**BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA**
**FACULTY OF MATHEMATICS AND COMPUTER SCIENCE**
**SPECIALIZATION COMPUTER SCIENCE IN ENGLISH**

# DIPLOMA THESIS

# Leaf Disease Detection Using Convolutional Neural Networks

**Supervisor**
**Assoc. Prof. PhD. Iuliana-Maria Bocicor**

*Author*
*Danicico George-Iulian*

2023

**UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA**
**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**
**SPECIALIZAREA COMPUTER SCIENCE IN ENGLISH**

# LUCRARE DE LICENŢĂ

# Detectarea bolilor frunzelor folosind rețele neuronale convoluționale

**Conducător științific**
**Conf. Univ. Dr. Iuliana-Maria Bocicor**

*Absolvent*
*Danicico George-Iulian*

2023

ABSTRACT

Plant diseases can cause significant damage to crops, leading to yield loss and decreased agricultural productivity. Detecting and identifying diseases in an accurate and timely manner is crucial for implementing effective plant disease management strategies.

The domain of computer vision and image classification has seen significant advancements and breakthroughs in recent years. Lately, CNNs have become a fundamental tool for computer vision and have significantly impacted various fields, including healthcare, transportation, security, entertainment, and more, with their ability to extract high-level features and make accurate predictions from complex visual data.

This thesis suggests three different strategies for the disease detection using leaves of 6 plant species, and for the classification of these species. The strategies include a custom model trained from scratch, a transfer learning-based architecture and a multi-task learning approach. There were conducted multiple experiments on the implemented the models. The InceptionV3 architecture was used for the transfer learning approach, as well as for one model which used the aforementioned architecture as a feature extractor for the multi-task learning strategy. This paper presents multiple original solutions, in the form of three models that were trained on the same dataset, PlantVillage [1] and obtained great results, some which were higher than the results obtained by the existing methods. The highest accuracy obtained during this research was 98.35% for the plant classification task and 94.55% for the disease detection task.

The models were later integrated in a web application, which allowed non-professional users, but it can also prove beneficial to specialized users, such as farmers or gardeners, providing them with valuable assistance in their specific domain. The user is able to upload an image of a leaf and will see two predictions, one for the plant classification task and a top three prediction for the disease detection task. This work is the result of my own activity. I have neither given nor received unauthorized assistance on this thesis.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Crops play a significant role in our society by fulfilling several important functions. They constitute the primary source of food providing us with everything that is required for our growth, development and overall health [6]. Proper crop management practices also have significant economic benefits. By ensuring that crops are healthy and have high yields, farmers can increase their profits, which in turn contributes to economic growth. Agriculture is a vital source of income for many countries, and taking care of crops can improve the livelihoods of farmers and support local economies.

Sustainable agriculture practices are another essential aspect of taking care of crops. Farmers can implement sustainable practices like reducing the use of harmful chemicals, conserving water resources, and preventing soil erosion. These practices have a positive impact on the environment, contributing to biodiversity and protecting natural resources. They also promote sustainable agriculture, which is necessary for long-term food security and economic growth. Climate change is having a significant impact on agriculture and crop production around the world. As temperatures rise, it can create conditions that are more favorable for the development and spread of some plant diseases. Changes in precipitation patterns can also impact crop health. Excessive rainfall can lead to soil erosion and waterlogging, which can damage crop roots and create conditions that are favorable for the development of root diseases.

Moreover, taking care of crops has social benefits. As stated in [7], agriculture provides employment for approximately 28% of the world's population is employed in agriculture, which translates to about 1.3 billion people. As a result, agriculture ranks among the top occupations globally, especially in emerging nations where it is a significant source of employment and revenue.

The motivation for choosing this topic is to contribute to the development of more efficient and accurate methods for detecting diseases using leaves. Advanced technology and machine learning can play a crucial role in preventing plant dis-

eases by providing early detection, rapid response, and targeted control measures. Machine learning algorithms can be trained to analyze images of plants to identify symptoms of disease. This can help identify diseases at an early stage in a timely manner [8].

This thesis focuses on the theory, design, and experimentation of multiple models for classification and detection problems. The study is structured into five chapters, which are as follows:

- The first chapter outlines the motivation behind the research, introduces the problem of leaf disease detection, and discusses the potential impact and benefits of utilizing machine learning techniques for this task.
- The second chapter provides an overview of the existing research and related work in the field of leaf disease detection using machine learning techniques.
- The third chapter of the thesis presents the theoretical foundations and concepts of machine learning. The fundamentals of convolutional neural networks, transfer learning, and multi-task learning are elucidated, alongside the principal algorithms and methodologies employed in these techniques.
- The fourth chapter provides an overview of the proposed methodology using convolutional neural networks, transfer learning, and multi-task learning to address the challenges related to leaf disease detection and plant classification problems. The dataset used in the study, along with the implementation details and experimental results are presented. This chapter also incorporates the integration of the top-performing model into a web application.
- The concluding chapter of the thesis provides an overview of the primary contributions and outcomes of the research study. The limitations of the proposed approach are highlighted, and possible directions for future improvements are discussed.

