Universidade Federal de Pernambuco
Centro de Informática

Graduação em Engenharia da Computação

Hydra: Crop Disease Classification with Multi-task Learning

Igor de Moura Philippini

Trabalho de Graduação

Recife
27 de Outubro de 2022

Universidade Federal de Pernambuco
Centro de Informática

Igor de Moura Philippini

Hydra: Crop Disease Classification with Multi-task Learning

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Prof. Stefan Michael Blawid*

Recife
27 de Outubro de 2022

Acknowledgements

Looking back at everything I've lived so far, for sure I wouldn't be where I am today without all the support of my family, who did whatever was in their reach, and often even when it wasn't, to provide me all I needed in order to succeed in life.

I'd like to give a thanks in particular to professor Edna Barros, who always motivated her students to always try their best in order to achieve what they wanted, followed by professor Stefan Blawid, who was always patient and showed lots of interest in what we were building, even if it were outside of his area of knowledge.

Also all of my friends who I've made along the way during this undergraduate course, whom I faced many hardships together and managed to plow through this degree, specially through the times were we discussed about giving up after every failed test, only to motivated each other back again to keep moving forward. In no specific order, my many thanks to Ladson, Augusto, Laís, Rodrigo, Mari, Gabi, Thiago, Pedro, Matheus, Nathalia, and many others who I'm sure I'm forgetting (sorry for that in advance!). Also worth noting my friends who helped me even before I got into university, Nivaldo, Yann and Edgar.

I would also like to thank my girlfriend Sara, who motivated me, helped me however she could and was always by my side helping me to grow even more as a person during those last semesters.

Last but not least, a special thanks to this university, where I spent a great part of my life and learned valuable lessons, allowing me to get in touch with all kinds of people that helped me understand to see different points of views and me more empathetic towards the unique background each person has.

One thing I've learned: you can know anything, it's all there, you just have to find it.

—NEIL GAIMAN

Resumo

Pragas e doenças são um dos maiores problemas da agricultura, representando uma perda econômica significante. Pesticidas são comumente mal utilizados, acarretando em um controle de pragas ineficiente. Para pequenos agricultores, a falta de informação é ainda mais danosa dado ao parco suporte oferecido por agências governamentais. Atualmente existem vários sistemas para detecção de doenças em plantas através de imagens, ajudando especialistas na área de fitopatologia a prover um diagnóstico para as plantações. Tais sistemas fazem uso de visão computacional e aprendizagem de máquina para classificar as doenças através de fotos das folhas e frutos. No entanto, prover um diagnóstico adequado envolve uma série de passos intermediários, como detecção do agente causador da doença e reconhecimento de sintomas. Nesse trabalho é proposta a construção de um sistema com as seguintes partes: um único modelo capaz de lidar com diferentes tarefas a partir de uma única entrada, sendo elas a detecção se está doente ou não, indicação de sintomas e detecção se a doença acometida é um fungo; um aplicativo móvel que fará uso das saídas do nosso modelo em produção.

Palavras-chave: Visão computacional, multi-task learning, doenças em folhas.

Abstract

Pests and diseases are one of the biggest problems in agriculture, responsible for significant economic losses. Pesticides are frequently misused, resulting in inefficient pest control. For small farmers, the lack of information is even more damaging due to the little support received by government agencies. Many systems to detect plant diseases through images exist nowadays, aiding phytopathology specialists in diagnosing crops. Those systems use computer vision and machine learning to classify diseases through photos of leaves and fruits. However, diagnosing properly involves many intermediary steps, such as disease agent recognition and symptom classification. In this work, we propose to build a single model capable of dealing with multiple tasks from a single input, being those the indication if a leaf is healthy or not, which symptoms it has, and if the disease is caused by fungi, along with a pipeline to make use of its outputs through a mobile app.

Keywords: Computer vision, multi-task learning, leaf disease.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Brazil is one of the few countries capable of reconciling its food production with conserving its unique biodiversity. However, it also has one of the biggest pesticide markets in the world, as shown by Rigotto, Vasconcelos, and Rocha 2014, not to mention the increase in sales of illegal agrochemicals seen in the last few years. Plagues and plant diseases are the culprits of huge economic deficits in the country, being responsible for losses equivalent to 43% of its annual production in 2016, as seen in IBGE 2017. Such significant losses are avoidable when caused by the misuse of phytosanitary products, or even lack thereof, meant to control those diseases.

Nowadays, small farmers are responsible for over 70% of the domestic food production in Brazil, IBGE 2017. In this scenario, they also act as inspectors for their crops. However, they usually lack the proper training and equipment to diagnose any possible disease. Some projects try to aid those farmers with tooling that provides an easy and fast way to perform such diagnosis. Machine learning tools detect disease symptoms within a plant's leaves, Jez et al. 2021a, sending this information to specialists through the internet, enabling timely disease identification before it can incur any biological and financial damage. Getting a proper diagnosis for plant diseases is crucial to any disease contention strategy, targeting the actual pathogen and avoiding any excess use of agrochemicals without professional supervision.

This research continues one of such projects, initially described in Barros et al. 2021. A scalable model based on the idea of multi-task learning (MTL), Ruder 2017, augments an end-to-end system capable of identifying the health status of a plant through a single picture of its leaves. The MTL approach enables one to perform more than one task with a single machine learning model and a single input for the said model. In the present context, a MTL model allows simultaneously to classify the plant's health status - as proposed by Barros et al. 2021 - but also to categorize any signs of macro-symptoms on the leaves required for disease recognition or even identify such diseases in a specific manner. Even more complex modularization can be applied, and additional tasks easily added without interfering with the existing ones, Crawshaw 2020, albeit beyond the scope of the present work and left for future research.

1.1 Objectives

This work aims to create a system capable of detecting and identifying plant diseases and the causing pathogens, using a multi-task learning (MTL) based machine learning model with limited scope but extensibility, which we will refer to as Hydra from now on.

1.2 Outline

The present work is organized as following:

- Chapter 2 - Theoretical Foundation: The chapter exposes the background knowledge in this area that serves as the basis for this work.
- Chapter 3 - Related Works: The chapter discusses works that serve as the baseline for the current one.
- Chapter 4 - Proposed Idea: The chapter gives an overview of what's the intended result from this work.
- Chapter 5 - Methodology: The chapter presents the actual steps that were taken during the development of this project.
- Chapter 6 - Results: The obtained results from the final model.
- Chapter 7 - Conclusion and future works: Closing of this work along with possible future improvements.

CHAPTER 2

Theoretical Foundation

This chapter describes some basic concepts, building blocks and theoretical foundations required for understanding the current contribution. Moreover, the chapter reviews selected works that served as an inspiration to the development shown in the following chapters.

2.1 Machine learning

Machine Learning (ML) is one of the many topics included in discussions about Artificial Intelligence (AI). ML is the subarea that tries to achieve results without being taught beforehand what should be done. Instead, predictions are obtained based on given examples and trying to attain similar results by trial and error, as discussed by Solomonoff 1957.

Using such techniques allows society to solve novel problems without spending much time pursuing mathematical proofs or complex formulas at the cost of having non-deterministic results due to the black-box nature of neural network-based models. The trade-off is deemed acceptable for most practical applications, ranging from recommendation systems to content creation, data forecasting, and speech and image recognition, the latter being the focus of this work.

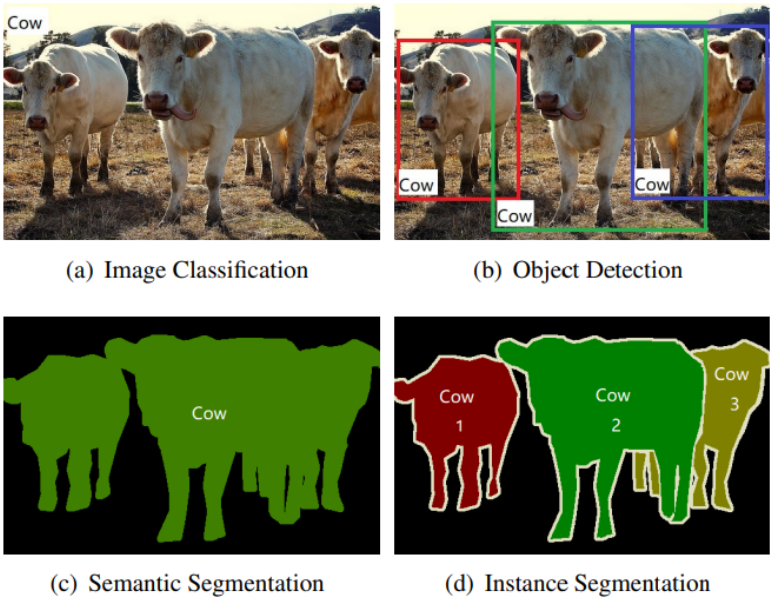
2.1.1 Neurons and Artificial Neural Networks

The basis of any modern model able to learn and achieve results without being explicitly programmed to do so is related to the neuron found in the human brain. McCulloch and Pitts 1943 showed how a simple math formula based on inputs, multiplied by arbitrary weights and then summed together before being compared through an activation function, can describe a human neuron. The model was called a Perceptron later on by Rosenblatt 1957, who also showed how multiple Perceptrons can work in tandem and in parallel.

