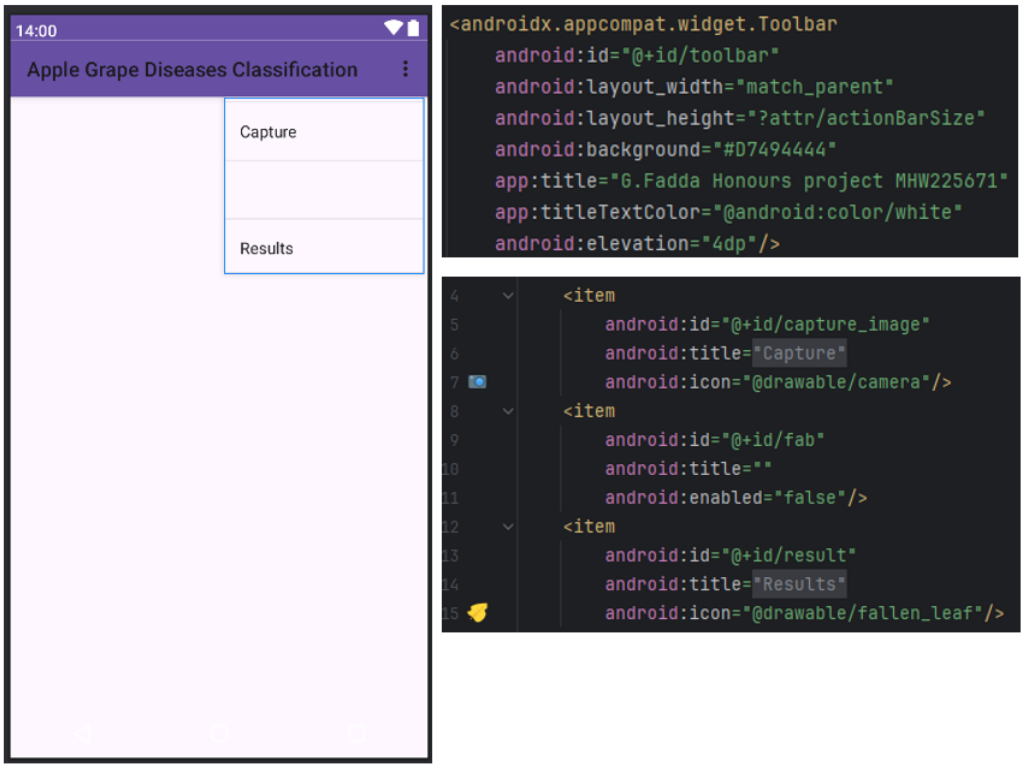



Figure 49: fragment menu toolbar

## 8.2 App Backend development

To implement the functions required to complete the project and follow the action provided in the project design lifecycle stage, the first step of the backend development involved implementing in the file gradle the latest dependencies to use the TensorFlow lite model in the app model and cameraX.

Thus, permission was added to access the camera hardware and the image gallery through the AndroidManifesto file.

As for developing the application frontend development and following the class diagram, it was structured in the following fragments and class.

### 8.2.1 CaptureFragment

Connected to the "fragment_capture" layout that provides it with the user interface, CaptureFragment is intended to provide the application with the functionality of

providing the images that the TensorFlow model uses for classification. The essential methods implemented in this fragment are:

**checkCameraPermission():**or user privacy, which checks if the user allows the app to use the device camera; if the permission is denied, the app is closed.

```
// ask permission to utilise the device camera
1 usage  Jeko86
private void checkCameraPermission() {
    if (ContextCompat.checkSelfPermission(getContext(), Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED) {
        startCameraX();
    } else {
        requestPermissionLauncher.launch(Manifest.permission.CAMERA);
    }
}
```

Figure 50: method to check device camera permission

**startCameraX():** Configures and starts the camera using the CameraX library. It sets up a preview, selects the back camera, and handles camera lifecycle events.

```
//method to start mobile camera using camerax once the permission to the hardware is allowed.
2 usages  jeko86 +1
private void startCameraX() {
    cameraProviderFuture.addListener(() -> {
        try {
            ProcessCameraProvider cameraProvider = cameraProviderFuture.get();
            Preview preview = new Preview.Builder().build();
            imageCapture = new ImageCapture.Builder().build();
            //use back camera only
            CameraSelector cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA;

            preview.setSurfaceProvider(previewView.getSurfaceProvider());
            cameraProvider.unbindAll();
            cameraProvider.bindToLifecycle( lifecycleOwner: this, cameraSelector, preview, imageCapture);
```

Figure 51: method to start cameraX

**capturePhoto():** This allows the user to capture images using the camera. It triggers a flash effect, captures the image, plays a shutter click sound, and saves the photo to a file.

```
//Method to capture photo
1 usage   Jeko86
private void capturePhoto() {
    triggerFlashEffect();

    File photoFile = new File(getOutputDirectory(), child: System.currentTimeMillis() + ".jpg");
    ImageCapture.OutputFileOptions outputFileOptions = new ImageCapture.OutputFileOptions.Builder(photoFile).build();

    //sound implementation when touched the button to take a photo
    MediaActionSound sound = new MediaActionSound();
    // standard android sound
    sound.play(MediaActionSound.SHUTTER_CLICK);

    Jeko86
    imageCapture.takePicture(outputFileOptions, ContextCompat.getMainExecutor(getContext()), new ImageCapture.OnImageSavedCallback() {
        1 usage   Jeko86
        @Override
        public void onImageSaved(@NonNull ImageCapture.OutputFileResults outputFileResults) {
            getActivity().runOnUiThread(() -> displayCapturedImage(photoFile));
        }
        Jeko86
        @Override
        public void onError(@NonNull ImageCaptureException error) { error.printStackTrace(); }
    });
}
```

Figure 52: method to capture image with device's camera

**openGallery():** This opens the device's gallery to allow the user to select an image.

```
//Method to open the gallery
1 usage   Jeko86
private void openGallery() { pickImageLauncher.launch( input: "image/*"); }
```

Figure 53: method to access to the device image gallery

**getOutputDirectory():** that determines and returns the directory where captured images should be saved.

```
1 usage  Jeko86
private File getOutputDirectory() {
    File mediaDir = requireActivity().getExternalMediaDirs()[0];
    if (mediaDir != null && mediaDir.exists()) {
        File output = new File(mediaDir, getResources().getString(R.string.app_name));
        if (!output.exists()) {
            output.mkdirs();
        }
        return output;
    }
    return requireActivity().getFilesDir();
}
```

Figure 54: method to determine and return the directory

**displaySelectedImage(Uri uri) & displayCapturedImage(File photoFile):** Handles displaying the selected or captured image by navigating to a ResultFragment with the image path passed as an argument.

```
1 usage  Jeko86
private void displaySelectedImage(Uri uri) { navigateToResultFragment(uri.toString()); }
//method to export the capture image to the fragment result for classification
1 usage  Jeko86
private void displayCapturedImage(File photoFile) {
    navigateToResultFragment(photoFile.getAbsolutePath());
}
```

Figure 55: method to display the selected or captured image

**navigateToResultFragment(String imagePath):** allows to navigate to the ResultFragment, passing the path of the selected or captured image as an argument for further processing or display.

```
//method to export the selected image to the fragment result for classification
2 usages  Jeko86
private void navigateToResultFragment(String imagePath) {
    Bundle bundle = new Bundle();
    bundle.putString("imagePath", imagePath);
    ResultFragment resultFragment = new ResultFragment();
    resultFragment.setArguments(bundle);
    getParentFragmentManager().beginTransaction()
            .replace(R.id.frame_layout, resultFragment)
            .addToBackStack( name: null)
            .commit();
}
```

Figure 56: method to esport image to the ResultFragment

### 8.2.2 ResultFragment

Connected to the "fragment_result" layout that provides it with the user interface, ResultFragment has the purpose of elaborating the images captured through the CaptureFragmant features and using the TensorFlow lite model for their classification, showing the result to the user.

The core method of the fragment_result is ClasifyImage(), which recreates all CNN steps that allow image classification within the Android application using the TensorFlow Lite model.
The part of the method that allows the image classification can be divided into:

**1. TensorFlow lite model initialisation:** The model is loaded into memory by creating a new instance of it using Model.newInstance(requireContext()) to prepare the application to use the model to process the images.

```
//load and set model to classify images imported
1 usage   Jeko86 +1 *
public void classifyImage(Bitmap image){
    try {
        //TensorFlow lite model initialisation
        Model model = Model.newInstance(requireContext());
```

Figure 57: model initialisation code segment

**2. Image Preprocessing:** In this method code segment, the input image is resized to meet the required model's input dimensions using Bitmap.createScaledBitmap (image, imageSize, imageSize, true).
Once the image is in the correct format, a TensorBuffer is initialised to match the model's expected input shape, setting up the proper structure for the input data, while the ByteBuffer is prepared to encapsulate the image data in a binary format that TensorFlow Lite can process efficiently converting data in floating-point values representing the RGB channels of each pixel to be sure that the image processed is in the correct format.

```
// Scale the selected image for classification
Bitmap scaledImage = Bitmap.createScaledBitmap(image, imageSize, imageSize, filter: true);

// Creates inputs for reference.
TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, imageSize, imageSize, 3}, DataType.FLOAT32);
ByteBuffer byteBuffer = ByteBuffer.allocateDirect( capacity: 4 * imageSize * imageSize * 3);
byteBuffer.order(ByteOrder.nativeOrder());

int[] intValues = new int[imageSize * imageSize];
scaledImage.getPixels(intValues, offset: 0, scaledImage.getWidth(), x: 0, y: 0, scaledImage.getWidth(), scaledImage.getHeight());
int pixel = 0;
//iterate over each pixel and extract R, G, and B values. Add those values individually to the byte buffer.
for(int i = 0; i < imageSize; i ++){
    for(int j = 0; j < imageSize; j++){
        int val = intValues[pixel++]; // RGB
        byteBuffer.putFloat( v: ((val >> 16) & 0xFF) * (1.f));
        byteBuffer.putFloat( v: ((val >> 8) & 0xFF) * (1.f));
        byteBuffer.putFloat( v: (val & 0xFF) * (1.f));
    }
}
```

Figure 58: Image preprocessing code segment

**3. Model Inference:** In this method code segment, ByteBuffer data are loaded into TensorBuffer, prepared for the model's input, and processed by the model, which evaluates the input and returns the output TensorBuffer that contains the confidence level for each class of the dataset used to train the model.

```
inputFeature0.loadBuffer(byteBuffer);

// Runs model inference and gets result.
Model.Outputs outputs = model.process(inputFeature0);
TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();

float[] confidences = outputFeature0.getFloatArray();
```

Figure 59: Model inference code segment

**4. result processing:** In this method code segment, the confidence score for each class is extracted from the output TensorBuffer, identifying the class with the highest score to be reported to the user.

```
// find the index of the class with the biggest confidence.
int maxPos = 0;
float maxConfidence = 0;
for (int i = 0; i < confidences.length; i++) {
    if (confidences[i] > maxConfidence) {
        maxConfidence = confidences[i];
        maxPos = i;
    }
}
```

Figure 60: result processing code segment

### 8.2.3 ClassificationResultVM class

The data of the last image classification, displayed in the Result Fragment, are temporarily left in the application memory to allow the user to visualise the last research done if they accidentally close the app or temporarily change the app page.