# Chapter 2

# Literature Review

## 2.1 Problem Description

Plant diseases result in significant decreases in agricultural productivity and present a huge risk to the overall global food security.[9]. Early detection and diagnosis of plant diseases are crucial to prevent their spread and mitigate their impact. Leaf diseases, in particular, are challenging to detect due to variations in their appearance and the complexity of differentiating between different types of diseases.

The development of automated systems for detecting diseases based on leaf images using machine learning algorithms has shown promising results. However, several challenges need to be addressed to make these systems more effective. These challenges include variations in the appearance of healthy and diseased leaves, variations in lighting and background, and the complexity of differentiating between different types of diseases. Convolutional Neural Networks [10] have shown remarkable success in image recognition tasks, including the detection of plant diseases. CNNs are capable of learning complex features and patterns from input images, making them ideal for detecting diseases. CNNs also have the ability to generalize well to new data, making them more robust to variations in the appearance of leaves and lighting conditions.

This study proposes the use of machine learning algorithms, specifically Convolutional Neural Networks, for the classification of leaves and detection of leaf diseases. Three different approaches were explored: CNNs, Transfer Learning [5], and Multi-Task Learning [11]. Transfer Learning leverages pre-trained models to improve the learning process with limited data, and Multi-Task Learning addresses the challenge of detecting multiple diseases simultaneously. The proposed methods aim to address the limitations of existing techniques and provide a more efficient and accurate approach to classify and detecting leaf diseases. The results of this study have the potential to improve the efficiency and accuracy of disease detection

in plants, contributing to sustainable agriculture and food security.

## 2.2 Existing Solutions

### 2.2.1 Leaf Disease Detection and Classification using LVQ Algorithm

Article [12] proposes an approach using Learning Quantization Vector. The dataset that has been used consists of 500 images of tomato leaves from the PlantVillage dataset [1]. In the pre-processing phase, the selected images have been resized from 256 x 256 to 512 x 512. For this experiment, the author uses five distinct classes, where four classes represent leaf diseases and one class represents healthy leaves. To initiate the convolution process for each image within the dataset, three separate input matrices have been generated for the red, green, and blue channels. Each input image matrix underwent four successive convolutions with ReLU activation function being applied after each convolution. Subsequently, the resulting matrix was subject to max pooling operation with stride two. The first and second convolution stages employed a 9 x 9 filter, while a 5 x 5 filter was employed in the third and fourth convolutions. The process outlined above resulted in three distinct 3 x 3 matrices that corresponded to the red, green, and blue channels. Upon applying these operations to an RGB feature image, three distinct 3 x 3 matrices were generated, with each matrix originating from its respective R, G, or B channel. The matrices underwent a transformation process to yield a vector of dimensions 27 x 1, which was utilized as the input for the neural network's input layer. The training set was composed of 400 feature vectors derived from the original images, while the testing set consisted of the remaining 100 vectors. The LVQ algorithm utilizes a Kohonen layer in its neural network architecture, comprised of 50 neurons, with 10 neurons allocated to each of the available classes.

The average accuracy was 86%. The highest accuracy (of 90%) was obtained for the healthy leaves and the bacterial spot disease as shown in Table 2.1

### 2.2.2 Detecting Tomato Leaf Diseases using CNN

Article [2] presents a comparison between different pre-trained models and a proposed model with three convolutions, which has been improved using Panda. Panda is a python library which can improve the accuracy by removing the noise, handling missing values and transforming the data into a suitable format.

For this experiment images depicting tomato leaves have been used from the

| Disease | Healthy | Septoria spot | Late blight | Bacterial spot | Yellow curved | Accuracy |
|---|---|---|---|---|---|---|
| Healthy | 18 | 0 | 0 | 0 | 2 | 90% |
| Septoria spot | 1 | 16 | 0 | 0 | 3 | 80% |
| Late blight | 0 | 0 | 17 | 0 | 3 | 85% |
| Bacterial spot | 0 | 0 | 0 | 18 | 2 | 90% |
| Yellow curved | 0 | 0 | 0 | 3 | 17 | 85% |
| Average | | | | | | 86% |

Table 2.1: Results based on the confusion matrix.