However, a single-layer Perceptron network was not sophisticated enough to solve even some trivial problems, such as a boolean XOR. Rosenblatt himself demonstrated that layers of Perceptrons built on top of one another are required to solve such problems. Multi-layer perceptrons (MLP) networks became feasible only after the definition of backpropagation was formulated by Rumelhart, Hinton, and Williams 1986, based on the work from Linnainmaa 1970. Backpropagation allows MLPs to be trained based on labeled examples through trial and error in an autonomous way, giving birth to the modern concept of Artificial Neural Networks (ANNs).

2.2 Computer Vision

One of the areas worked by researchers in the area of artificial intelligence is the one related to how the human brain can see and interpret its surroundings and how we can mimic those abilities with a machine, as discussed by T. Huang 1996. As it turns out, machine vision requires many different capacities such as, but not limited to, Edge detection, Semantic segmentation, Instance segmentation, Image classification, and Object detection, as seen in Figure 2.1.



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

Figure 2.1: Different tasks found withing the computer vision domain. (a) "Image Classification" only needs to assign categorical class labels to the image; (b) "Object detection" not only predict categorical labels but also localize each object instance via bounding boxes; (c) "Semantic segmentation" aims to predict categorical labels for each pixel, without differentiating object instances; (d) "Instance segmentation", a special setting of object detection, differentiates different object instances by pixel-level segmentation masks. Source: Wu, Sahoo, and Hoi 2020

In this work, we will focus on the segmentation and classification aspects of computer vision, which have seen many advances in the last years, as shown in Wu, Sahoo, and Hoi 2020, and which we will discuss below.

2.2.1 Image segmentation

Image segmentation is the process of partitioning an image into meaningful regions. Typically, image segmentation enables to locate objects and boundaries in images. More precisely, image segmentation assigns a label to every pixel in an image such that pixels with the same label share certain characteristics.

There are two main types of image segmentation:

- Semantic segmentation: This type of segmentation classifies each pixel in an image into a class. For example, semantic segmentation can classify each pixel as belonging to a particular object (e.g. person, car, tree) or background.
- Instance segmentation: This type of segmentation not only classifies each pixel but also differentiates between different instances of the same class. For example, instance segmentation can segment distinct people in an image.

2.2.2 Image classification

Image classification is the process of assigning a class label to an image. The class label can be any one of a number of predefined classes, or it can be a user-defined class. Image classification is a supervised learning problem where a training set of images with known class labels is used to train a classifier. The classifier can then label new images.

Various applications employ image classification, such as object detection, facial recognition, and image retrieval.

2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of artificial neural network that is particularly well suited to image classification tasks. They work by extracting features from images and then using those features to classify the images.

CNNs typically have several layers, with each layer extracting different features from the images. The first layer is usually convolutional and extracts low-level features such as edges, corners, textures, and curves. The second layer is typically a pooling layer, which downsamples the image in a non-linear way to reduce the amount of data and parameters that need to be processed by the network, thus reducing the chances of overfitting. The final layer is a fully-connected layer, which performs the actual classification. A representation of this architecture can be seen in Figure 2.2.

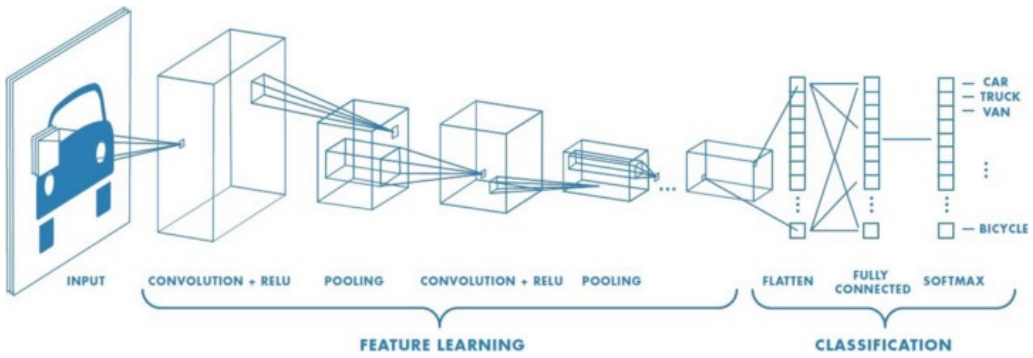


Figure 2.2: Representation of a CNN architecture. Source: Saha 2018

CNNs are very effective at image classification, outperforming traditional neural networks and other machine learning methods. In part, the better performance is due to their ability to

extract features from images hierarchically, making them well-suited to the task.

The first CNN architecture that got widespread attention was the LeNet-5 architecture proposed by LeCun et al. 1998. It consisted of 7 layers — two convolutional layers, three fully-connected layers, and two subsampling layers. The LeNet-5 architecture was used to recognize handwritten digits from the MNIST dataset.

In 2012, Krizhevsky, Sutskever, and Hinton 2017 proposed a deep CNN architecture called AlexNet. The network consisted of 8 layers — 5 convolutional layers, two fully-connected layers, and one subsampling layer. AlexNet was used to recognize objects from the ImageNet Russakovsky et al. 2015 dataset, being the first CNN architecture to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

In 2016, He, Zhang, et al. 2016b introduced a Residual Network (ResNet) architecture. It was the first CNN architecture to use skip connections or shortcuts to jump over some layers. The idea behind skip connections is to allow the gradient to flow more easily back to earlier layers during training. ResNet was used to recognize objects from the ImageNet dataset and won first place in the ILSVRC.

2.4 Multi-task Learning

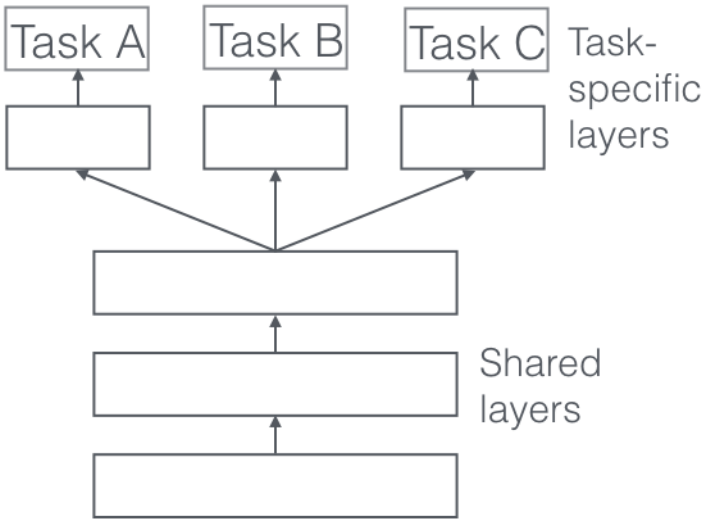
Multi-task learning (MTL) is an ML methodology resulting in a model that simultaneously performs multiple tasks while sharing low-level features or knowledge, as discussed by Caruana 1997. MTL differs from single-task learning, where a model only performs a single task. Multi-task learning improves the capacity of models to generalize, as the model can learn to transfer knowledge between tasks. Several approaches address the challenge to design models that can effectively exploit the information shared between tasks while avoiding negative transfer, including hard and soft parameter sharing, as shown in Figure 2.3, task clustering, and low-rank parameter regularization.

Hard and soft parameter sharing is also called "shared trunk" and "cross-talk", respectively, Crawshaw 2020. A hard-sharing model has many common layers before task-specific not-shared layers are employed. In soft sharing, on the other hand, the task-specific layers are connected, keeping each task somewhat independent but capable of eavesdropping on their neighbors. Crawshaw 2020 also shows how modern models use this technique in different areas like transformer-based models and reinforcement learning.

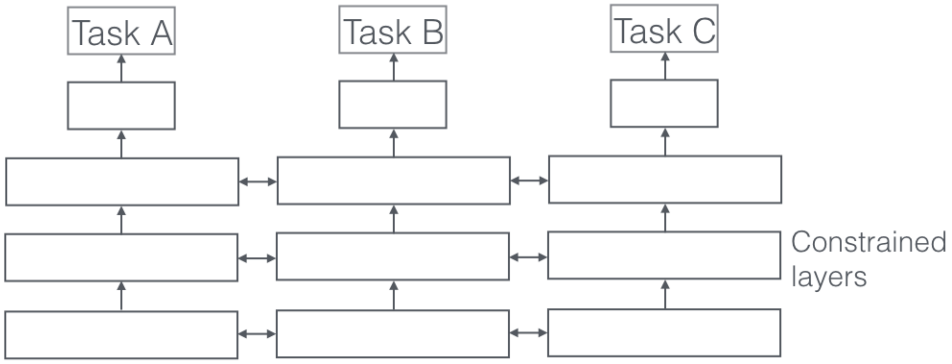
Moreover, Ruder 2017 discusses how such models can improve upon their single-task counterparts, increasing the usefulness of the MTL technique. The performance boost results from the shared underlying layers, empowering each task to reuse features acquired from another task. In addition, Baxter 1997 has shown that the risk of overfitting a MTL model is an order of N smaller than a single-task model, with N being the number of tasks in the model. The underlying shared model has to be tuned, resulting in end values for the weights between the ideal ones for the multiple tasks, which reduces overfitting.

Another benefit of such architecture is that one can easily add new tasks on demand, making the model modular and extensible. In contrast, most traditional machine learning models are much harder to modify once trained, often requiring the sacrifice of the previous task used by the model when training for a new one.