This temporary storage is facilitated by the ClassificationResultVM class, a ViewModel designed to cache state information and persist it across configuration changes.

ClassificationResultVM class uses MutableLiveData, a subclass of LiveData that allows the modification of its value(developer 2024) to temporarily memorise the last

image classification result until another classification is not done or the application is not fully closed.

```
public class ClassificationResultVM extends ViewModel {

    2 usages
    private MutableLiveData<String> classifiedResult = new MutableLiveData<>();
    2 usages
    private MutableLiveData<String> confidenceResult = new MutableLiveData<>();
    2 usages
    private MutableLiveData<String> imagePath = new MutableLiveData<>();
    2 usages
    private MutableLiveData<Integer> comparisonImageId = new MutableLiveData<>();
```

Figure 61: Data temporarily memorised as MutableLiveData

### 8.2.4 MainActivity class

The 2 fragments are managed and connected by the MainActivity class, the default activity of the Android apps, which also implements the menu bar to move around the 2 fragments.

```
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    // Sets the Toolbar to act as the ActionBar for this Activity window.
    setSupportActionBar(toolbar);


    binding.bottomNavigationView.setOnItemSelectedListener(item -> {
        int id = item.getItemId();
        if (id == R.id.capture_image) {
            replaceFragment(new CaptureFragment());
        }
        else if (id == R.id.result) {
            replaceFragment(new ResultFragment());
        }
        return true;
    });
}
```

Figure 62: implementation of the menu bar in MainActivity class

# 9 Application testing

Application testing involves testing all application features to ensure they meet the associated requirements.

For this purpose, it was created a test plan document with a list of test cases and, for each of them, a field that reported the precondition to test the case, the expected result, post-condiction and the result:

| List of test cases | | | |
|---|---|---|---|
| **Test n.** | **Purpose of the test** | | |
| 1 | Testing the view of the "permission dialog." for the camera's Device access. | | |
| 3 | Testing the "Don't Allow" button in the "permission dialog." | | |
| 4 | Testing the "Only this time" button in the "permission dialog." | | |
| 5 | Testing the "While Using the app" button in the "permission dialog." | | |
| 6 | Testing all elements on the main page are visualised. | | |
| 7 | Testing the menu bar elements. | | |
| 8 | Testing the access to the photo gallery. | | |
| 9 | Testing the image classification capturing an image. | | |
| 10 | Testing the image classification selecting an image from the gallery. | | |
| 11 | Testing the correct visualisation of the image classification page result. | | |
| **date** | 01/04/24 | **System** | Android 13 go edition on Motorola Moto E 13 |

| Test n. | Test cases | Pre-Conditions | Expected result | Post-Condition | Pass/Fail |
|---|---|---|---|---|---|
| 1 | Verify that the app runs. | On the mobile, select the app icon to run the app. | The app will be open, and the main page will be displayed. | The app is opened, and the main page is displayed. | **pass** Link to Image |

Figure 63: Extract of the test cases document provided

The figure above shows an extract of the test cases document provided in **Appendix B**.

Moreover, It was created a user feedback document and asked volunteer users to test the app and fill the document to provide an evaluation of the app's usability, compatibility, UI learnability, loading speed, speed on image valuation, accuracy of the image classification, and if they would suggest the app to a student or early career farming entrepreneurs. The results of the users' evaluations are available in **Appendix C**.

# 10 Evaluation

This section aims to evaluate whether the developed project can satisfy the hypothesis formulated and the research question:

**"Can a mobile application that implements Machine learning be precise enough to help agricultural science students and early-career farming entrepreneurs to train to recognise the apples and grapes diseases?"**

Therefore, this section starts from the test phase of the software developed to answer the hypothesis at the project's base, consisting of understanding the reliability of the Convolutional Neural Network model classification applied in a mobile app that consequently has to guarantee good performance in a device with limited hardware resources.

Thus, it was necessary to evaluate the models created in the development stage 1 to convert these into TensorFlow lite versions and test them through direct implementation in the Android application.

## 10.1 Convolutional Neural Network trained models testing and evaluation

The first step consists of evaluating how well the model learned to classify the images of apple and plant diseases, understanding the prediction capacity and confidence in classifying these images before converting the models into a TensorFlow lite version for the next testing step.

As described in the data preprocessing step of the developing stage 1, a dataset of images was created to test the model's confidence result, using images randomly taken from the main dataset used to train the CNN models.


Figure 64: Sequential model with standalone training images confidence results


Figure 65: Sequential model with transferring learning EfficientNet B0 images confidence results


Figure 66: Sequential model with transferring learning EfficientNet B1 images confidence results

As shown in the figure above, the results of both developed models were positive, as they recognised all the images with a confidence of 100%. Extra charts about the classification models' performance can be found in **Appendix A**.

However, models' performance differences were found during the image recognition phase once implemented in the Android application as a TensorFlow lite version of the model and using a batch of images different from those present in the test dataset.

### 10.1.1 TensorFlow lite models in the in the Android app testing and evaluation

The next step consisted of evaluating the prediction capacity and confidence in classifying the images of each model converted in the TensorFlow light version and implemented in the Android application, as well as assessing the battery power usage of the app based on the CNN model used.

Initially, this evaluation step was planned to be tested by having some farmers working in the apple and grape production field try it out.
However, this was impossible due to an evaluation error, as the project was completed at a time of year when the apple and grape plants' foliar system still needed to grow.

Therefore, a batch of images was grouped to solve the evaluation error, where each disease class that made up the dataset used to train the model was represented by two images.

Each leaf's image used for this test had a variegated background to simulate real-world conditions as much as possible, as opposed to the image of the dataset used to train the models, where most of the images have a uniform grey background as the dataset is based on laboratory images.

The result of the test using that dataset is shown in the table below:

| Image tester Name | Sequential model Results + confidence | Sequential model pre-trained EfficientNet B0 Results + confidence | Sequential model pre-trained EfficientNet B1 Results + confidence |
|---|---|---|---|
| grape_healthy(1) | apple_healthy 100% | grape_healthy 99.9% | grape_esca 67.2% |
| grape_healthy(2) | apple_healthy 100% | grape_healthy 99.1% | grape_healthy 99.7% |
| grape_isariopsis(1) | grape_isariopsis 99.9% | grape_isariopsis 95.3% | grape_isariopsis 99.9% |
| grape_isariopsis(2) | grape_isariopsis 99.5% | grape_isariopsis 98.9% | apple_cedar_rust 51.6% |
| grape_esca(1) | grape_esca 100% | grape_esca 100% | grape_esca 100% |
| grape_esca(2) | grape_esca 100% | grape_esca 100% | grape_esca 100% |
| grape_black_rot(1) | no_leaves 96.7% | grape_black_rot 94.8% | grape_esca 83.3% |
| grape_black_rot(2) | no_leaves 98.7% | grape_black_rot 100% | grape_black_rot 93.9% |
| apple_scab(1) | grape_esca 100% | apple_scab 98.9% | grape_esca 100% |
| apple_scab(2) | apple_healthy 100% | apple_scab 95.6% | apple_black_rot 99.5% |
| apple_healthy(1) | apple_healthy 100% | apple_healthy 100% | apple_healthy 100% |
| apple_healthy(2) | apple_healthy 100% | apple_healthy 100% | apple_healthy 100% |
| apple_cedar_rust(1) | apple_cedar_rust 94.1% | apple_cedar_rust 100% | apple_cedar_rust 100% |
| apple_cedar_rust(2) | grape_esca 100% | apple_cedar_rust 99% | apple_cedar_rust 97% |
| apple_black_rot(1) | apple_black_rot 100% | apple_black_rot 100% | apple_black_rot 100% |
| apple_black_rot(2) | apple_black_rot 99.9% | apple_black_rot 100% | apple_black_rot 100% |
| no_leaves(1) | no_leaves 100% | no_leaves 100% | no_leaves 100% |
| no_leaves(2) | no_leaves 100% | no_leaves 100% | no_leaves 100% |

Table 6: Results of image classification of the models created and implemented in the Android app

The table above shows the TensorFlow lite model image classification results and confidence, highlighting for each model, in green, the correct image classification result and in red, the wrong one.

The model that could recognise 100% of the dataset images correctly was the Keras sequential model with transferring learning from the pre-trained EfficientNet B0.
Whereas the Keras Sequential model with standalone training correctly classified 61.11% of the images and the Keras sequential model with transferring learning

from the pre-trained EfficientNet B1 72.2% of them.

However, how the classification of the image can be affected by the quality of the image and the focus of the subject must be highlighted.