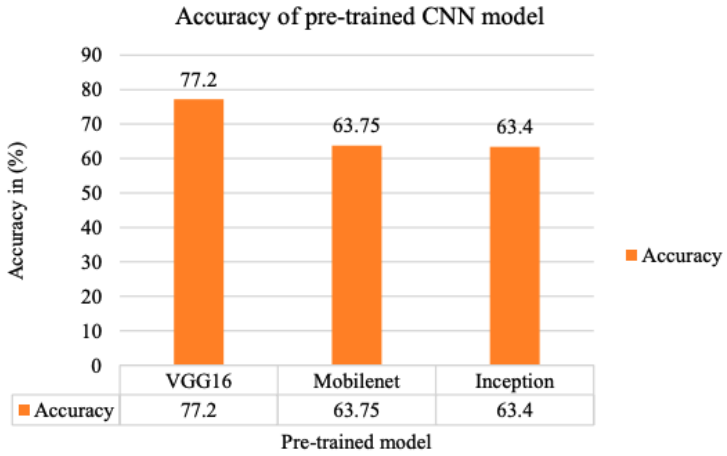


Figure 2.1: The obtained accuracy of the pre-trained models.[2]

Plant Village dataset [1]. There were ten categories of leaves, nine of them represented diseases and one represented healthy leaves. Using the Augmentor package from Python, the author performed dataset pre-processing by generating new images from the original images through techniques like rotation, flipping, cropping, and resizing. This resulted in an equal number of images (1000) for each class. When a class had more images than 1000, the first 1000 were picked.

The proposed model architecture consisted of a sequence of three convolutional layers, followed by three max pooling layers. This was further followed by a flatten layer, a fully connected layer, and finally an output layer. In order to improve accuracy, the Panda library has been used. The accuracy of the model after 1000 epochs was 91.2%.

For the purpose of comparing performance, the author evaluated three pre-trained models, namely VGG16, Mobilenet, and InceptionV3. The obtained accuracy of these pre-trained models is illustrated in Fig. 2.1.

### 2.2.3 Grape Disease Detection using Multi-task learning

A relevant approach is proposed in [3]. The diagnosis and classification of grape diseases rely on the utilization of Region-based CNN (R-CNN) and Multi-task learning techniques. The first four blocks of the ResNet-18 architecture are utilized to extract

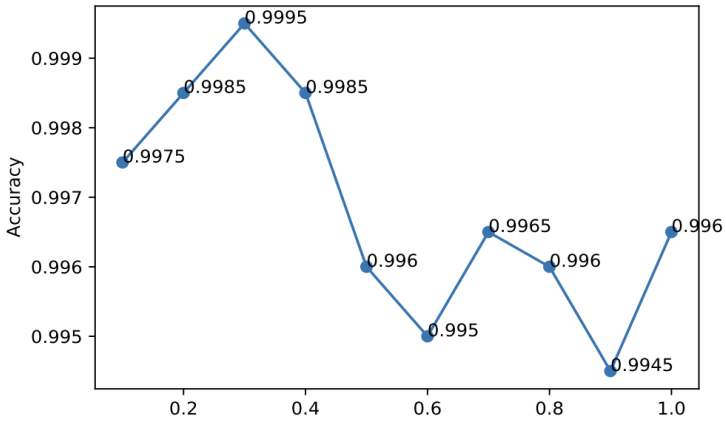


Figure 2.2: The model's accuracy based on various values assigned to the loss weighting factor [3]

features. Subsequently, the feature maps at various levels were merged to form attention vectors across channels, weights, and space. The resulting features were then assessed to determine their suitability for use in a region proposal network (RPN). In the ResNet architecture, at the fifth stacked block and the region proposal network, the upper block layer of the R-CNN is responsible for identifying the proposed object regions, while the lower layer determines the corresponding class label for each of the proposed regions.

The parameter configurations were adopted according to the methodology outlined in [13]. The region proposal network used a batch size of 256, and the author determined a loss weighting factor $\lambda$ of 0.3 through experimentation with values ranging from 0 to 1.

The model's performance evaluation involves the utilization of 4639 grape images sourced from the PlantVillage dataset. The author split the dataset into three distinct sets, namely training set, validation set, and test set, with the proportions of 57%, 13%, and 30%, respectively. The evaluation of the grape disease detection network is presented in two stages. The model's performance is evaluated by averaging the results from two stages, while the validation set is utilized to prevent overfitting. Initially, the grape dataset is utilized to train the ResNet-18 model. Subsequently, the proposed model is employed for the disease detection and classification of diseased objects. Based on the results, the proposed approach demonstrates enhanced efficacy in object detection and attains an overall accuracy of 99.93%.

### 2.2.4 Plant Disease Identification using Conditional Multi-Task Learning

The article [14] introduces Conditional Multi-task Learning (CMTL), a new approach which employs a conditional learning mechanism to learn task-specific features that depend on another task. Two methods, namely conditional feature-wise linear modulation and conditional feature fusion, were employed to achieve conditional probability. The approach based on feature fusion aims to enable disease-specific features to be integrated smoothly with species-specific features for disease prediction modeling by fusing the features. In this study, feature-wise linear modulation was used as a conditional normalization method to adapt the features of the disease in the fully connected layer. This modulation process was performed based on the host species information, thereby incorporating conditional adjustments into the model.