However, there are some downsides with an MTL model, as shown by Karpathy n.d. in an actual real-life production environment. One of the hardest parts is how to limit the sharing of parameters between tasks and how much of the model should be frozen for each specific task, especially when working with larger teams. The parameters of the shared trunk are often disputed among sub-teams responsible for different tasks. Versioning such models is also an issue since, from a single starting point, multiple training sessions for each task may end up branching the results achieved.



(a) Example of MTL model with hard parameter sharing.



(b) Example of MTL model with soft parameter sharing.

Figure 2.3: Different ways to share parameters between tasks in a MTL model. Source: Ruder 2017.

CHAPTER 3

Related Works

3.1 Supervised Training of a Simple Digital Assistant for a Free Crop Clinic

Barros et al. 2021 proposed a system that allows farmers to get in touch with other farmers and phytopathology experts to discuss and diagnose any issues within their crops. To this extent, a mobile app was built, allowing the farmer to share pictures of the suspected diseased leaf and receive feedback on the possibility of showing disease symptoms. Simultaneously, all provided information and photos are also sent to an expert, making it possible for both parties to get in touch to try to diagnose the issue. The work focuses on the diagnosis of grape crops. Figure 3.1 shows the system's pipeline.

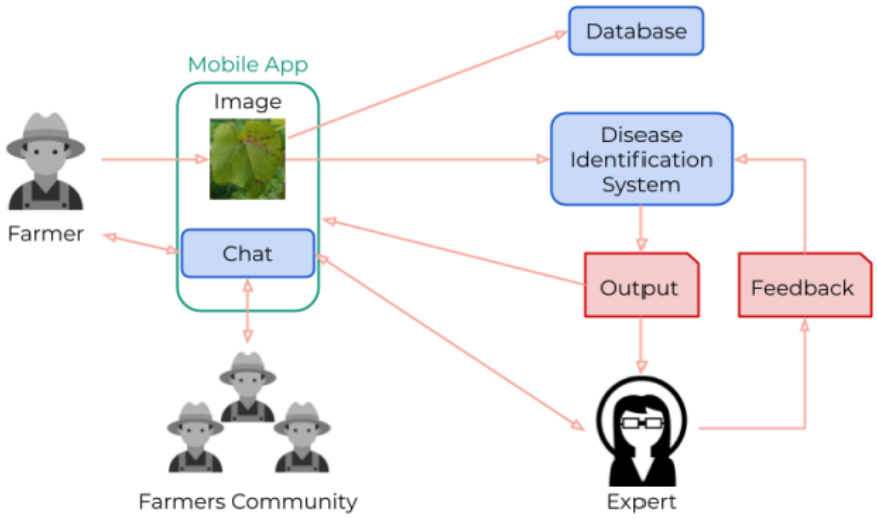


Figure 3.1: General architecture of the proposed system by Barros et al. 2021 composed by a mobile app and a digital assistant for a crop clinic

A machine-learning model based on pre-trained ResNet50v2 weights enabled the app to provide such feedback. The training dataset for the model comprised 3289 images of grape leaves, created and labeled by phytopathology experts in two classes: 1302 images showing "Symptoms" and the remaining 1987 showing "No symptoms". The images were then cropped and segmented to augment the dataset. Figure 3.2 shows the pre-processing and classification pipeline. The model achieved promising results, obtaining a recall of over 95% for detecting a

disease in a leaf image.

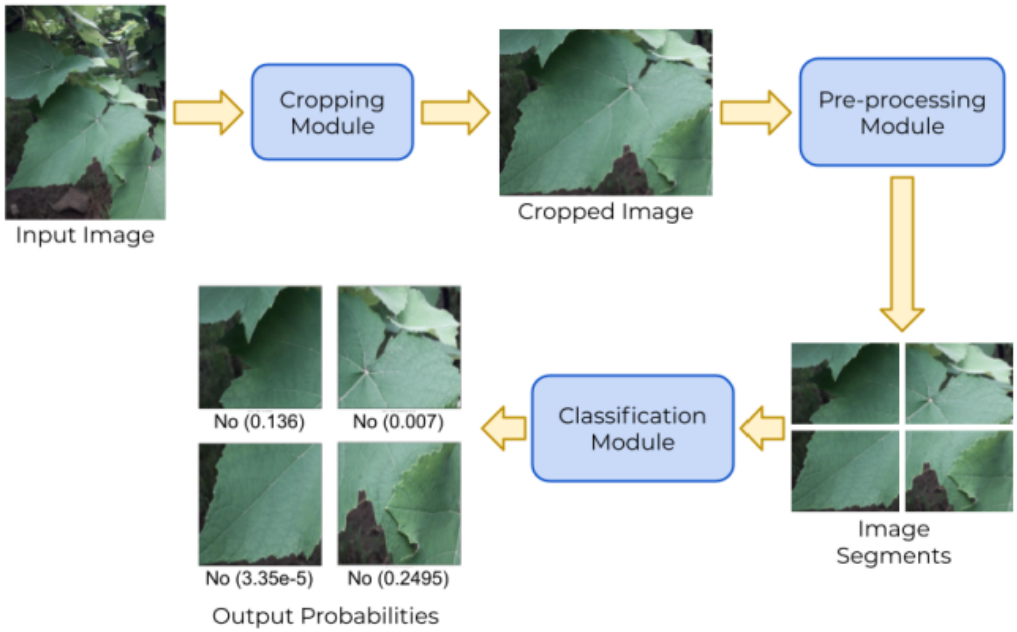


Figure 3.2: Task pipeline proposed by Barros et al. 2021

The current work employs the same app but proposes a different model, using the model created by Barros et al. 2021 as a basis. The proposed model adds new outputs besides the original one that classified input pictures for showing a diseased or healthy leaf. Model training includes several data sets in addition to the labeled dataset compiled by Barros et al. 2021. Moreover, the pre-processing of pictures employs a different pipeline, described in Chapter 4.

3.2 Using Deep Learning for Image-based Plant Disease Detection

Exploring the dataset created by David. P. Hughes and Salathe 2015, which has a total of 54306 leaf images containing 14 different crops and 26 different classes (including healthy pictures), Mohanty, David P Hughes, and Salathé 2016 tested two CNN models, AlexNet and GoogLeNet, both mentioned in Section 2.3, comparing the performance between fine-tuning the existing models trained on ImageNet and a model trained from scratch, along with different models for each variation of the original dataset (Color, Grayscale and Segmented), obtaining similar results for all of their tests.

However, as shown by Barros et al. 2021, since the PlantVillage dataset only has pictures of leaves taken in a controlled environment, it does not capture crop field conditions. Consequently, the performance of the models trained solely on that dataset is insufficient for real-life scenarios. Nonetheless, the work by Mohanty, David P Hughes, and Salathé 2016 has paved the way for using CNNs to classify crop diseases from leaf pictures, impulsing the area to search for better techniques and data. In addition, the assembled dataset serves as an excellent baseline

for training models with various objectives. The dataset clearly labels the actual disease agents and includes numerous plant species, hardly found in other public datasets.

CHAPTER 4

Proposed idea

This works aims to provide a system capable of recognizing diseases from a single picture of a plant's leaf taken with a smartphone's mobile camera. This chapter gives an overview of the required innards enabling such a system. The reader gains a better insight into the main steps in the proposed pipeline and how they interact before diving into the details of each step in the next chapter.

4.1 System Overview

The overall system is split into what we will call "modules", represented as little blocks that take some input, for example, an image or other form of data, and returns another kind of processed data. We can even consider the entire project as a macro-module that takes an image of a leaf as input and returns information related to its health. The system's output may include the health status (diseased or not), a classification of any possible macro-symptoms, the disease that causes such symptoms, and even the pathogen for said disease. Figure 4.1 shows a simplified pipeline of the system.

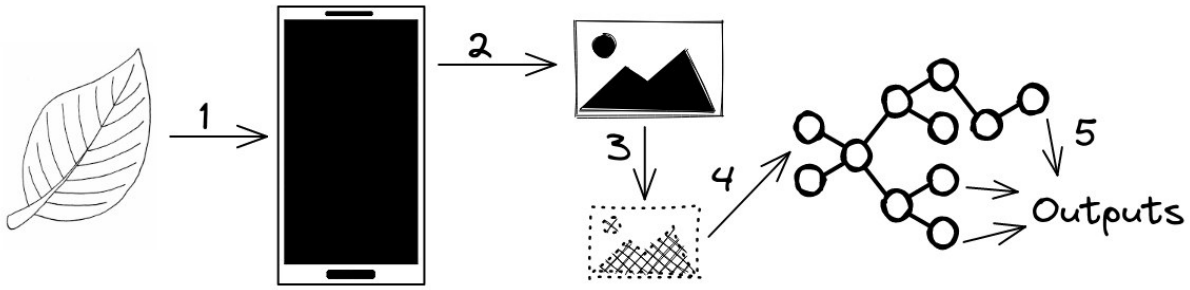


Figure 4.1: Overview of the proposed system, with a mobile app capturing an image of a leaf, forwarding this to a cloud storage, where preprocessing and classification takes place.

In the following sections, we will give an overview of the working principle of each module shown in Figure 4.1, specifying its inputs, outputs and relevance to the proposed system.

4.2 Smartphone Application

The first module in our system is the application installed onto the smartphones of our users, be they farmers or agronomy specialists. The user takes leaf pictures with the app and sends them

into the remaining pipeline. Thus, the app module receives a user action as input and outputs an image that will be processed in the later steps of our pipeline.