The same image may report different classification and confidence results if it has a background that shows many non-blurred elements or if the image highlights only the main subject by cropping the background or blurring it, as shown in the image below.

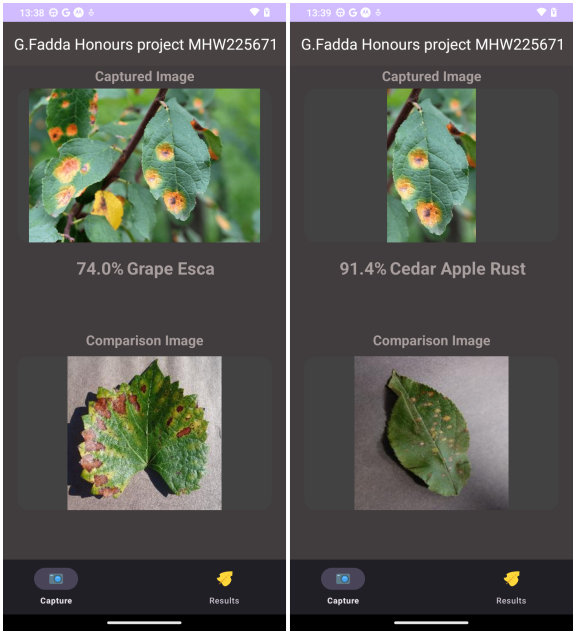



Figure 67: Example of the same image classified with and without background

The figure above shows the difference in the classification results of the same image with the main subject having a background with non-blurred elements, where the model predicted the wrong result, and the same image where from the main subject the background was cropped, reporting a correct prediction.

The Battery Historian tool was then used to test the three models' energetic efficiency, and the same batch of images used to test the TensorFlow lite models' prediction capacity and confidence.

Each model was separately implemented in the app. All the applications opened in the device's memory were closed, and battery data gathering was reset each time

the app with a different model was tested.
The same bath of images was then classified three times consecutively for a total of 54 image classifications.

The results did not show appreciable differences in the use of the app using one model instead of another model, as shown in the figures below:

| | |
|---|---|
| Application | org.me.gcu.g_fadda_honours_project_23_24 |
| Version Name | 1.0 |
| Version Code | 1 |
| UID | 10199 |
| Device estimated power use | 0.33% |
| Foreground | 58 times over 1m 51s 385ms |
| CPU user time | 1m 23s 396ms |
| CPU system time | 15s 654ms |
| Device estimated power use due to CPU usage | 0.00% |
| Camera | 81 times for a total duration of 1m 37s 630ms |

(a) Sequential model with standalone training estimate power used results

| | |
|---|---|
| Application | org.me.gcu.g_fadda_honours_project_23_24 |
| Version Name | 1.0 |
| Version Code | 1 |
| UID | 10197 |
| Device estimated power use | 0.31% |
| Foreground | 58 times over 2m 3s 815ms |
| CPU user time | 1m 26s 7ms |
| CPU system time | 12s 854ms |
| Device estimated power use due to CPU usage | 0.00% |
| Camera | 86 times for a total duration of 1m 30s 58ms |

(b) Sequential model with transferring learning EfficientNet B0 estimate power used results

| | |
|---|---|
| Application | org.me.gcu.g_fadda_honours_project_23_24 |
| Version Name | 1.0 |
| Version Code | 1 |
| UID | 10198 |
| Device estimated power use | 0.33% |
| Foreground | 58 times over 2m 12s 457ms |
| CPU user time | 1m 33s 748ms |
| CPU system time | 13s 727ms |
| Device estimated power use due to CPU usage | 0.00% |
| Camera | 89 times for a total duration of 1m 31s 990ms |

(c) Sequential model with transferring learning EfficientNet B1 estimate power used results

Figure 68: Models estimate power used results

The figures shows the same estimated power use of 0.33% for the Sequential model with standalone training and the Sequential model with transferring learning EfficientNet B1 and 0.31% for the Sequential model with transferring learning EfficientNet B0.
These data do not justify the use of one model rather than another.

Thus, the choice of the model to implement in the final application was based on the test of the prediction capacity and confidence in classifying, and based on the obtained results, it was decided to utilise the Sequential model with transferring learning EfficientNet B0.

## 10.2 Evaluation conclusion

In conclusion, the tests' results allowed answers to the hypothesis of the project, demonstrating that setting well the parameters of the Convolutional Neural Network can achieve satisfactory results in the recognition of apple and grape plants' diseases visible in the foliar system without excessive losses of reliability when implemented in an application for Android device.

Among the tested options, the Keras sequential model transfers learnings from the pre-trained EfficientNet B0, reports the highest reliability on all tested images, and even though the reliability of the classification can be influenced by whether the subject to be classified is placed in the foreground or not in the image selected or captured, it must be considered that, given the purpose of the application, it is necessary to analyse leaf by leaf and not the entire plant.

# 11 Legal, Social, Ethical and Professional Issues

No potential legal, social, ethical or professional problems were found for the project's development.

## 11.1 material with copyright licence

To train the Convolutional Neural Network models it was used images taken from datasets licensed under a Public Domain licence or Attribution NonCommercial ShareAlike as follows:

- **Plant leaf diseases dataset with argumentation dataset**(J and Gopal 2019), under licence CC0 1.0
- **New Plant Diseases Dataset**(Hughes and Salathé 2015), under licence CC0 1.0
- **ESCA-dataset**,(Alessandrini et al. 2021) under licence CC BY-NC-SA 4.0 DEED
- **Apple Tree Leaf Disease Segmentation Dataset**,(Feng and Chao 2022) under licence CC BY-NC-SA 4.0 DEED

## 11.2 User data protection

It was created a document with multiple choice questions to allow users to provide an evaluation of the app's usability, compatibility, UI learnability, loading speed, speed on image valuation, the accuracy of the image classification, and if they would suggest the app to a student or early career farming entrepreneurs.

To guarantee user privacy was not collected any sensitive data; only asked to provide the Brand and the model of the device used to test the app and and the Android version installed.

The application then does not collect any data, as it works offline only and can access the device camera as an instrument to capture images for the CNN classification, subject to user authorisation asked the first time the app is run through a permission dialog.

The people who offered to test the app were all adults over 18. The complete evaluation document can be found in the **appendix C**.

# 12 Conclusions

## 12.1 Project Resume

Among the main Italian agricultural products grown are grapes for wine production and apples, of which Italy is one of the leading producers in the world.

In order to preserve these products from potential pathogens that can dramatically affect production with a significant economic loss and in perspective of a future scenario where climate changes can also impact the incidence of plant pathogens as some new study seems to suggest, it is essential to try to reduce or eradicate this problem through the detection, monitoring and management of potential pathogens in the early stages that can often manifest themselves on the foliar system of the plants, which symptoms result challenging to recognise for an untrained eye of a student and an early career farming entrepreneur.

With today's high presence of portable devices and the current level of maturity of Machine Learning technologies, the research question that the project tried to satisfy is:

**Can a mobile application that implements Machine learning be precise enough to help agricultural science students and early-career farming entrepreneurs to train to recognise the apples and grapes diseases?**

Thus, to achieve this result, the project involved the develop a mobile application that can run on Android devices and use machine learning to detect the presence of pathogen symptoms in the foliar system of the plants, processing images captured through the device camera or already stored in it.

Following the stages of the Agile method, it was found the project's requirements, 3 potential Convolutional Neural Network model designs, created the Android application wireframe and diagrams to facilitate the Android application development stage, Developed the CNN models and implemented them in the Android application developed to deploy it an Android device, tested all features to ensure they meet the associated requirements and chose the best model to implement into the final app concluding evaluating the project to answer the research question and satisfy the hypothesis.

## 12.2 Discussion of the results

The evaluation of the project was mainly focused on testing the 3 models developed to choose the one that guaranteed the best reliability in recognising the apple and grape diseases visible on the foliar system before and after their conversion into a light version for mobile devices, TensorFlow lite, the potential power energy usage of the Android application with each model implemented.

### 12.2.1 Convolutional Neural Network models testing conclusions

The results showed that the models provided good results in classifying the plants' disease images, and even if after the conversion in TensorFlow lite, two models tended to reduce their classification performances, one of them, the sequential model with transfer learning from EfficientNet B0, guaranteed good classification reliability with all the images provided that covered all the disease classes that composed the dataset used to train the models.

The test of the power energy usage of the Android application with each model implemented due to the similarity of the 3 models, and the small-scale type of project does not report appreciable differences between the various models.

### 12.2.2 Overall Conclusions

Overall, the project satisfied the research question by developing an Android application capable of recognising the images of pathogen symptoms in the foliar system of the apple and grape plants.

The project results align with similar projects and models' classification results identified during the project feasibility research, giving students and early-career farming entrepreneurs a good instrument to improve their knowledge in recognising apple and grape diseases.

## 12.3 Limitation and Future work

### 12.3.1 Limitations

Adopting a critical approach to the project result, even satisfying the research question and hypothesis, is essential to notice some limitations that affected the project development.

This first limitation is due to the dataset used to train the models, which results to be relatively small, comporting the risk of model overfitting during the training stage, and because the dataset provides only leaves images without a variegated background, the image classification sometimes reported the wrong result if the main subject is not well focused.
However, this limitation can be mitigated by having a close-up of the subject to classify or implement the dataset images with a varied background to train the model.

The other limitation was the lack of possibility of testing the application on apple and grape plants as they did not yet develop a leaf system due to the season.

### 12.3.2 Future work

Overall, the application developed can be considered a valid tool to help agricultural science students and early-career farming entrepreneurs recognise the diseases of apples and grapes, giving them a further tool to integrate with more traditional learning methods.

The results of this project, including the current limitations, can be used as a further starting point for future projects by testing new models that can guarantee an additional level of reliability for the application and by developing a more complete and varied dataset image that can further mitigate evaluation errors.