The collection of images utilized in this study was comprised of data from various sources, including the Plant Village dataset [1], Digipathos dataset [15], Pl@ntNet database [16], and IPM website [17]. A set of 12,290 images comprising 7,163 healthy images and 5,127 unhealthy images were collected and subsequently partitioned into training and validation sets. The training set accounted for 80% of the total images (10,324), while the remaining 20% (1,966) were assigned to the validation set.

| Dataset | Training data | Validation data |
|---|---|---|
| PlantVillage | 594 | 146 |
| IPM | 4522 | 812 |
| Pl@ntNet | 3540 | 663 |
| Digipathos | 1668 | 345 |
| Total | 10324 | 1966 |

Table 2.2: The distribution of the images between training and validation sets

The model was trained after performing several pre-processing steps on the images, which included resizing to 299 x 299, as well as implementing random cropping, mirroring, and color data augmentation. The networks were trained using the stochastic gradient descent and back-propagation algorithm.

Inception-V3 was utilized as the basis for the proposed model. A new logit layer was utilized to classify the joint species-disease categories, replacing the original activation layer of Inception-V3. The disease detection accuracy reached a maximum of 82.96%, while the host species classification accuracy reached 78.64%. The overall average prediction accuracy per class was found to be 64.79%. The obtained results were achieved with the disease layer being conditioned on the species layer.

# Chapter 3

# Background

## 3.1 Deep Learning and Artificial Neural Networks

The human brain is the most complex entity in the world and it serves as the model for neural networks. Neurons are the building blocks of the human brain. Any neural network has neurons as its most fundamental computational unit. Getting a better knowledge of how information is processed in biological systems is, in fact, a major goal of the majority of current research into neural network algorithms.[18]

### 3.1.1 Structure of a Neural Network

The structure of an artificial neural network encompasses a series of interconnected nodes that comprise an input layer, one or more hidden layers, and an output layer. Each node, also known as an artificial neuron, is linked to another and has a weight, threshold, and activation function associated with it. The architecture of the artificial neurons is designed to resemble the structure of biological neurons. The input vectors in an artificial neural network model are analogous to the dendrites of biological neurons, which receive signals from nearby neurons. Processing and determining whether to transmit the signals further is performed by the cell body or soma, which in the model is represented by the node. The model's output corresponds to the biological axon, which transmits processed information to other neurons. The synapses in the biological neuron are mirrored in the model as weighted connections.

The activation function is responsible for determining the node's output based on the input it receives. When a node's output surpasses its threshold value, it becomes active and transmits the data to the next layer of the neural network. In contrast, when a node's output is below the threshold, it does not pass any data to the following layer of the network.

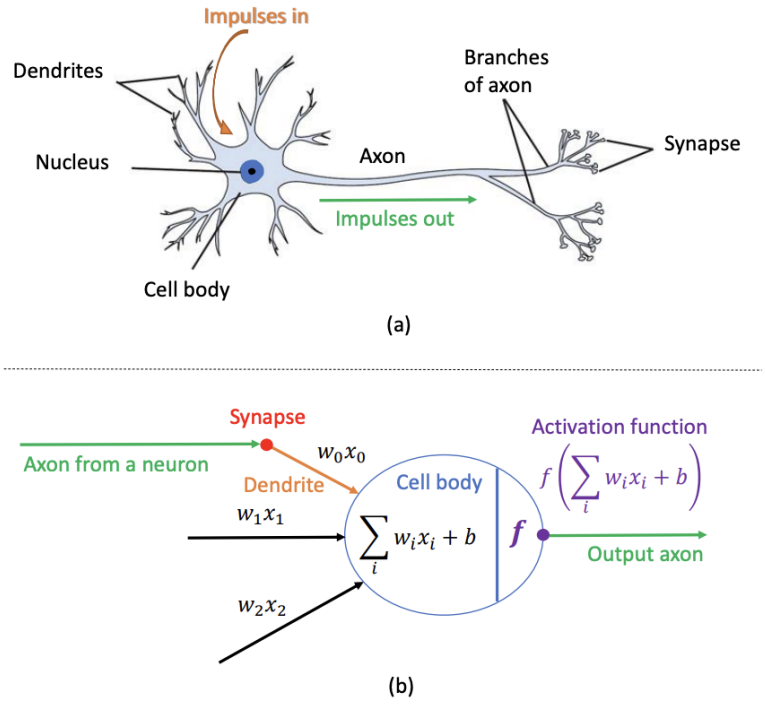


Figure 3.1: The similarity between the human neuron illustrated in image (a) and the artificial neuron illustrated in image (b).[4]

### 3.1.2 Activation Functions

The activation function employed in a neural network may either be linear or non-linear. However, non-linear activation functions are more prevalent as they overcome some of the limitations associated with linear activation functions, such as their inability to facilitate backpropagation [19]. Additionally, the combination of all network layers into a single layer, which may occur with a linear activation function, presents another disadvantage since the final layer merely constitutes a linear aggregation of the initial layer's outputs. Some of the most common and used activation functions are:

**Sigmoid Activation** - It is a non-linear function that is often used in binary classification problems, because it maps any real number between 0 and 1. Its formula is: $\sigma(x) = \frac{1}{1+e^{-x}}$ and one big advantage is that the function is differentiable, which means it can be used in gradient-based algorithms, such as backpropagation [19].