Figure 4.2: Image capture through mobile app

Those images can be either taken in real-time from the app itself or chosen from the user's image gallery, giving flexibility to the user when it comes to the moment of the image delivery to our pipeline.

4.3 Image Pre-processing

Once the image has been successfully taken and sent to our storage, it serves as input for the pre-processing module. Pre-processing includes resizing and background removal, eliminating any noise found in the image. Here, noise means any unnecessary picture element other than the leaf itself that may confuse the classification model applied in later steps of our pipeline.

The pre-processing module uses a border detection algorithm that will be specified later. The module standardizes all images. The output is an image containing a centred leaf with a black background, eliminating extra information found in the different areas of the picture. The standardized picture becomes largely independent from the environment where it was taken, be it a crop field, an indoor room or even the top of a car's hood.

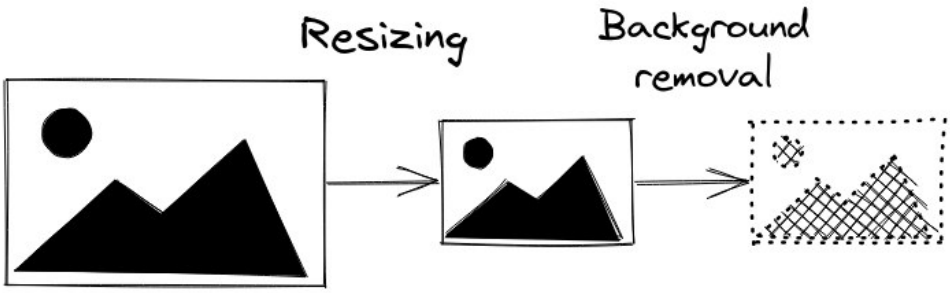


Figure 4.3: Pre-processing steps: image resizing followed by background removal

4.4 Classification module

The classification module is composed of different sub-modules, each built on top of the next. This composition makes it possible to re-utilize existing state-of-the-art models that already excel at their functions, using those as the base of a new model with different inputs and outputs, as seen in Figure 4.4. By the end of the execution of this model, three different outputs are given, as seen in Figure 4.5.

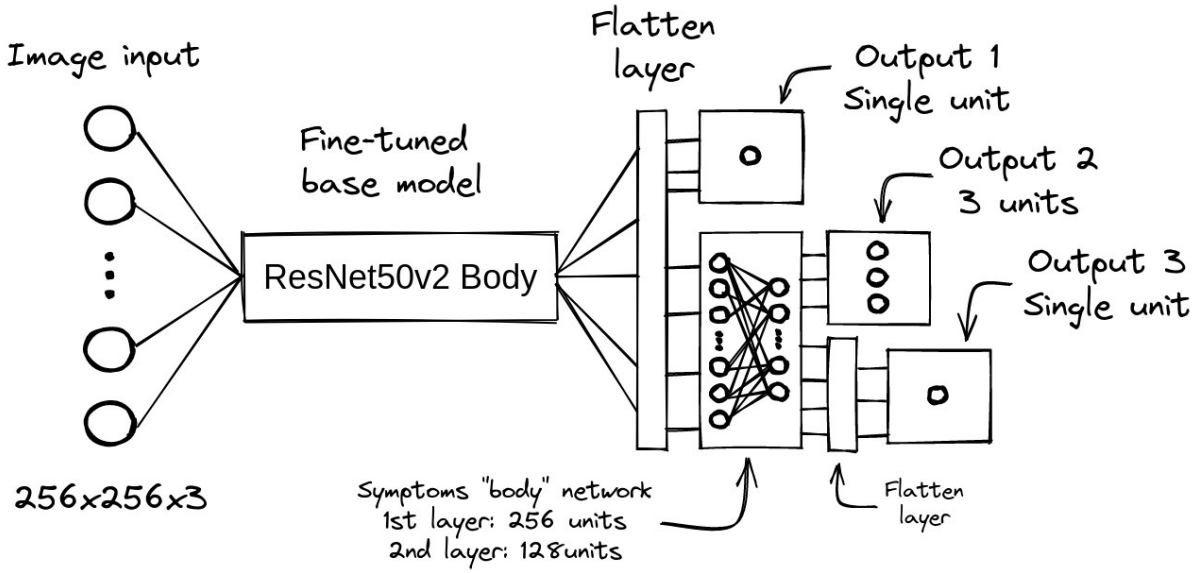


Figure 4.4: Representation of the proposed Neural Network for this work, with its main body being based off pre-trained ResNet50v2 weights, with 3 outputs: checking if a leaf is healthy or not; classifying which symptom is present; checking if the disease is of fungal origin.

4.4.1 Main Body

The main "body" of the proposed architecture is based on the ResNet50v2 model, which is an improved version of the original ResNet model (He, Zhang, et al. 2016b), modified to our

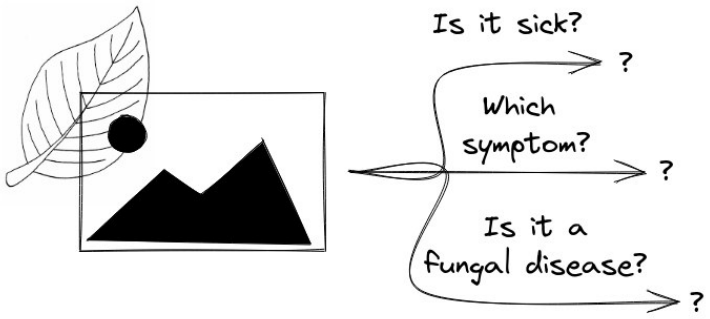


Figure 4.5: Example of outputs given by the classification model, each answering one important question related to disease diagnosis and treatment.

needs by replacing both its input and output units with ones that match our input data and classification outputs.

The input is a single layer containing almost 200000 units representing the image as illustrated in the first part of Figure 4.4. The picture is converted to a matrix of *256x256* elements in agreement with the picture's resolution after pre-processing. Each matrix element is a tuple of red, green and blue (RGB) intensity pixel values.

The original ResNet is a classification model trained on the ImageNet dataset (Russakovsky et al. 2015) with 1000 output units representing the probabilities for any of the 1000 classes found in the original dataset. For our application scenario, however, we eliminate this final layer and replace it with a flattened layer of fully connected units, allowing us to build additional blocks on top of the ResNet body required for the next steps.

4.4.2 Binary disease detector

Our network's first output, Output 1 in Figure 4.4, is a replica of what was done previously by Barros et al. 2021, with a single unit that's fully connected to the outputs of the base model. This unit returns the probability that the leaf depicted in the input image has a disease.

4.4.3 Symptoms detector

We built a new sub-network on top of the outputs of the base model. The first layer contains 256 units fully connected to the second layer which consists of 128 units. Those, in turn, are fully connected to three units that create the second model output, Output 2 in Figure 4.4. Those three units each represent the probability for a specific symptom in the input image, i.e, chlorosis (called yellowing throughout this document), rust spots, and necrosis, respectively.

We chose these symptoms based on the availability of pictures containing those in the dataset and the ease of manually selecting images to be separated for model training.

4.4.4 Fungi detector

The third and final output of the neural network is a single unit, Output 3 in Figure 4.4. The third unit is built on top of the sub-network for symptom classification and in parallel to the second output layer. This output represents the probability that the given input has a fungi-caused disease.

CHAPTER 5

Methodology

This chapter will discuss the methods used during the development of this work. The first section will introduce the used datasets, followed by discussions about the image processing pipelines, and finally, the details on how the proposed model was built and trained.

5.1 Datasets

We built the dataset employed in this project on top of many existing others, such as the ones already discussed in Sections 3.1 and 3.2, along with data provided by the DiaMOS Fenu and Malloci 2021 and Plant Pathology's Fine-Grained Visual Categorization (FGCV7) Thapa et al. 2020 datasets.

The Plant Disease Dataset (PDD) proposed by Liu et al. 2021 is five times larger than the PlantVillage dataset in terms of the number of pictures and the number of available classes. However, the dataset itself is not publicly available, and the sample data offered along their public repository (Liu et al. n.d.) does not have proper labeling. In addition, the authors do not provide a pre-trained model, making it impossible to reproduce or validate their results.

The DiaMOS dataset focuses on images of pears, with 3006 leaf images and 499 fruit images. Only the leaf images were used within this work, containing four major classes: curl, slug, spot, and healthy. As for the FGCV7 dataset, it contains 3642 pictures of leaves of apple trees, containing three major classes: rust, scab, and healthy. Samples of each used dataset can be seen in Figures 5.1, 5.2, 5.3 and 5.4.

Training each output of our model for their specific tasks requires a sub-division of all datasets. For our first output, the classifier that tells us if a leaf is healthy or diseased, all datasets were split into those two classes. We divided the datasets into three classes, each representing one of the chosen symptoms, for training the second output. Lastly, for the third output, pictures were sorted into leaves showing fungi-related diseases and leaves that do not. Tables 5.1, 5.2 and 5.3 give the data distribution for each of the outputs.

A proportion of 80% of the data was used for training, and the remaining 20% for validation. This division occurs at the class level to maintain the original proportions. No data augmentation was employed to try and balance the data, given that the intended objective for this work is only to fine-tune a network, meaning that most of the parameters are actually frozen. Thus, there is no possibility that overfitting on the hidden layers may happen, while the task-specific layers actually have shared components between those, again reducing the risk of overfitting as discussed in Section 2.4. However, balancing the classes could be explored in future works to see if the performance improves.