# References


Abbaspour-Gilandeh, Yousef et al. (2022). "Feasibility of Using Computer Vision and Artificial Intelligence Techniques in Detection of Some Apple Pests and Diseases". In: *Applied sciences* 12.2, p. 906. DOI: 10.3390/app12020906.

Abdallah, Ali (2019). *PlantVillage Dataset*. Kaggle.com.

Alessandrini, Michele et al. (2021). *ESCA-dataset*. Version V1. Mendeley Data. DOI: 10.17632/89cnxc58kj.1.

Atila, Ümit et al. (2021). "Plant leaf disease classification using EfficientNet deep learning model". In: *Ecological informatics* 61, p. 101182. DOI: 10.1016/j.ecoinf.2020.101182.

Belete, T. and N. Boyraz (2017). "Critical review on apple scab (Venturia inaequalis) biology, epidemiology, economic importance, management and defense mechanisms to the causal agent". In: *J.Plant Physiol.Pathol* 5.2, p. 2.

Bhatt, Dulari et al. (2021). "Cnn variants for computer vision: History, architecture, application, challenges and future scope". In: *Electronics (Basel)* 10.20, p. 2470. DOI: 10.3390/electronics10202470.

Bhattarai, Samir (2018). *New Plant Diseases Dataset*.

Bregaglio, Simone, Marcello Donatelli, and Roberto Confalonieri (2013). "Fungal infections of rice, wheat, and grape in Europe in 2030–2050". In: *Agronomy for sustainable development* 33, pp. 767–776.

Caffarra, Amelia et al. (2012). "Modelling the impact of climate change on the interaction between grapevine and its pests and pathogens: European grapevine moth and powdery mildew". In: *Agriculture, Ecosystems Environment* 148, pp. 89–101. DOI: 10.1016/j.agee.2011.11.017.

Chowdhary, K. R. (2020a). *Fundamentals of Artificial Intelligence*. 1st ed. New Delhi: Springer India. ISBN: 81-322-3972-5. DOI: 10.1007/978-81-322-3972-7.

— (2020b). *Fundamentals of Artificial Intelligence*. 1st ed. New Delhi: Springer India. ISBN: 81-322-3972-5. DOI: 10.1007/978-81-322-3972-7.

Ciancio, Aurelio and K. G. Mukerji (2008). *Integrated Management of Diseases Caused by Fungi, Phytoplasma and Bacteria*. 1st ed. Dordrecht: Springer Netherlands. ISBN: 1-281-86151-0. DOI: 10.1007/978-1-4020-8571-0.

Corsi, Alessandro, Simonetta Mazzarino, and Eugenio Pomarici (2019). "The Italian wine industry". In: *The Palgrave handbook of wine industry economics*, pp. 47–76.

developer, Android (Jan. 2024). *LiveData overview*.

Ellis, M (2008). "Apple powdery mildew". In: *The Ohio State University Extension, Fact Sheet, Agriculture and Natural Resources, HYG-3001-08*.

Eurostat (Oct. 2018). *Archive:Agricultural census in Italy*.

— (Jan. 2019). *Agricultural production - orchards*.

— (May 2022). *Vineyards in the EU - statistics*.

Eurostat (Oct. 2023). *Agricultural production - crops*.
FAO (Dec. 2023). *FAOSTAT*.
Feng, Jingze and Xiaofei Chao (Mar. 2022). *Apple Tree Leaf Disease Segmentation Dataset*. Version V1. DOI: 10.11922/sciencedb.01627.
Fergus, Paul and Carl Chalmers (2022). *Applied deep learning : tools, techniques, and implementation*. Cham, Switzerland: Springer. ISBN: 3-031-04420-7.
Francesca, Salinari et al. (2006). "Downy mildew (Plasmopara viticola) epidemics on grapevine under climate change". In: *Global Change Biology* 12.7, pp. 1299–1307.
Geetharamani, G and Arun Pandian (2019). "Identification of plant leaf diseases using a nine-layer deep convolutional neural network". In: *Computers & Electrical Engineering* 76, pp. 323–338.
Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. Cambridge, Massachusetts: The MIT Press. ISBN: 9780262035613.
Google (Aug. 2022a). *Build tensorflow lite models*.
— (May 2022b). *Tensorflow Lite*.
Al-Hiary, Heba et al. (2011). "Fast and accurate detection and classification of plant diseases". In: *International Journal of Computer Applications* 17.1, pp. 31–38.
Hoeser, Thorsten and Claudia Kuenzer (2020). "Object detection and image segmentation with deep learning on Earth observation data: A review-part I: Evolution and recent trends". In: *Remote sensing (Basel, Switzerland)* 12.10, p. 1667. DOI: 10.3390/rs12101667.
Hughes, David P. and Marcel Salathé (2015). "An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing". In: *CoRR* abs/1511.08060. arXiv: 1511.08060.
J, Arun Pandian and Geetharamani Gopal (2019). *Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network*. https://data.mendeley.com/datasets/tywbtsjrjv/1. Version V1. DOI: 10.17632/tywbtsjrjv.1.
Ji, Miaomiao, Lei Zhang, and Qiufeng Wu (2020). "Automatic grape leaf diseases identification via UnitedModel based on multiple convolutional neural networks". In: *Information processing in agriculture* 7.3, pp. 418–426. DOI: 10.1016/j.inpa.2019.10.003.
Jo, Taeho (2021). *Machine learning foundations : supervised, unsupervised, and advanced learning*. 1st ed. Cham, Switzerland: Springer. ISBN: 3-030-65900-3. DOI: 10.1007/978-3-030-65900-4.
Johnson, Justin M. and Taghi M. Khoshgoftaar (2019). "Survey on deep learning with class imbalance". In: *Journal of big data* 6.1, pp. 1–54. DOI: 10.1186/s40537-019-0192-5.

Jordão, António M., Renato V. Botelho, and Uroš Miljić (2023). *Recent Advances in Grapes and Wine Production New Perspectives for Quality Improvement*. London, UK.

Kabir, Muhammad Mohsin, Abu Quwsar Ohi, and M. F. Mridha (2021). "A Multi-Plant Disease Diagnosis Method Using Convolutional Neural Network". In: Singapore: Springer Singapore, pp. 99–111. ISBN: 2524-7565. DOI: 10.1007/978-981-33-6424-0_7.

Keras (June 2020). *Keras documentation: Image classification via fine-tuning with EfficientNet*.

Keras, Team (n.d.). *Keras documentation: EfficientNet B0 to B7*.

Khaled, Alfadhl Yahya et al. (2018). "Early detection of diseases in plant tissue using spectroscopy - applications and limitations". In: *Applied spectroscopy reviews* 53.1, pp. 36–64. DOI: 10.1080/05704928.2017.1352510.

Loewen, Craig (Aug. 2023). *Install WSL*.

McIntosh, Andrea, Safwat Hassan, and Abram Hindle (2019). "What can Android mobile app developers do about the energy consumption of machine learning?" In: *Empirical software engineering : an international journal* 24.2, pp. 562–601. DOI: 10.1007/s10664-018-9629-2.

Militante, Sammy V, Bobby D Gerardo, and Nanette V Dionisio (2019). "Plant leaf detection and disease recognition using deep learning". In: *2019 IEEE Eurasia conference on IOT, communication and engineering (ECICE)*. IEEE, pp. 579–582.

Pasquier, Grégory et al. (2013). "Impact of foliar symptoms of "Esca proper" on proteins related to defense and oxidative stress of grape skins during ripening". In: *Proteomics (Weinheim); Proteomics* 13.1, pp. 108–118. DOI: 10.1002/pmic.201200194.

"Pathogens, precipitation and produce prices" (2021). In: *Nature climate change* 11.8, p. 635. DOI: 10.1038/s41558-021-01124-4.

Reda, Mariam et al. (2022). "Agroaid: A mobile app system for visual classification of plant species and diseases using deep learning and tensorflow lite". In: *Informatics*. Vol. 9. 3. MDPI, p. 55.

Ristaino, Jean B. et al. (2021). "The persistent threat of emerging plant disease pandemics to global food security". In: *Proceedings of the National Academy of Sciences - PNAS* 118.23, p. 1. DOI: 10.1073/pnas.2022239118.

Rossi, Vittorio et al. (2015). "Use of systems analysis to develop plant disease models based on literature data: grape black-rot as a case-study". In: *European Journal of Plant Pathology* 141.3, pp. 427–444. DOI: 10.1007/s10658-014-0553-z.

Sahu, Niket (Oct. 2021). *Qa Agile Methodologies*.

Saillog, LTD (2023). *Agrio*.

Serrador, Pedro and Jeffrey K. Pinto (2015). "Does Agile work? — A quantitative analysis of agile project success". In: *International Journal of Project Management* 33.5, pp. 1040–1051. DOI: 10.1016/j.ijproman.2015.01.006.

Sewak, Mohit, Md Rezaul Karim, and Pradeep Pujari (2018). *Practical convolutional neural networks: implement advanced deep learning models using Python*. 1; 1; 1. Birmingham: PACKT Publishing. ISBN: 9781788392303.

Szabó, Márton et al. (2023). "Black Rot of Grapes (Guignardia bidwellii)—A Comprehensive Overview". In: *Horticulturae* 9.2, p. 130. DOI: 10.3390/horticulturae9020130.

Tan, Mingxing and Quoc Le (2019). "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR, pp. 6105–6114.

Team, Dataneb (Oct. 2019). *What's Artificial Intelligence, Machine Learning, deep learning, Predictive Analytics, data science?* https://www.dataneb.com/post/artificial-intelligence-machine-learning-deep-learning-predictive-analytics-data-science.

TensorFlow (Mar. 2024a). *Image classification — TensorFlow Core*.

— (Mar. 2024b). *tf.keras.preprocessing.image*$_{d}ataset_{f}rom_{d}irectory$.

Thapa, Ranjita et al. (2020). "The Plant Pathology Challenge 2020 data set to classify foliar disease of apples". In: *Applications in plant sciences; Appl Plant Sci* 8.9, e11390–n/a. DOI: 10.1002/aps3.11390.