**Rectifier Linear Unit (ReLU)** - It is a non-linear function and one of the most popular activation function these days. Its formula is: $f(x) = max(0, x)$. It is computationally efficient and represents a suitable choice for large-scale neural networks. The ReLU function suffers less from the vanishing gradient problem, compared to

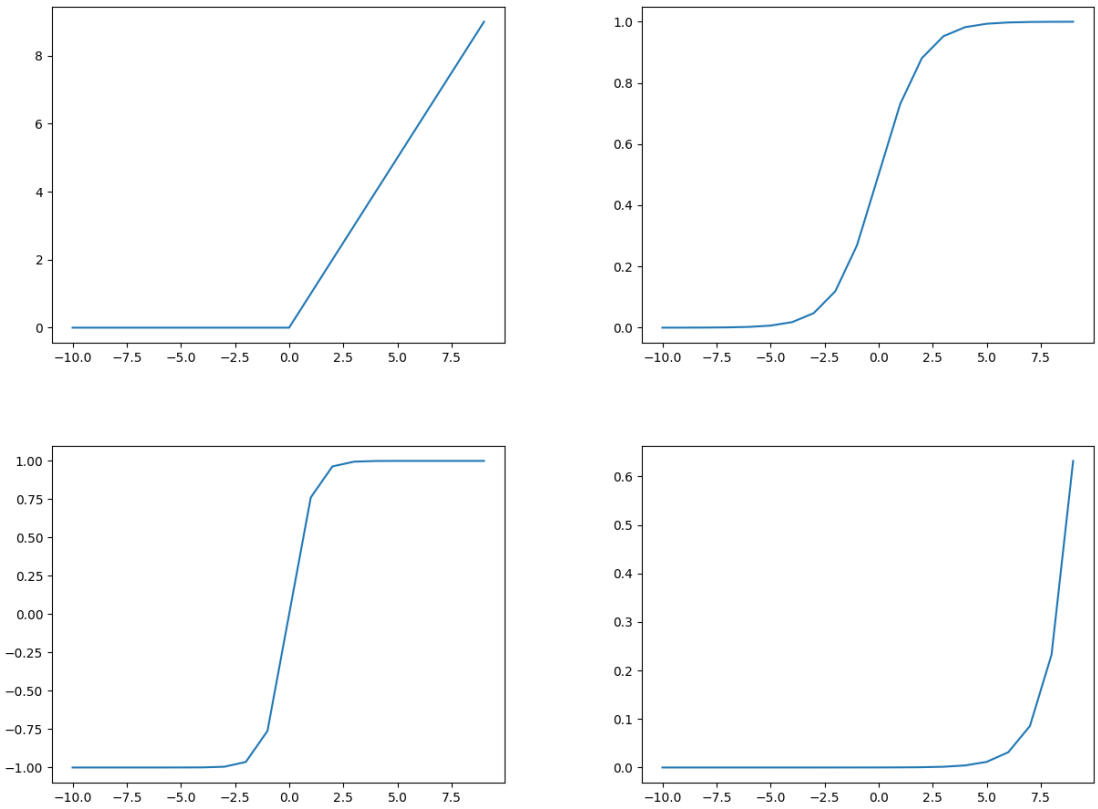


Figure 3.2: The representations of the ReLU, Sigmoid, Tanh and Softmax activation functions.

other activation functions such as the sigmoid function.[20]

**Hyperbolic Tangent Activation** - It is a non-linear function, similar to the sigmoid function, but its output ranges between -1 and 1 [21]. Its formula is: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

**Softmax Activation** - It is a non-linear activation function which is used mostly in problems which require multi-class classification. Its formula is: $softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$. It takes a real number vector as input and generates an output vector with values ranging from 0 to 1, where the sum of all elements equals 1 [19].

In an artificial neural network, the learning process is achieved by adapting the weights, which depend on the variance between the projected output and the factual output. The feedback of this deviation travels in a backward direction through the network, a technique known as backpropagation, employing the link weights. Figure 3.3 illustrates this process. To minimize the cost function, the network uses gradient descent, which finds the lowest point where the cost is minimized. Initially,

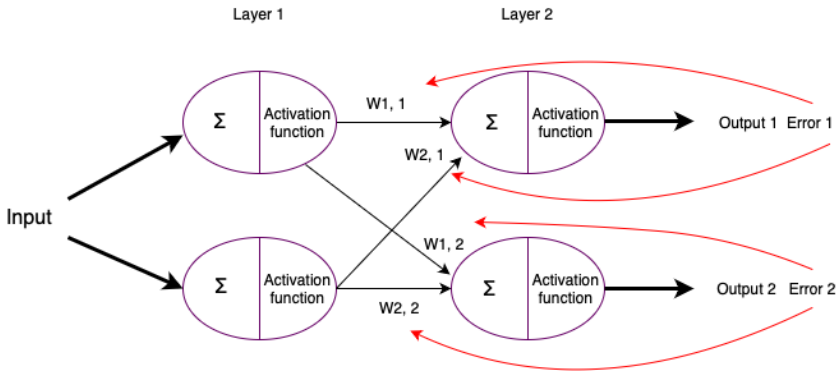


Figure 3.3: Representation of how the weights are updated during the backpropagation phase based on the error.

the network starts with randomly assigned weights, which are gradually adjusted to reduce the cost until it reaches a minimum [22].