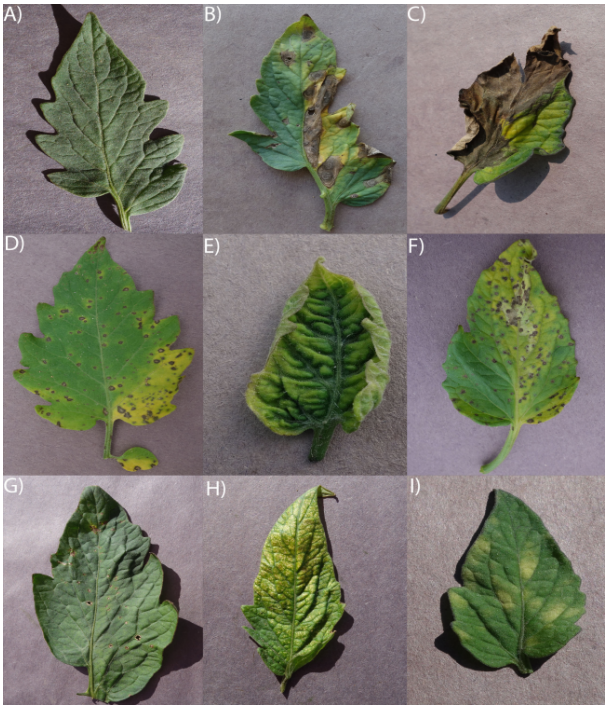


Figure 5.1: Sample data for the PlantVillage dataset. Source: David. P. Hughes and Salathe 2015.

Figure 5.2: Sample data for the dataset made by Barros et al. 2021.

Figure 5.3: Sample data for the DiaMOS dataset. Source: Fenu and Malloci 2021.

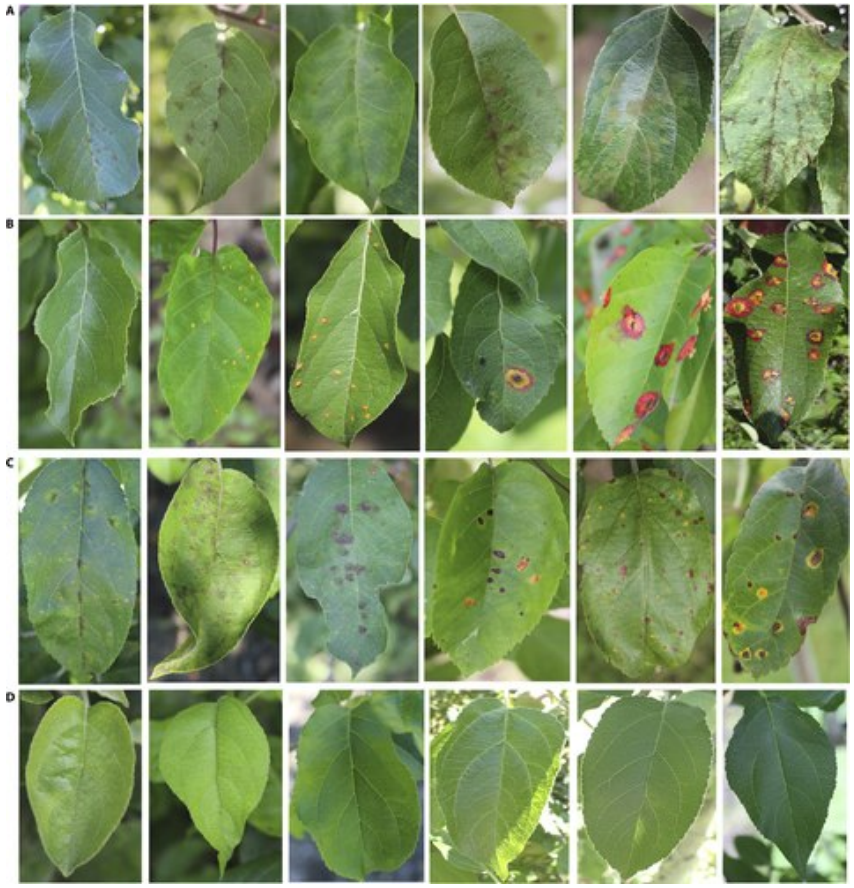


Figure 5.4: Sample data for the FGCV7 dataset. Source: Thapa et al. 2020.

Check if leaf is healthy or not		
Dataset	Healthy	Sick
PlantVillage	15084	39221
DiaMOS	43	2963
FGV7	516	1305
Barros et al. 2021	1987	1302
Total	17630	44791

Table 5.1: Amount of data used to train the output related to disease detection.

Check which symptoms are present			
Dataset	Yellowing	Rust	Necrosis
PlantVillage	10864	4904	8093
DiaMOS	0	0	884
FGV7	0	622	592
Total	10864	5526	9569

Table 5.2: Amount of data used to train the output related to symptom classification.

Check if the present disease is caused by fungi or not		
Dataset	Fungi	Non-fungi
PlantVillage	16710	37599
Total	16710	37599

Table 5.3: Amount of data used to train the output related to fungi detection.

5.2 Image Pre-processing

Pre-processing is needed to standardize images before sending them to a model. The two main components are picture resizing and background removal. Images are down-scaled from their original size to a fixed resolution of 256x256 pixels using OpenCV's (Bradski 2000) resize function with its default bilinear interpolation mode. The chosen value corresponds to the smallest resolution encountered in the final dataset, making it easy to work with as the common denominator.

Different methods were tested for background removal, such as LeafMask (Guo et al. 2021) and MaskRCNN (He, Gkioxari, et al. 2017), which are machine learning-based solutions. The best working solution, however, was to use tools not related to machine learning, relying instead on OpenCV's GrabCut (Rother, Kolmogorov, and Blake 2004) implementation. The approach iteratively segments the picture based on the pixel distribution of the image, checking neighboring pixels to decide if they represent the background or foreground, similar to how a clustering algorithm works. For this work, it is assumed that the leaf is localized in the center of the picture so that the GrabCut algorithm can properly work.

LeafMask showed promise, but reproducing results proved difficult. The source code repository (*easton-cau/LeafMask* n.d.) only had a vague description, and the issue tracker only had comments about other users that weren't able to run their code without any meaningful response, as seen in Figure 5.5.

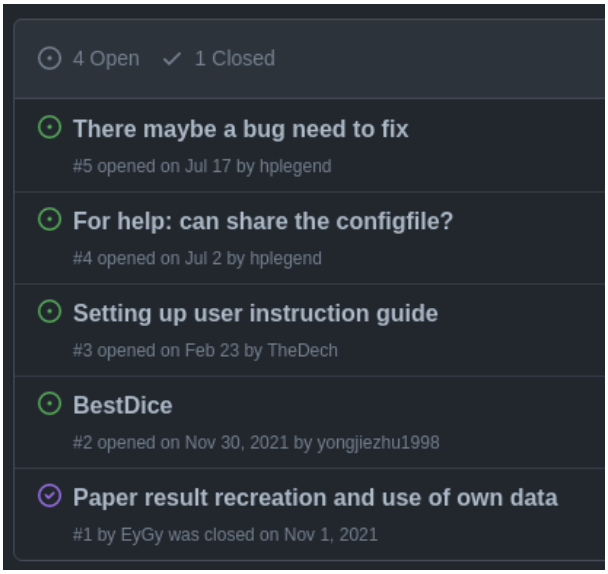


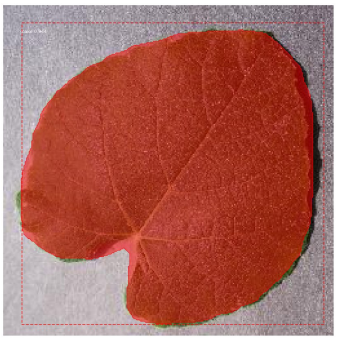
Figure 5.5: Issue tracker for LeafMask on GitHub, without any meaningful response from the authors.

The MaskRCNN model uses outdated libraries, hindering successful execution. Running the model was only possible after isolating all the required environments inside a Docker container (Merkel 2014), making it self-contained with specific library versions that other users reported to work. The default model wasn't good enough to segment the pictures in our dataset,

often mislabeling leaves as other objects (as seen in Figure 5.6) or simply not segmenting the image properly at all (as seen in Figure 5.7). To improve the results, we segmented 200 pictures manually with the help of the VGG Image Annotator (VIA) (Dutta and Zisserman 2019) (as seen in Figure 5.8) and used this to fine-tune the model. Even with this manual segmentation, the results were still underwhelming and useless for our needs.

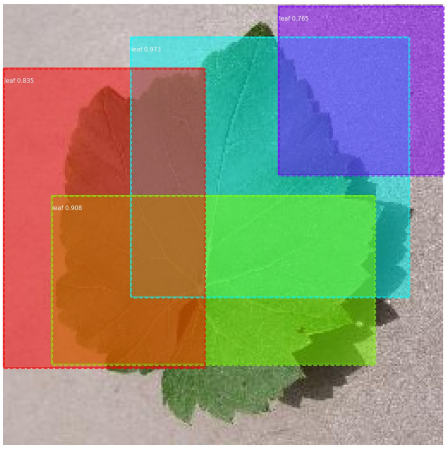
(a) Leaf mislabeled as an umbrella.