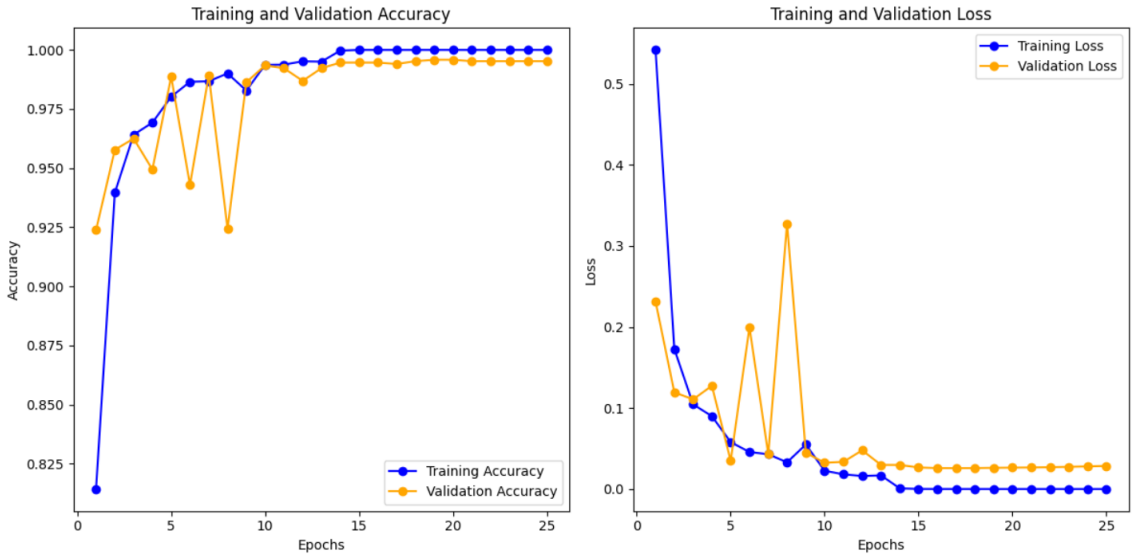
Turechek, William (2023). *Powdery mildew of apple and pear*. Michigan State University.

Vogelweith, F. and D. Thiéry (2017). "Cover crop differentially affects arthropods, but not diseases, occurring on grape leaves in vineyards". In: *Australian journal of grape and wine research* 23.3, pp. 426–431. DOI: 10.1111/ajgw.12290.

Wei, Dr. Shouke (July 2023). *Practical guide to create a Convolutional Neural Network (CNN) model*.

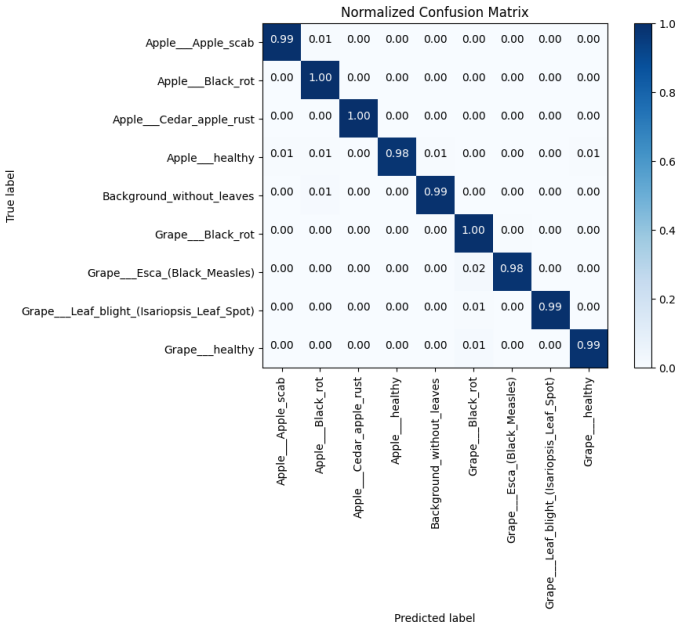
# A Appendix - CNN Models classification performance

## Sequential model with standalone training results

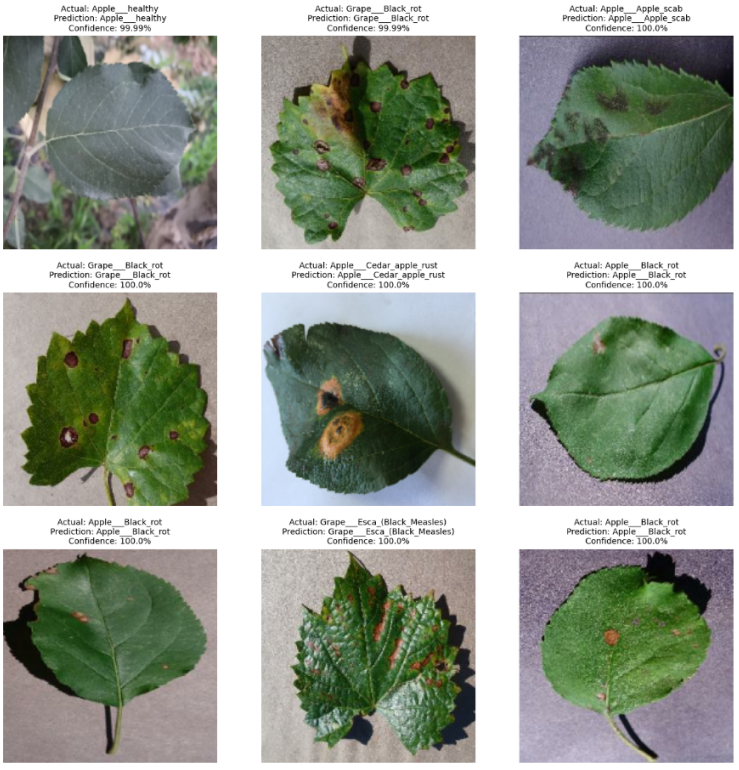



### Classification Performance Metrics

| Class Name | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Apple___Apple_scab | 0.99 | 0.99 | 0.99 | 200 |
| Apple___Black_rot | 0.99 | 1.00 | 0.99 | 225 |
| Apple___Cedar_apple_rust | 1.00 | 1.00 | 1.00 | 198 |
| Apple___healthy | 1.00 | 0.98 | 0.99 | 196 |
| Background_without_leaves | 0.99 | 0.99 | 0.99 | 117 |
| Grape___Black_rot | 0.97 | 1.00 | 0.98 | 208 |
| Grape___Esca_(Black_Measles) | 1.00 | 0.98 | 0.99 | 232 |
| Grape___Leaf_blight | 1.00 | 0.99 | 1.00 | 160 |
| Grape___healthy | 0.99 | 0.99 | 0.99 | 160 |
| | | | | |
| accuracy | | | 0.99 | 1696 |
| macro avg | 0.99 | 0.99 | 0.99 | 1696 |
| weighted avg | 0.99 | 0.99 | 0.99 | 1696 |

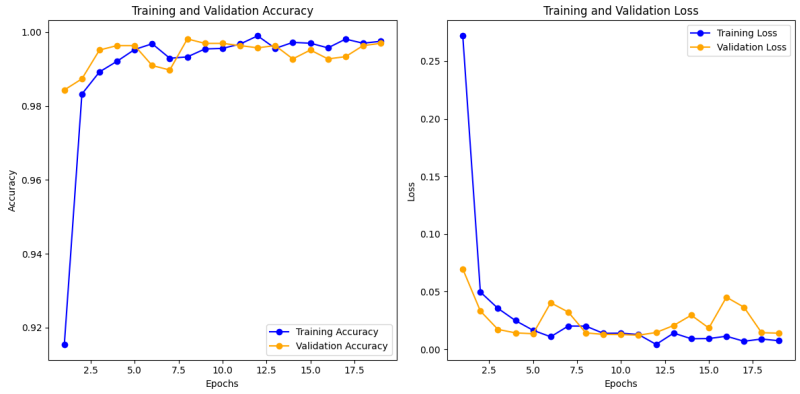



## Test Images Classification Performance



## Keras sequential model with transferring learning from the pre-trained EfficientNet B0 results

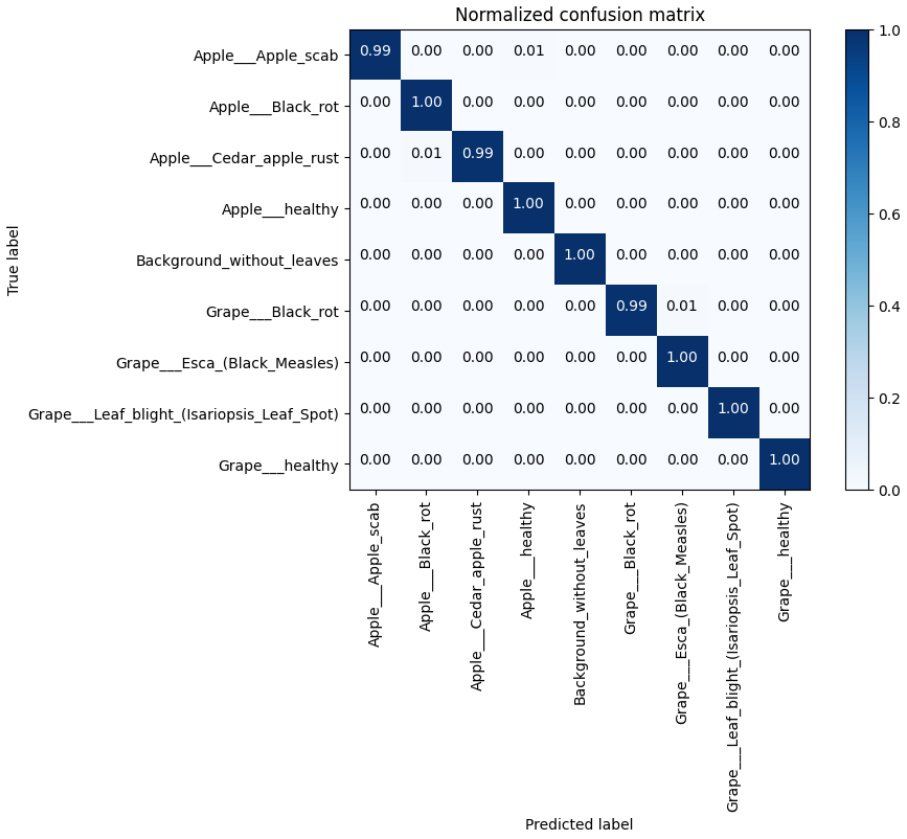
### Train validation accuracy and training validation loss before Fine-tuning