## 3.2 Convolutional Neural Networks

In the field of deep learning, CNNs, or convolutional neural networks, are a type of artificial neural network that are frequently employed for image processing [23]. Since its creation, CNNs have seen a quick evolution in terms of design, and recently, they have produced outcomes that were previously assumed to require human interaction solely. There are numerous architectures available today that can be customized for particular needs and limitations.

One of the first architecture was Neocognitron, the earliest precursor of Convolutional Neural Networks introduced the concepts of features extraction, pooling layers and convolutional usage in a neural network.[24] Yann LeCun and his team introduced for the first time the notion of Convolutional Neural Network in the article [25]. They proposed it for the recognition of handwritten digits.

Convolutional Neural Networks are a distinct form of deep feedforward neural networks that employ the mathematical operation of convolution to extract intricate features from input data. Initially created to process visual data, CNNs have rapidly gained acceptance across a wide range of domains, beyond their original image processing domain, due to their superior performance over existing techniques. As a result, they have emerged as a highly influential and disruptive technology, significantly impacting the field of artificial intelligence [10].

### 3.2.1 Architecture

There are many different CNN architectures, but they all follow the same pattern [10]. The CNN essentially transmits the data through interconnected layers and

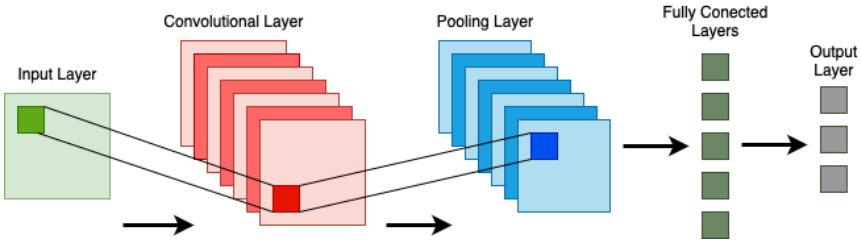


Figure 3.4: Basic architecture of a CNN

transforms the input layer information into a set of class scores determined in the output layer.

The layers of a convolutional neural network can be classified into three primary categories: the input layer, the convolutional layer responsible for identifying features within the input data, and the classification layers, which include pooling layers and fully connected layers.

The feature-extraction component is the brain of convolutional neural networks. In this stage, machine learning takes place, and patterns are discovered automatically rather than manually as they are in conventional algorithms. Automatic learning is made possible by the layers in this component, which look for various features in images and gradually build up higher-order features. In essence, this procedure serves as the framework for the network's image analysis capabilities.

### 3.2.2 Convolutional Layer

The convolutional layer is a fundamental component of a CNN and is responsible for performing a majority of the computations. Its required constituents include an input data, a filter, and a feature map. Assuming that the input constitutes a color image comprising a 3D matrix of pixels, the input will comprise three dimensions: height, width, and depth, which correspond to the RGB values. In addition, a kernel/filter or feature detector is employed to examine the receptive fields of an image for the existence of a feature.

To perform convolution, a filter is moved across a section of the input image while computing the dot product between the pixel values of the input and the filter. After each convolution operation, the filter is moved a fixed distance called the stride length, and the process is repeated until the kernel has covered the entire input image.

A feature detector is a matrix of weights arranged in two dimensions, which corresponds to a specific area of an input image. While the dimensions of the detector

may vary, a 3 x 3 filter size is commonly used to determine the receptive field's size. Convolution involves sliding the filter over a portion of the input image and calculating the dot product between the input pixel values and the filter. The results are then saved in an output array, which represents an activation map, feature map, or convolved feature. The filter is then shifted by a predetermined stride length, and the convolution process repeats until the entire input image is covered by the kernel.

The crucial characteristics of the convolutional layer include local connectivity, shared weights and bias, as well as the spatial arrangement. The local connectivity is managed with a hyperparameter called the receptive field. Throughout the input volume, the depth-wise connectivity remains unchanging and matches its depth. It is essential to acknowledge that there exists a discrepancy in the handling of the spatial dimensions, i.e., width and height, as compared to the depth dimension. Despite connections being confined to the local 2D area of width and height, they are always comprehensive throughout the entire depth of the input volume.[23].