(b) Leaves mislabeled as a banana and a vase.



(c) Leaf mislabeled as a cake.

Figure 5.6: Mislabeled images with the default weights from MaskRCNN.



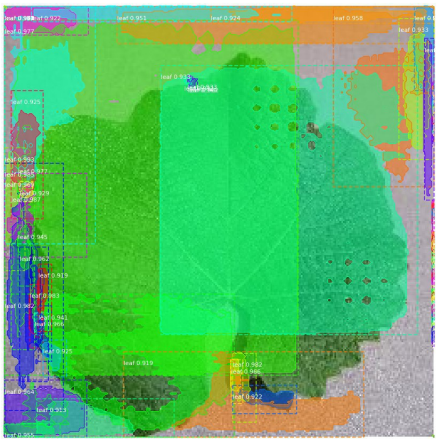


Figure 5.7: Examples of broken segmentation faced during tests with MaskRCNN.

Another promising option, but which ended up not being tested due to the lack of time, was rembg Gatis n.d., which makes use of a lightweight machine learning model called U²-Net (Qin et al. 2020). The tool can remove the background of many different kinds of pictures whilst not needing a dedicated GPU for faster inference times. Thus, we used GrabCut for the background removal. Figure 5.9 shows an example.

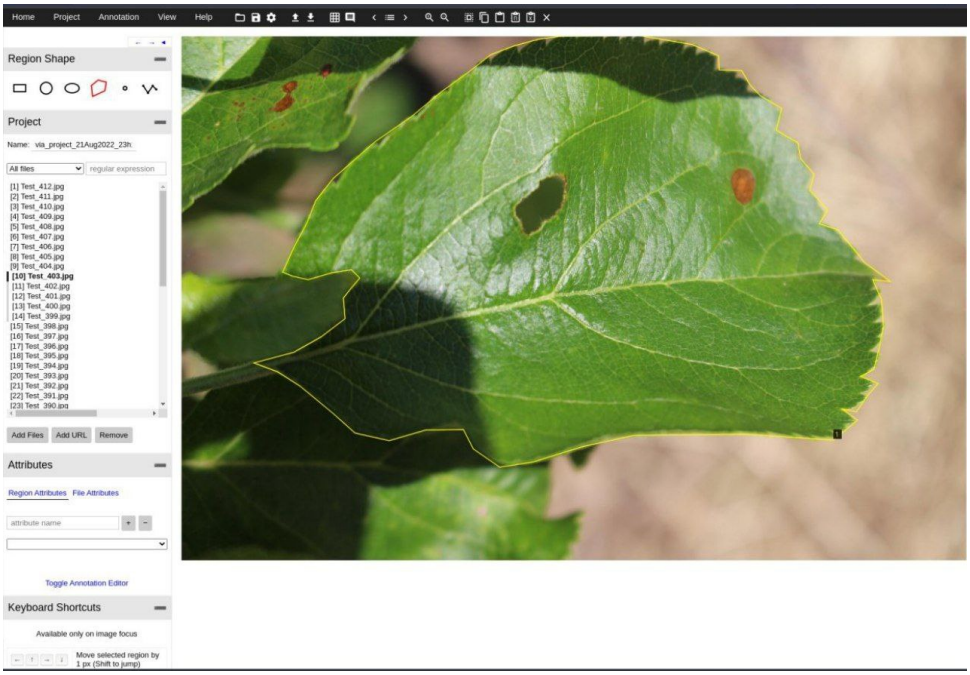


Figure 5.8: Example of manual leaf segmentation using VIA. The yellow border represents the manually added segmentation.

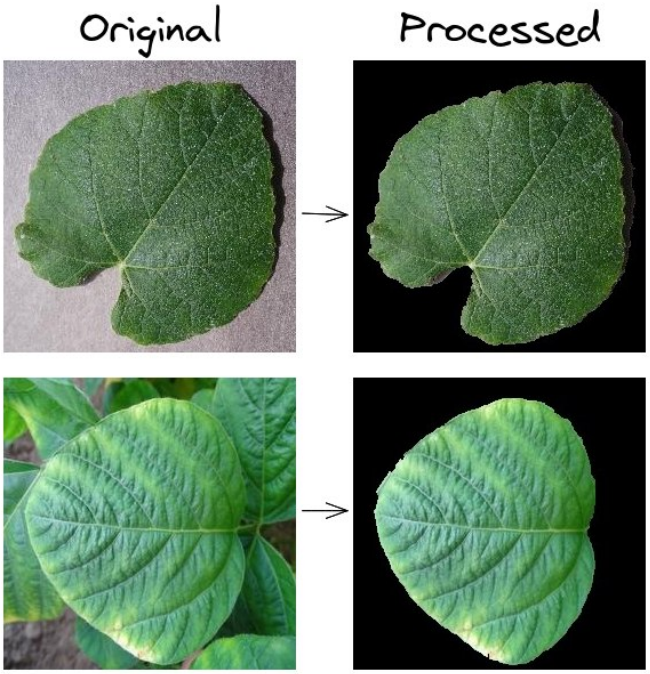


Figure 5.9: Example of original pictures and results of the background removal step using the GrabCut algorithm.

5.3 Multi-task Learning Model

5.3.1 Baseline and Important Remarks

To properly compare our model, the work by Barros et al. 2021 was chosen as the baseline for our first output since both studies use a similar architecture and base model (ResNet50V2). The experiment also proves that the pre-processing methods applied in this work fare better than the ones in Barros et al. 2021. Finally, we advance on the previous results by classifying leaves of other crops rather than just grapes.

While most of the ideas are similar, the actual implementation differs since the chosen framework for this work was Martín Abadi et al. 2015 instead of PyTorch, with the Van Rossum and Drake 2009 language serving as the tool to interface with the used libraries and frameworks. Another aspect in common is the training pipeline used, as seen in Figure 5.10, with the difference that it was repeated three times, once for each different output. For the training of the first output, the second and third outputs-related weights were frozen not to create any interference between the tasks.

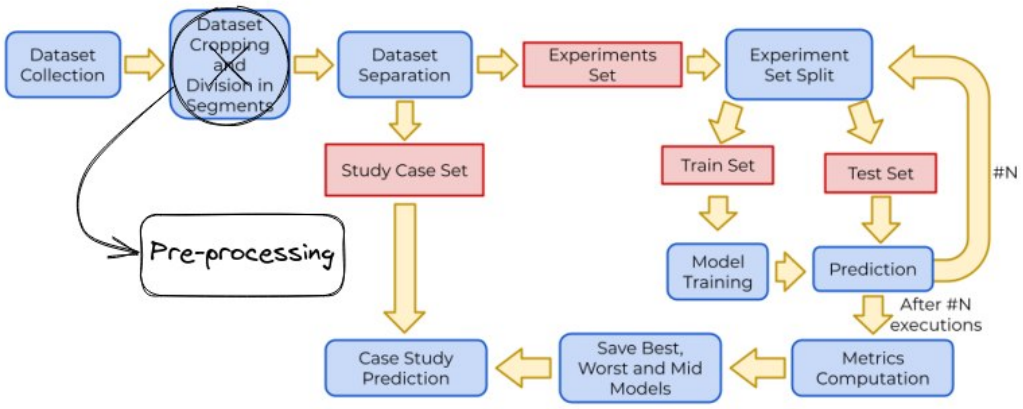


Figure 5.10: Training pipeline flow used, with our proposed pre-processing step being used instead of the original cropping and segmentation steps. Source: Barros et al. 2021.

The Adam optimizer (Kingma and Ba 2014) was used, with a learning rate of 0.0001, a learning rate ten times lower than the default rate used by Tensorflow. A low learning rate is justified since, for our first task, the goal is to fine-tune the network and not to train it from scratch, maintaining the feature extraction capabilities of the base model and also matching the value used by Barros et al. 2021. Table 5.4 gives an overview of the configuration used to compile this model. A difference from the original work is that we used a larger batch size, given that more powerful hardware was available, speeding up the training process. An early stopping function monitored if the model performance during training did not improve for more than three epochs. In other words, model training did not necessarily employ the pre-defined number of epochs but could stop once there were no further improvements.

Pre-trained Model	ResNet50V2
Activation Function	Sigmoid
Loss Function	Binary Cross-Entropy
Optimizer	Adam
Learning Rate	0.0001
Batch Size	1024
Max. Epoch Number	100
Early Stopping	3 epochs
Framework	Tensorflow

Table 5.4: Technical details for the network implementation for the first and third outputs. The second output employs Softmax for the activation function and Categorical Cross Entropy as loss function.

5.3.2 First Task: Binary Health Checker

The task of binary disease classification employs values of "0" and "1" for a healthy and diseased leaf, respectively. Some precautions were taken when training this first output. We optimized the recall metric during the training to avoid false negatives (FN) since their cost is high. By the end of the training, the results are similar to the ones obtained by Barros et al. 2021, even though we used an even larger dataset and similar model, confirming that the original presented idea works as intended and is reproducible. The model returns the probability, a number between 0 and 1, of a leaf being diseased, with examples of those leaves seen in Figure 5.11.

(a) Healthy leaf. (b) Diseased leaf.

Figure 5.11: Samples of the classes detected by the first task, classifying leaves between healthy (a) or not (b).