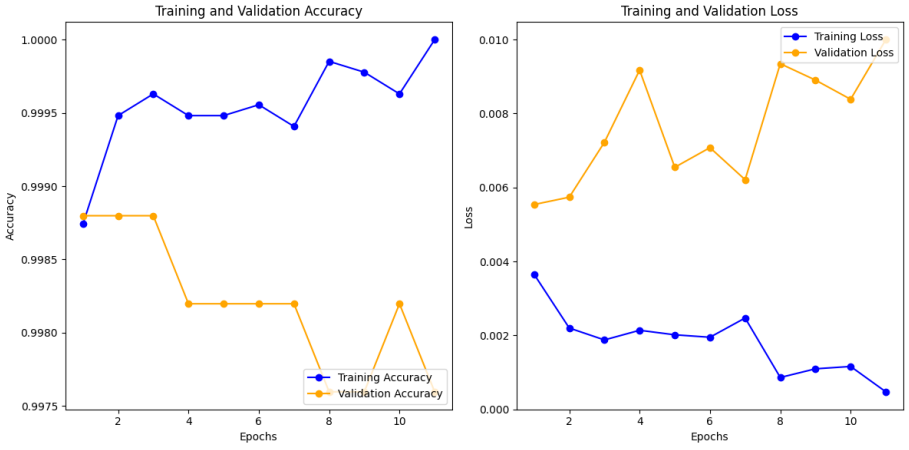

### Classification Performance Metrics before Fine-tuning

| Class Name | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Apple___Apple_scab | 1.00 | 0.99 | 1.00 | 198 |
| Apple___Black_rot | 0.99 | 1.00 | 0.99 | 199 |
| Apple___Cedar_apple_rust | 1.00 | 0.99 | 1.00 | 212 |
| Apple___healthy | 1.00 | 1.00 | 1.00 | 225 |
| Background_without_leaves | 1.00 | 1.00 | 1.00 | 122 |
| Grape___Black_rot | 1.00 | 0.99 | 0.99 | 200 |
| Grape___Esca_(Black_Measles) | 0.99 | 1.00 | 1.00 | 210 |
| Grape___Leaf_blight | 1.00 | 1.00 | 1.00 | 155 |
| Grape___healthy | 1.00 | 1.00 | 1.00 | 207 |
| | | | | |
| accuracy | | | 1.00 | 1728 |
| macro avg | 1.00 | 1.00 | 1.00 | 1728 |
| weighted avg | 1.00 | 1.00 | 1.00 | 1728 |

## Confusion Matrix before Fine-tuning



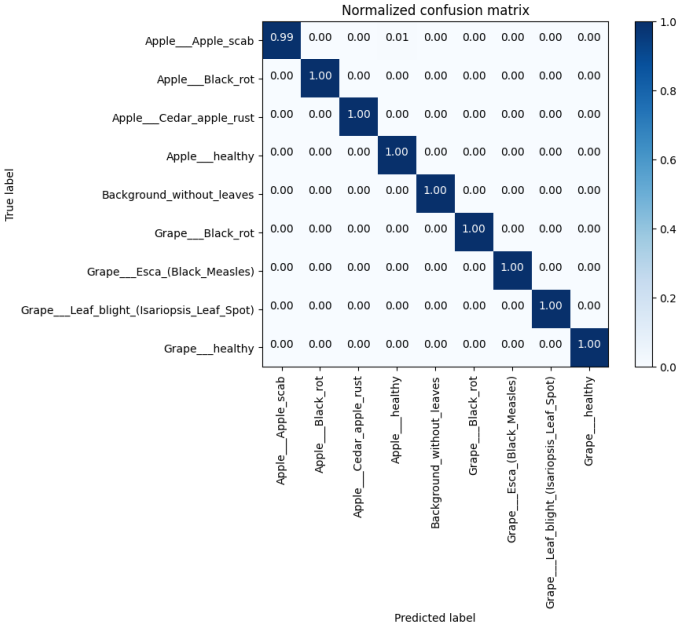

## Train validation accuracy and training validation loss after Fine-tuning



### Classification Performance Metrics after Fine-tuning

| Class Name | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Apple___Apple_scab | 1.00 | 0.99 | 1.00 | 198 |
| Apple___Black_rot | 1.00 | 1.00 | 1.00 | 199 |
| Apple___Cedar_apple_rust | 1.00 | 1.00 | 1.00 | 212 |
| Apple___healthy | 1.00 | 1.00 | 1.00 | 225 |
| Background_without_leaves | 1.00 | 1.00 | 1.00 | 122 |
| Grape___Black_rot | 1.00 | 1.00 | 1.00 | 200 |
| Grape___Esca_(Black_Measles) | 1.00 | 1.00 | 1.00 | 210 |
| Grape___Leaf_blight | 1.00 | 1.00 | 1.00 | 155 |
| Grape___healthy | 1.00 | 1.00 | 1.00 | 207 |
| | | | | |
| accuracy | | | 1.00 | 1728 |
| macro avg | 1.00 | 1.00 | 1.00 | 1728 |
| weighted avg | 1.00 | 1.00 | 1.00 | 1728 |

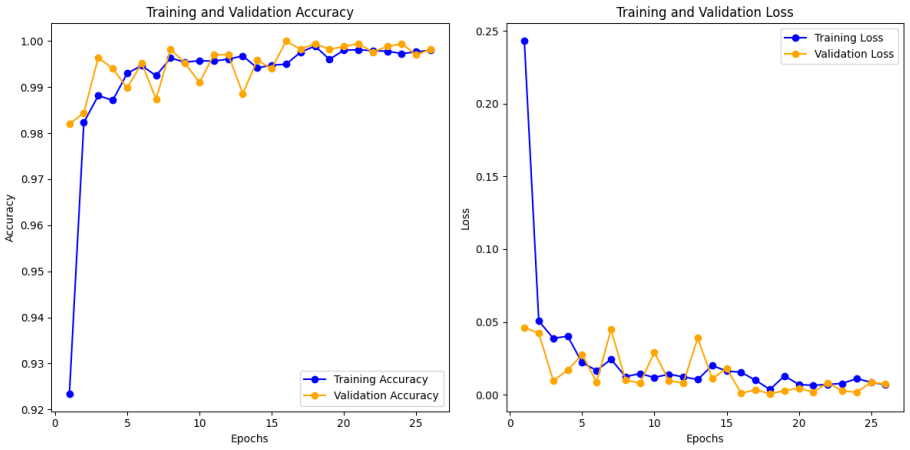
### Confusion Matrix after Fine-tuning



**Test Image Classification Performance**

## Keras sequential model with transferring learning from the pre-trained EfficientNet B1 results

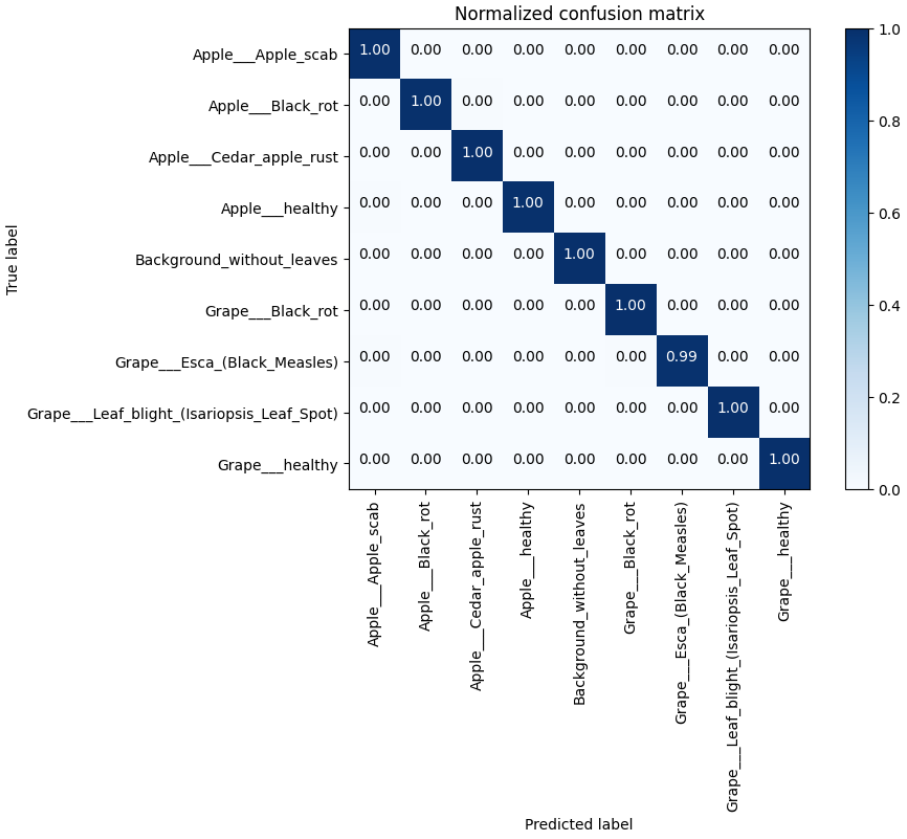
### Train validation accuracy and training validation loss before Fine-tuning