The spatial arrangement dictates how the neurons or units are arranged in the layers that process input data. Typically, input data for a CNN takes the form of a multi-dimensional array, where each dimension corresponds to a specific aspect of the input, such as width, height, and color channels. In a CNN, the neurons in each layer form a grid-like structure and are connected to a local patch of neurons in the previous layer. This local patch of neurons is referred to as the receptive field, and its size dictates the size of the filter utilized in the convolution operation. The spatial arrangement is crucial in identifying spatial features in the input data. By connecting every neuron to a local patch of neurons in the previous layer, the network can learn to recognize patterns and structures that are spatially correlated. Additionally, CNNs can capture spatial features at different scales by utilizing various filter sizes and pooling operations, enabling them to acquire hierarchical representations of the input data.

Lastly, parameter sharing is important in a convolutional layer because it enables the layer to learn and detect similar features across different regions of an input image.

#### Pooling Layer

In convolutional neural networks it is customary to introduce a pooling layer subsequent to the convolutional layer, with the possibility of being repeated multiple times in a single model. The pooling layer operates on a set of feature maps and produces an equivalent set of pooled feature maps. The pooling process involves choosing a pooling operation that works similarly to a filter applied to the feature maps. Typically, the dimensions of the filter or pooling operation are smaller in size

than those of the feature map, with the most common dimensions being 2 x 2 pixels, and a stride of two pixels.

The most common pooling operations are:

- **Average Pooling** - compute the mean value of each patch from the feature map.
- **Maximum Pooling** - as the name suggests, it computes the maximum value within each patch of the feature map.

The pooled feature maps generated by the pooling layer serve as a condensed representation of the feature maps identified in the input. Their significance lies in the fact that minor shifts in the feature's location in the input detected by the convolutional layer lead to a pooled feature map where the feature maintains the same location. Consequently, this process enhances the model's capacity for invariance to local translation.

**Flatten Layer**

The purpose of the Flatten layer is to transform a multi-dimensional input tensor into a one-dimensional tensor, which is subsequently fed to the fully connected layers. This layer is useful when working with fully conected layers that require a one-dimensional input.

**Global Average Pooling Layer**

The main objective of including the Global Average Pooling layer is to downsize the spatial dimensions of the feature map, while simultaneously conserving the channel information. This layer is usually used for classification tasks due to its ability to reduce the number of parameters in the network and mitigate the risk of overfitting.

The final component of a CNN is the classification component, consisting of multiple fully connected layers. In a fully connected layer, every neuron is connected to all neurons in the previous layer. The last layer of a network is considered the classification layer, which applies a different activation function, usually a softmax or sigmoid for classification tasks to compute the possible output classes.

## 3.3 Transfer Learning

Transfer learning is a technique in the field of machine learning that involves training a model on a source task or domain and utilizing the acquired knowledge to facilitate the training of a target model on a related task or domain. The reasoning

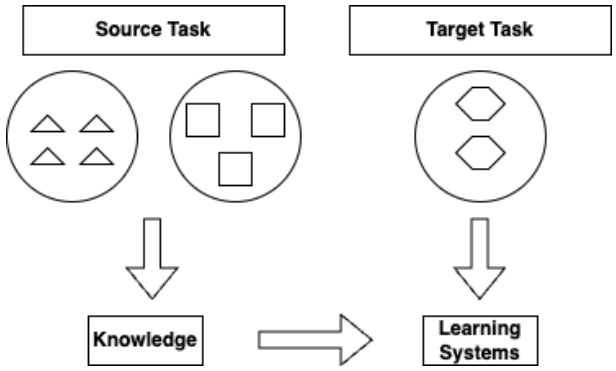


Figure 3.5: Generalization leveraged when the tasks are similar

behind transfer learning is that the experience of solving a source task or domain contains valuable knowledge that can be generalized to help improve the performance of a target model [5]. One reason why transfer learning is efficient is that it enables machine learning models to use the knowledge acquired from a previous task and apply it to a new task, leading to enhanced performance and quicker training durations. It reduces the amount of data needed for training a new model, because the weights are already pre-trained. It also reduces the amount of computational resources that is needed for the training phase of a model. Because of the pre-trained weights, the speed of training is improved, allowing developers and researchers to iterate more quickly and efficient. The pre-trained model leverages the knowledge gained from the previous training, and applies it to the current task. This proves to be useful when there is limited data accessible for the new task.

Another benefit of transfer learning is generalization. A model is first trained on a source task that is often a complex one, with a large dataset. Then, the learned features are fine-tuned on the second or target task, which is often a simpler task. It is important to mention that the generalization highly depends on the similarity between the source task and target task. If they are similar, the knowledge from the first training can be used and the model is likely to perform well, but if the task are different, there is a probability that the model may not perform well.

As mentioned in [26], it is very important to choose how much of a pre-trained model you want to use because in CNNs the features from the early layers are more generic, and the features from the later layers tend to be more specific to the dataset the network has been trained on.