5.3.3 Second Task: Symptom Classifier

The second task identifies disease symptoms. The input training values were composed of three binary values for each picture, with each value in the array representing the symptom revealed. Only a single value would be a "1", with the other two being "0" for the presence of the chosen symptoms. The already-trained neurons for the first task were frozen, and the sub-network (described in 4.4.3) along with the three units related to the second output was unfrozen, allowing us to load the model again and retrain on the dataset meant for this task. Compiling the model for the second task required two changes to the parameters shown in Table 5.4. First, we used as loss function Categorical Cross Entropy since we are now dealing with multiple classes. Second, we used Softmax instead of Sigmoid for the activation function since our dataset for this task is mutually exclusive for each symptom. The classifier outputs the probabilities for each of the three symptoms, adding up to one. The neuron with the higher probability value is chosen to classify the symptom. Samples of the three classes for this task can be seen in Figure 5.12.

(a) Sick leaf with rust symptom.

(b) Sick leaf with yellowing symptom.

(c) Sick leaf with necrosis symptom.

Figure 5.12: Examples of the different symptom classes inferred by the second task.

5.3.4 Third Task: Fungi Detector

The network configuration was identical to the one used in the first task. Training employs a binary input indicating with a "1" if the disease was of fungal origin and with a "0" otherwise. The sub-network from the second task was frozen, meaning that the fungi detector will try to exploit the features extracted by the symptoms classifier, fine-tuning the results with its single output unit. The model configuration was unchanged from the values found in Table 5.4. The output was the probability between 0 and 1 for a leaf being infested by a fungus. Examples of different pictures with different pathogens, including fungi-caused ones, are shown in Figure 5.13.

(a) Example of tomato leaf with fungi-caused disease.

(b) Example of tomato leaf with virus-caused disease.

(c) Example of tomato leaf with bacteria-caused disease.

Figure 5.13: Samples of different pathogens affecting tomato leaves. It's important to notice that the third task will only indicate of a leaf is afflicted by a fungal disease or not, not discerning the other types of pathogens.

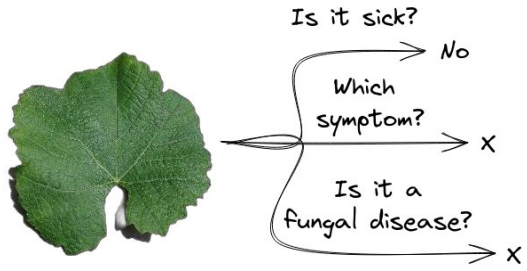
5.3.5 Experimental setup

It's worth noting that the experimental setup used for this work was done on a physical machine compromising a Ryzen 9 5950x with 32 threads, 128GB of RAM, and an Nvidia 3090 with 24GB of VRAM. Both training and testing employed the same machine.

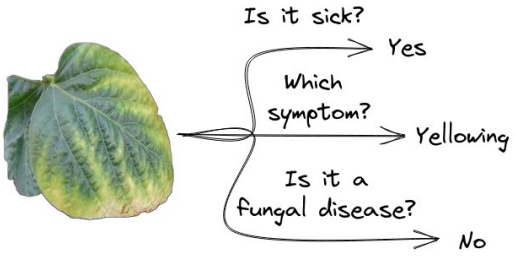
CHAPTER 6

Results

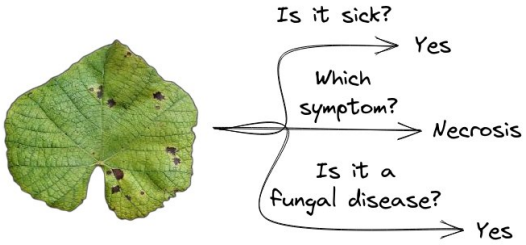
This chapter will present the obtained results for each task and discuss any shortcomings. The metrics used to evaluate a task performance were: accuracy, precision, and recall. With those values, we can also compute the F1-Score for each output. We will also show the evolution of the performance metrics during the training steps. Samples of the classification results from our model can be seen in Figure 6.1.



(a) Sample classification from our MTL model for a healthy leave, with the disease-related outputs being ignored.



(b) Sample classification from our MTL model for a diseased leave, presenting yellowing but which was not caused by a fungi.



(c) Sample classification from our MTL model for a diseased leave due to fungi, showing signs of necrosis.

Figure 6.1: Examples of the results obtained by our final model, being able to simultaneously give results from three tasks, indicating if a leaf is diseased or not, which symptom it has, and if it's caused by a fungi or not.

6.1 First Task: Binary Health Checker

As mentioned, we could improve the already good results obtained by Barros et al. 2021. The boost might be expected, given the increased training dataset. Moreover, the different pre-processing approach applied in this work will have improved the generalization capabilities of the model. Figure 6.2 shows the accuracy values during training, with the final metrics reported in Table 6.1.

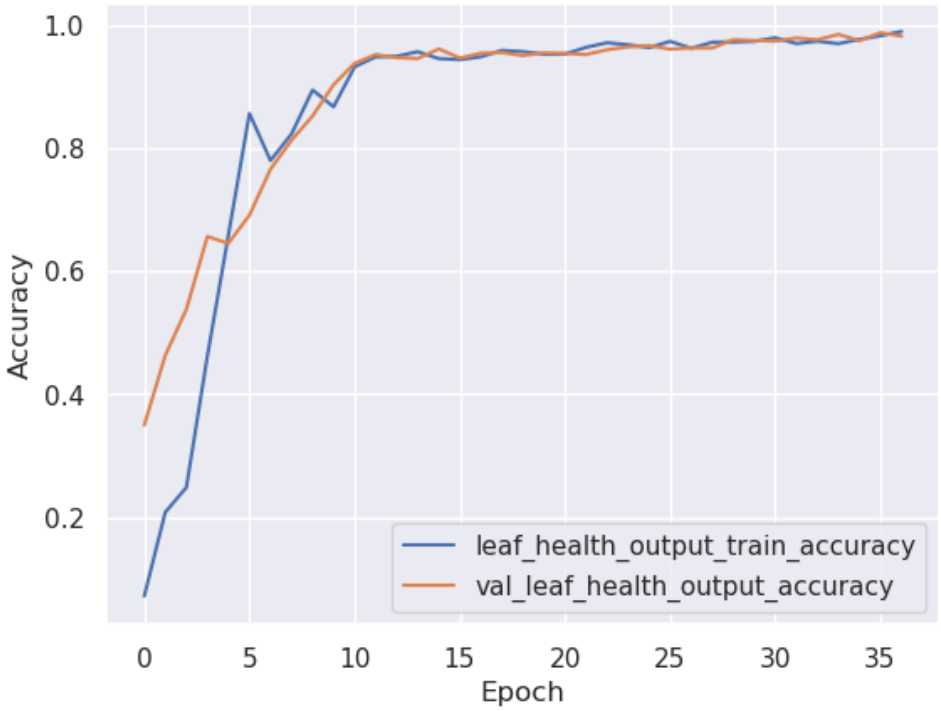


Figure 6.2: Accuracy during the training steps for the first task.

	Barros et al. 2021	Our work
Accuracy	88,2%	96,7%
Precision	78,8%	92,8%
Recall	86,4%	91,3%
F1-Score	85,6%	92,0%

Table 6.1: Resulting metrics for the first task, compared to the results obtained by Barros et al. 2021.

6.2 Second Task: Symptom Classifier

For our second and third tasks, we have no baseline to compare. However, the results achieved with the chosen datasets are still exciting. The second model output tells which macro-symptoms are present in a leaf. One thing worth noting is that since we are using a softmax function, the symptoms are mutually exclusive, so only one of the three symptoms can be present. However, in diseased crops, more than one symptom may be visible. Even simultaneous infection by multiple diseases is possible. Such scenarios were out of the scope of this project. Figure 6.3 shows the evolution of the accuracy metric during training steps. Table 6.2 lists the final metrics achieved.

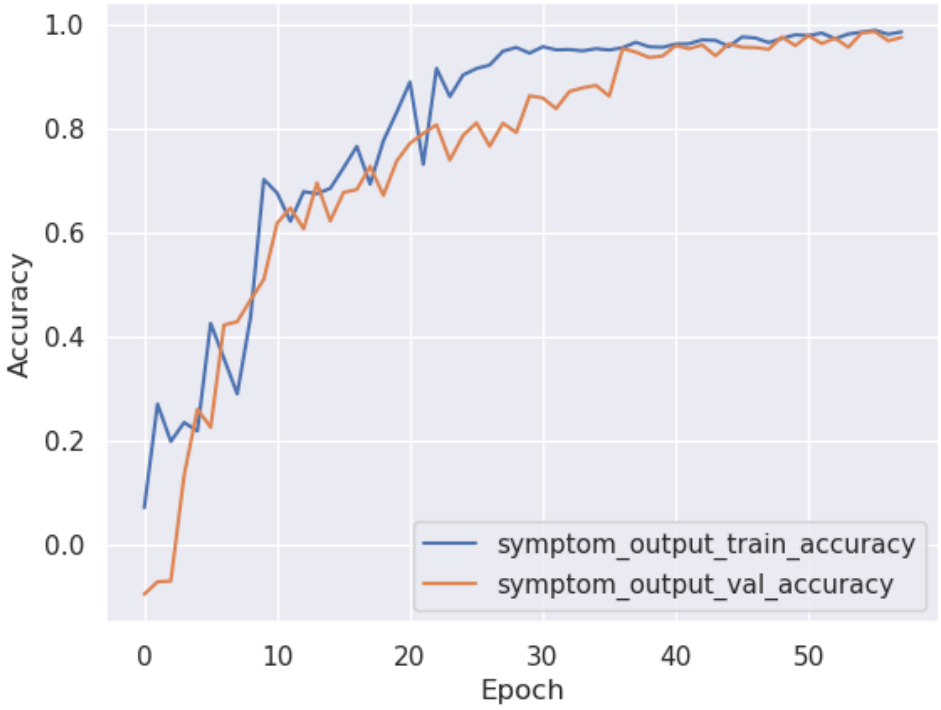


Figure 6.3: Accuracy during the training steps for the second task.