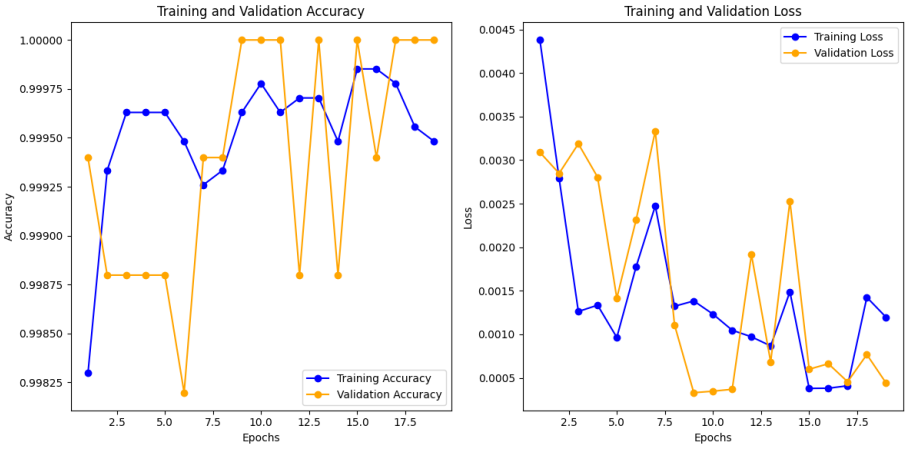

### Classification Performance Metrics before Fine-tuning

| Class Name | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Apple___Apple_scab | 0.99 | 1.00 | 0.99 | 192 |
| Apple___Black_rot | 1.00 | 1.00 | 1.00 | 225 |
| Apple___Cedar_apple_rust | 0.99 | 1.00 | 1.00 | 193 |
| Apple___healthy | 1.00 | 1.00 | 1.00 | 213 |
| Background_without_leaves | 1.00 | 1.00 | 1.00 | 111 |
| Grape___Black_rot | 0.99 | 1.00 | 1.00 | 188 |
| Grape___Esca_(Black_Measles) | 1.00 | 0.99 | 1.00 | 246 |
| Grape___Leaf_blight | 1.00 | 1.00 | 1.00 | 172 |
| Grape___healthy | 1.00 | 1.00 | 1.00 | 188 |
| | | | | |
| accuracy | | | 1.00 | 1728 |
| macro avg | 1.00 | 1.00 | 1.00 | 1728 |
| weighted avg | 1.00 | 1.00 | 1.00 | 1728 |

## Confusion Matrix before Fine-tuning




## Train validation accuracy and training validation loss after Fine-tuning

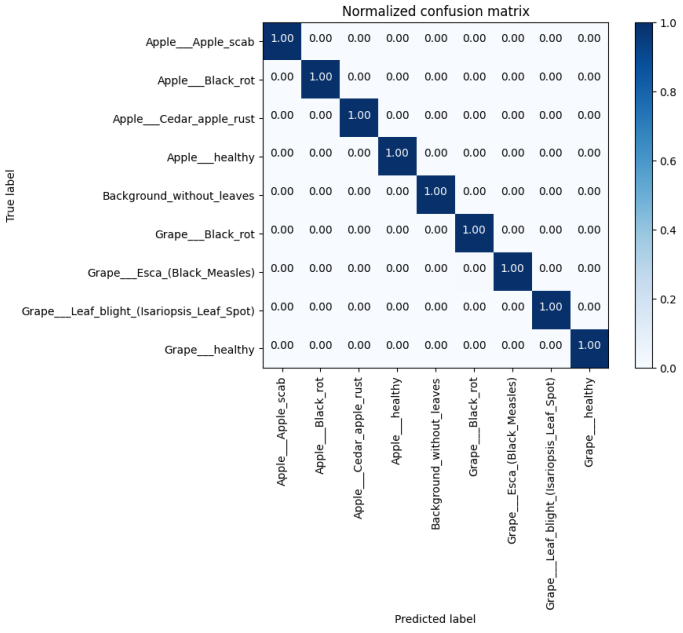


### Classification Performance Metrics after Fine-tuning

| Class Name | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Apple___Apple_scab | 1.00 | 1.00 | 1.00 | 192 |
| Apple___Black_rot | 1.00 | 1.00 | 1.00 | 225 |
| Apple___Cedar_apple_rust | 1.00 | 1.00 | 1.00 | 193 |
| Apple___healthy | 1.00 | 1.00 | 1.00 | 213 |
| Background_without_leaves | 1.00 | 1.00 | 1.00 | 111 |
| Grape___Black_rot | 0.99 | 1.00 | 1.00 | 188 |
| Grape___Esca_(Black_Measles) | 1.00 | 1.00 | 1.00 | 246 |
| Grape___Leaf_blight | 1.00 | 1.00 | 1.00 | 172 |
| Grape___healthy | 1.00 | 1.00 | 1.00 | 188 |
| | | | | |
| accuracy | | | 1.00 | 1728 |
| macro avg | 1.00 | 1.00 | 1.00 | 1728 |
| weighted avg | 1.00 | 1.00 | 1.00 | 1728 |

Table 7: Classification Performance Metrics

### Confusion Matrix after Fine-tuning



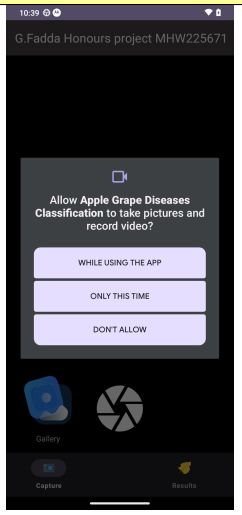
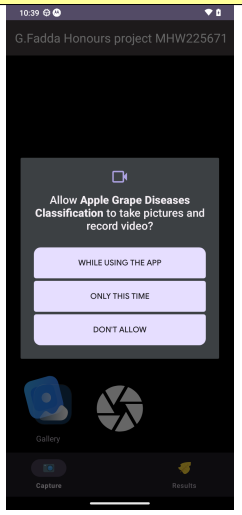
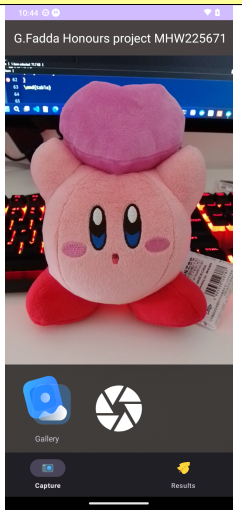
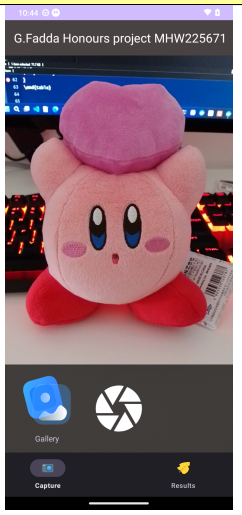
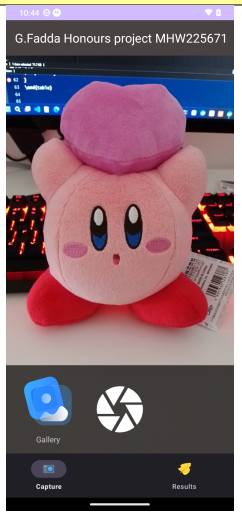
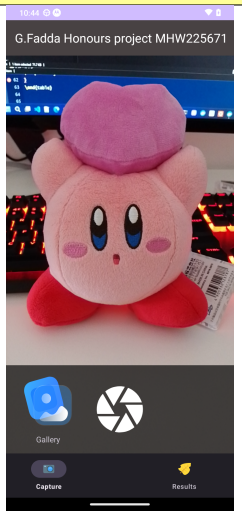
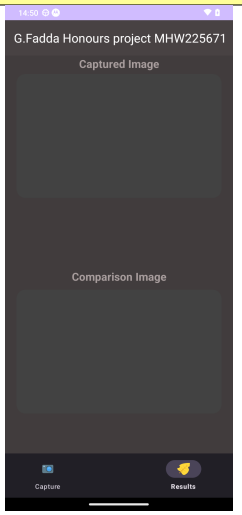
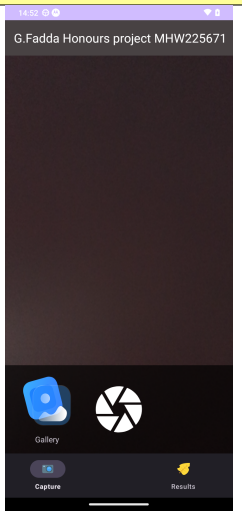
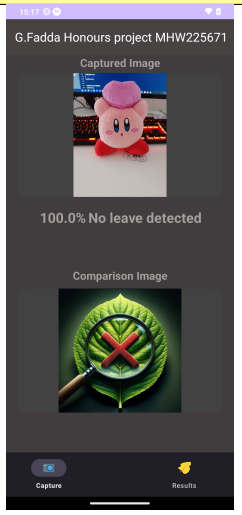
**Test Image Classification Performance**

# B Appendix - Android app test document

| List of test cases | | | |
|---|---|---|---|
| **Test n.** | **Purpose of the test** | | |
| 1 | Testing the view of the "permission dialog." for the camera's Device access. | | |
| 3 | Testing the "Don't Allow" button in the "permission dialog." | | |
| 4 | Testing the "Only this time" button in the "permission dialog." | | |
| 5 | Testing the "While Using the app" button in the "permission dialog." | | |
| 6 | Testing all elements on the main page are visualised. | | |
| 7 | Testing the menu bar elements. | | |
| 8 | Testing the access to the photo gallery. | | |
| 9 | Testing the image classification capturing an image. | | |
| 10 | Testing the image classification selecting an image from the gallery. | | |
| 11 | Testing the correct visualisation of the image classification page result. | | |
| **date** | 01/04/24 | **System** | Android 13 go edition on Motorola Moto E 13 |

| Test n. | Test cases | Pre-Conditions | Expected result | Post-Condition | PASS/Fail |
|---|---|---|---|---|---|
| 1 | Verify that the app runs. | On the mobile, select the app icon to run the app. | The app will be open, and the main page will be displayed. | The app is opened, and the main page is displayed. | **PASS** Link to Image |
| 2 | Verify that the "permission dialog." for the camera's Device access is visualised. | On the mobile, select the app icon to run the app. | The app will be open, and the main page will be displayed with the "permission dialog.". | The app will be open, and the main page will be displayed with the "permission dialog.". | **PASS** Link to Image |
| 3 | Verify that the camera on the main mage is not open if the "Don't Allow" button is selected in the "permission dialog.". | On the mobile, select the app icon to run the app the first time and press the the "Don't Allow" button. | Access to the camera will be denied, and the main page's camera area will be black. | Access to the camera is denied, and the main page's camera area is black.". | **PASS** Link to Image |
| 4 | Verify that the camera on the main mage is opened if the "Only this time" button is selected in the "permission dialog.". | On the mobile, select the app icon to run the app the first time and press the the "Only this time" button. | Access to the camera will be allowed, and the camera will be opened on the main page. | Access to the camera is allowed, and the camera is opened on the main page. | **PASS** Link to Image |
| 5 | Verify that the camera on the main mage is opened if the "While using the app" button is selected in the "permission dialog.". | On the mobile, select the app icon to run the app the first time and press the the "While using the app" button. | Access to the camera will be allowed, and the camera will be opened on the main page. | Access to the camera is allowed, and the camera is opened on the main page. | **PASS** Link to Image |

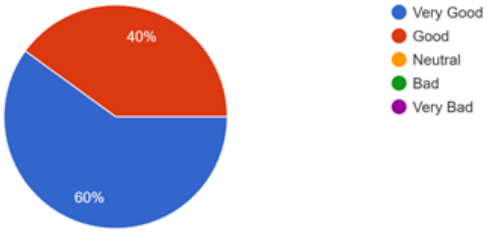
| Test n. | Test cases | Pre-Conditions | Expected result | Post-Condition | Pass/Fail |
|---|---|---|---|---|---|
| 6 | Verify that the main page is visualised with all components as expected. | On the mobile, select the app icon. | The main page displayed will contain the menu bar consisting of the buttons **'Capture'** and **'Results'**, and the camera menu will contain **Gallery** and **Capture images** buttons. | The main page displayed contains the menu bar consisting of the buttons **'Capture'** and **'Results'**, and the camera menu contains **Gallery** and **Capture images** buttons. | **Pass**Link to Image |
| 7 | Verify that the buttons in the menu-bar work. | Select the button "results" to display the Results page and button "capture" to display the main page. | Selecting the button "results" the results page will be opened, while selecting the button "capture" the main page will be opened. | Selecting the button "results" the results page is opened, while selecting the button "capture" the main page is opened. | **Pass** Link to Image |
| 8 | Verify the access to the image gallery. | from the main page select the button "Gallery" to open the device image gallery. | Selecting the button "Gallery", the device image gallery will be opened. | Selecting the button "Gallery", the device image gallery is opened. | **Pass** Link to Image |
| 9 | Verify the image classification capturing an image . | from the main page select the button Capture to capture an image. | The app will open the "results" page showing the classification result. | The app will open the "results" page showing the classification result. | **Pass** Link to Image |
| 10 | Verify the image classification selecting an image from the gallery. | from the main page select the button "Gallery" to open the device image gallery and select an image. | The app will open the "results" page showing the classification result. | The app will open the "results" page showing the classification result. | **Pass** Link to Image |
| 11 | Verify the correct visualisation of all elements of the Results page. | Select the button "results" to display the "Results" page. | The "result" page will display, starting from the top, the image classified, the name of the diseases, the confidence of the classification, and an example image. | The "result" page displays, starting from the top, the image classified, the name of the diseases, the confidence of the classification, and an example image. | **Pass** Link to Image |

| Test n.1 | Test n.2 | Test n.3 | Test n.4 |
|---|---|---|---|
|  |  |  |  |
| **Test n.5** | **Test n.6** | **Test n.7** | **Test n.7** |
|  |  |  |  |
| **Test n.8** | **Test n.9** | **Test n.10** | **Test n.11** |
|  |  |  |  |

# C Appendix - Android app User Feedback

Application usability. Provide your feedback about the capacity of the application to allow to the user to accomplish their goals efficiently and satisfactorily.

5 responses




Explain your choice.

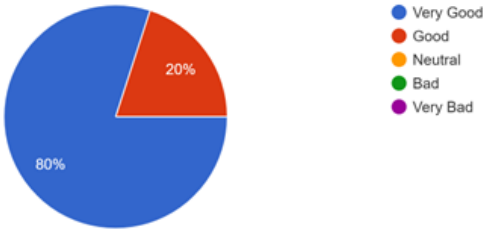
2 responses

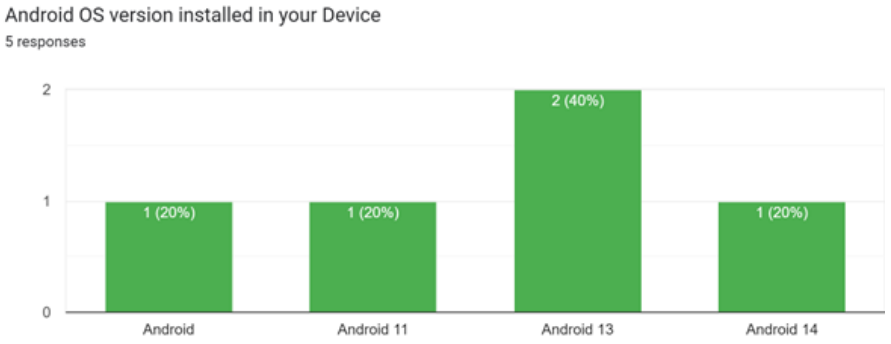
the application is very user friendly and intuitive

I have tried the app and it is efficient

Application Compatibility. Provide your feedback about the capacity of the application to operate and interact effectively in you device.

5 responses

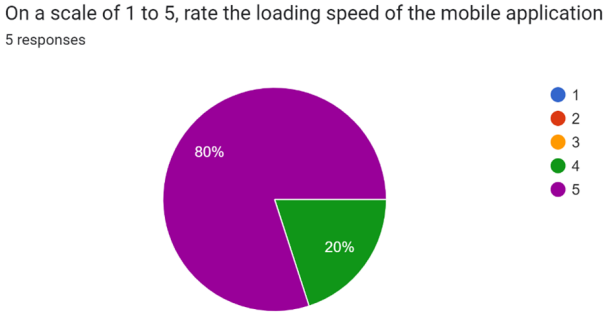





Provide the Brand and model of your device

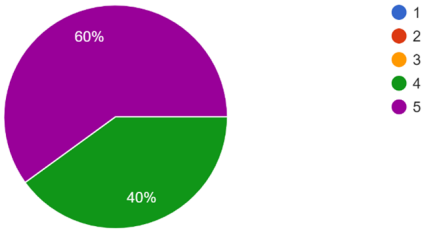
5 responses

| |
|---|
| Honor Magic 5 Pro |
| Google pixel |
| Moto e13 |
| Samsung Galaxy s22 |
| Motorola moto g 8 Power |



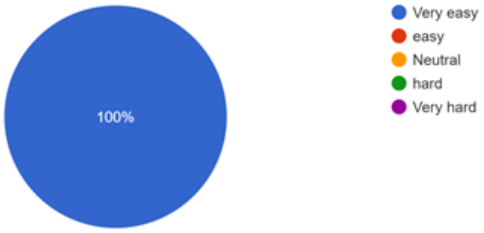
On a scale of 1 to 5, rate the accuracy of the image result.

5 responses




Application UI learnability. How easy is it to pick up, understand and use the application?

5 responses




Explain your choice.

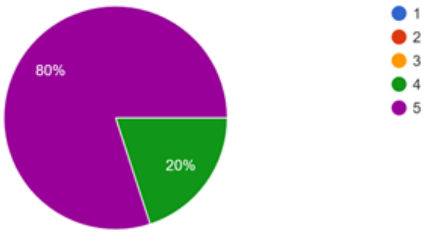
2 responses

the app is easy to learn as it is well structured and simple

Open the app and if camera detect leaves you will get result

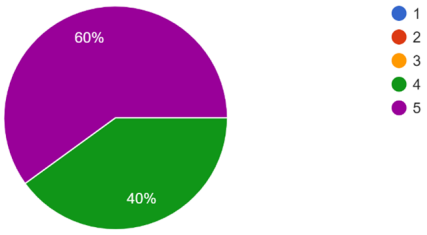
On a scale of 1 to 5, rate the speed of the mobile application to give a image result.

5 responses



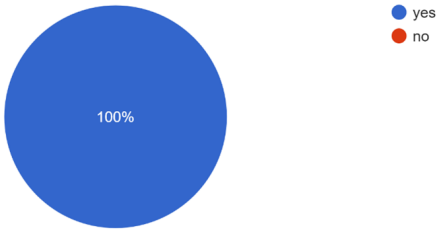
On a scale of 1 to 5, rate the accuracy of the image result.

5 responses




Would you suggest a student or early career farming entrepreneurs use this app for educational purposes?

5 responses




Explain your choice.

2 responses

I believe that the future of every agricultural entrepreneur will be based on and will be helped by this type of app, and those who know how to use and exploit them will have a significant advantage

Anyone who interested about trees or maybe flowers will interest about this app as well as it is very useful.

Anything else you would like to share about the mobile app?

2 responses

In conclusion, I believe that the app is well studied and developed, has no bugs and is very easy to use, so I believe that its further development can really become a concrete help for the agricultural entrepreneur

maybe providing more languages

# D Appendix - Code used to develop the project

All code developed for the project can be found on Microsoft OneDive at the following link:

Giacomo_Fadda_Dissertation_Project_S2219926