Accuracy	92,9%
Precision	89,9%
Recall	90,7%
F1-Score	90,3%

Table 6.2: Resulting metrics for the second task.

6.3 Third task: Fungi Detector

The third task also achieved good results, summarized in Figure 6.4 and Table 6.1. However, we acknowledge that the results may not represent real-life performance since the dataset used for this task only came from PlantVillage. As noted in Section 5.1, when using only pictures for training obtained in a controlled environment, one may create a model that does not perform well on actual field pictures. Nonetheless, the fungi detector was built successfully on top of the sub-network from the symptom classifier. Thus, we can safely assume that building new tasks on top of others is a great way to share parameters, reducing the number of total parameters needed compared to individual single-task models.

Figure 6.4: Accuracy during the training steps for the third task.

Accuracy	91,2%
Precision	91,1%
Recall	88,5%
F1-Score	89,8%

Table 6.3: Resulting metrics for the third task.

CHAPTER 7

Conclusions and Future Works

This work described a crop disease classifier obtained by multi-task learning (MTL). The MTL classifier analyzes pictures of crop leaves and consists of a binary health checker, a symptom classifier and a fungi detector. The binary health checker beat the baseline results from Barros et al. 2021. We achieved better performance by employing a larger dataset and modified pre-processing of the leaf pictures. Using an MTL model made it possible to easily add new tasks relevant to the problem on top of a basic one, here the symptom classifier and the fungi detector. New outputs do not interfere with existing ones and exploit features reasonable to share.

Several future improvements are possible. Adding more tasks would increase the scope of the suggested model, e.g., outputs detecting different pathogens, such as viruses, mold, or bacteria, or even giving the specific disease name. Balancing the training data might lead to further performance improvements. Human experts could provide feedback triggering retraining to optimize the MTL model for real-world scenarios. Unfreezing some shared layers, both from the base model and the task-exclusive ones, during the training steps of different tasks might also allow them to eavesdrop on each other, with the possibility that those tasks may be able to further improve their performances based on the features learned from their neighboring tasks, which may be also possible by doing a final training step with all of those layers unfrozen with a smaller learning learning rate.

Another idea to expand this project would be to optimize the size of some parts of the model, enabling immediate inferences on the same smartphone used to take the leaf pictures. Such optimization would require changes to the model architecture, mainly immigrating from the ResNetV2 as the main body to another, more lightweight model. Testing different pre-trained models is another possibility to see if they outperform the ResNetV2, with one possible candidate being the current Vision Transformers (ViT). Try to use the first layers of the model to extract only the relevant leaf part from the picture, replacing the pre-processing step, is also worth investigating.

Another topic to be addressed is the mobile app, which could have a better UI/UX, making it easier for the farmers to take and save pictures even if there is no internet available. From the specialist's perspective, the tool currently used to label and monitor leaf pictures needs an overhaul.

Lastly, adding a hardware device to the system to measure the health of the soil would be of great value. Data fusion allows us to differentiate between actual diseases and false positives due to abiotic stress, like the lack of nutrients in the soil. The result is a fully-fledged system that can monitor the presence of diseases and nutrient deficiency during a crop's lifespan.

Bibliography

McCulloch, Warren S and Walter Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Rosenblatt, Frank (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Solomonoff, Raymond J (1957). “An inductive inference machine”. In: *IRE Convention Record, Section on Information Theory*. Vol. 2. Institute of Radio Engineers New York, pp. 56–62.

Linnainmaa, Seppo (1970). “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. PhD thesis. Master’s Thesis (in Finnish), Univ. Helsinki.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). “Learning representations by back-propagating errors”. In: *nature* 323.6088, pp. 533–536.

Huang, Thomas (1996). “Computer vision: Evolution and promise”. In.

Baxter, Jonathan (1997). “A Bayesian/information theoretic model of learning to learn via multiple task sampling”. In: *Machine learning* 28.1, pp. 7–39.

Caruana, Rich (1997). “Multitask learning”. In: *Machine learning* 28.1, pp. 41–75.

LeCun, Yann et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Bradski, G. (2000). “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools*.

Rother, Carsten, Vladimir Kolmogorov, and Andrew Blake (2004). “" GrabCut" interactive foreground extraction using iterated graph cuts”. In: *ACM transactions on graphics (TOG)* 23.3, pp. 309–314.

Van Rossum, Guido and Fred L. Drake (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace. ISBN: 1441412697.

Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Merkel, Dirk (2014). “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux journal* 2014.239, p. 2.

Rigotto, Raquel Maria, Dayse Paixão Vasconcelos, and Mayara Melo Rocha (2014). “Uso de agrotóxicos no Brasil e problemas para a saúde pública”. In: *Cadernos de Saúde Pública* 30, pp. 1360–1362.

Hughes, David. P. and Marcel Salathe (2015). *An open access repository of images on plant health to enable the development of mobile disease diagnostics*. DOI: 10.48550/ARXIV.1511.08060. URL: https://arxiv.org/abs/1511.08060.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Russakovsky, Olga et al. (2015). "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3, pp. 211–252.

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

He, Kaiming, Xiangyu Zhang, et al. (2016a). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

— (2016b). "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer, pp. 630–645.

Mohanty, Sharada P, David P Hughes, and Marcel Salathé (2016). "Using deep learning for image-based plant disease detection". In: *Frontiers in plant science* 7, p. 1419.

He, Kaiming, Georgia Gkioxari, et al. (2017). "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

Huang, Gao et al. (2017). "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708.

IBGE (2017). *Censo Agrícola 2017*. https://censoagro2017.ibge.gov.br/templates/censo/_agro/resultadosagro/index.html. [Acessado em 1 de Março de 2021].

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2017). "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6, pp. 84–90.

Ruder, Sebastian (2017). *An Overview of Multi-Task Learning in Deep Neural Networks*. DOI: 10.48550/ARXIV.1706.05098. URL: https://arxiv.org/abs/1706.05098.

Saha, Sumit (2018). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (Accessed on 09/20/2022).

Dutta, Abhishek and Andrew Zisserman (2019). "The VIA Annotation Software for Images, Audio and Video". In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: ACM. ISBN: 978-1-4503-6889-6/19/10. DOI: 10.1145/3343031.3350535. URL: https://doi.org/10.1145/3343031.3350535.

Crawshaw, Michael (2020). "Multi-task learning with deep neural networks: A survey". In: *arXiv preprint arXiv:2009.09796*.

Harris, Charles R. et al. (Sept. 2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Qin, Xuebin et al. (2020). "U2-Net: Going deeper with nested U-structure for salient object detection". In: *Pattern recognition* 106, p. 107404.

Thapa, Ranjita et al. (2020). "The Plant Pathology Challenge 2020 data set to classify foliar disease of apples". In: *Applications in Plant Sciences* 8.9, e11390.

Wu, Xiongwei, Doyen Sahoo, and Steven CH Hoi (2020). "Recent advances in deep learning for object detection". In: *Neurocomputing* 396, pp. 39–64.

Barros, Mariana da Silva et al. (2021). "Supervised Training of a Simple Digital Assistant for a Free Crop Clinic". In: *Brazilian Conference on Intelligent Systems*. Springer, pp. 162–176.

Fenu, Gianni and Francesca Maridina Malloci (2021). "DiaMOS plant: A dataset for diagnosis and monitoring plant disease". In: *Agronomy* 11.11, p. 2107.

Guo, Ruohao et al. (2021). "LeafMask: Towards Greater Accuracy on Leaf Segmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1249–1258.

Jez, Joseph M et al. (2021a). "Plant pest surveillance: from satellites to molecules". In: *Emerging topics in life sciences* 5.2, pp. 275–287.

— (2021b). "Plant pest surveillance: from satellites to molecules". In: *Emerging topics in life sciences* 5.2, pp. 275–287.

Liu, Xinda et al. (2021). "Plant disease recognition: A large-scale benchmark dataset and a visual region and loss reweighting approach". In: *IEEE Transactions on Image Processing* 30, pp. 2003–2015.

easton-cau/LeafMask (n.d.). `https://github.com/easton-cau/LeafMask`. (Accessed on 07/28/2022).

Gatis, Daniel (n.d.). *danielgatis/rembg: Rembg is a tool to remove images background.* `https://github.com/danielgatis/rembg`. (Accessed on 10/13/2022).

Karpathy, Andrej (n.d.). *Multi-Task Learning in the Wilderness*. `https://slideslive.com/38917690/multitask-learning-in-the-wilderness`. (Accessed on 02/07/2022).

Liu, Xinda et al. (n.d.). *liuxindazz/PDD271*. `https://github.com/liuxindazz/PDD271`. (Accessed on 09/29/2022).