There are three major benefits to look for when using transfer learning:

1. **Higher start** - the model's initial accuracy is higher because it has already learnt some features from the target task.

2. **Higher slope** - the pre-trained model has learnt a broader range of features, enabling it to adapt much more efficiently, resulting in a faster convergence

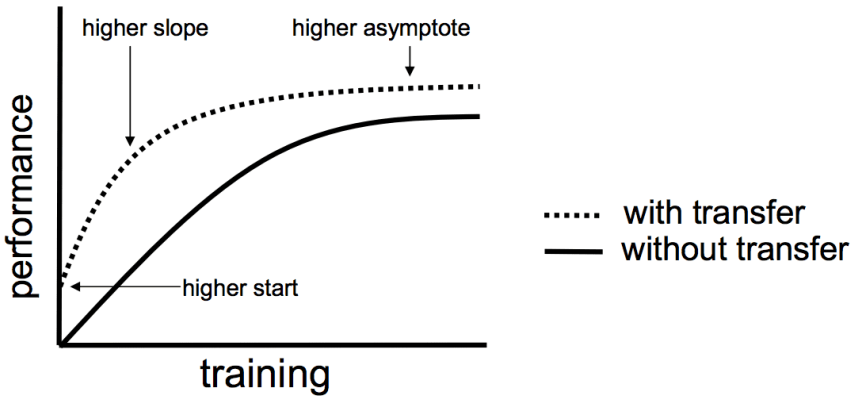


Figure 3.6: Three ways in which transfer learning can enhance the performance of a model [5].

and an improved performance.

3. **Higher asymptote** - the pre-trained model has learnt from its previous training a lot of useful features for the target task, which can improve the overall performance of the model.

### 3.3.1 Techniques for transfer learning

In order to leverage the power of transfer learning, it is necessary to use some techniques to obtain a better performance of the model, ultimately leading to much more reliable predictions.

**Fine-tuning**

This technique involves taking a pre-trained model and training it further on a new dataset. The pre-trained model denotes a deep neural network that has been trained on a large dataset like ImageNet for image classification. By fine-tuning the model, we can adapt it to a specific task such as object detection, segmentation, or classification on a new dataset. There are several types of fine-tuning, among which the most common are [27]:

1. Full Fine-tuning - all the layers of the pre-trained model are fine-tuned on the new task. This requires a large amount of data in order to prevent overfitting.

2. Partial Fine-tuning - only some of the top layers of the pre-trained model are fine-tuned on the new task, while the lower layers are kept frozen. This is useful when the task-specific dataset is small, as it can prevent overfitting.

3. Progressive Fine-tuning - The model undergoes progressive fine-tuning on a series of tasks, with the pre-trained model initially fine-tuned on the first task, and the resulting model subsequently fine-tuned on the second task, and so

forth. This allows the model to learn multiple tasks sequentially while retaining the knowledge learned from previous tasks [28].

**Feature Extraction**

This technique involves employing a pre-trained model as a feature extractor with fixed weights, excluding the final layers of the model. The remaining layers are used to extract high-level features from the input data, which can then be used as input to a new model that is trained for a specific task. This technique is useful when there is not enough task-specific data to fine-tune the entire pre-trained model or when the computational resources required for fine-tuning are not available. Feature extraction can also speed up training by reducing the number of parameters that need to be learned for the new task.

**Domain Adaption**

This technique involves adapting a pre-trained model to a new domain with different statistical properties than the original domain. This is useful when there is a lack of labeled data in the target domain or when collecting new data for the target domain is expensive or impractical. There are several techniques for domain adaptation, including instance reweighting, feature-based adaptation, and adversarial adaptation [5].

### 3.3.2 Pre-trained models in transfer learning

Pre-trained models are machine learning models that have been trained on large datasets to solve a specific task. Pre-trained models are available in various domains, including natural language processing, computer vision and speech recognition. Some popular pre-trained models include VGG [29], Inception [30], ResNet [13], BERT and GPT [31]. The use of pre-trained models has led to significant advances in machine learning and has enabled researchers to tackle new and more complex problems.

**ResNet-50**

ResNet is a type of neural network that utilizes an ”identity shortcut connection” to skip one or more layers, allowing it to be trained on thousands of layers without compromising performance. Due to this, ResNet has become one of the most used pre-trained model for computer vision tasks.

ResNet-50 is a pre-trained convolutional neural network (CNN) architecture that was introduced by Microsoft in 2015. It is a variant of the ResNet family of models, which stands for ”Residual Network”. ResNet-50 contains 50 layers and is deeper than many other CNN architectures used for image classification. ResNet-50 was

developed specifically to tackle the issue of the vanishing gradient, which can arise in deep neural networks. This problem can pose a challenge for the network to grasp and transmit gradients through the layers while it's being trained. The vanishing gradient problem can occur while training a network with gradient-based learning methods and backpropagation. This happens when the partial derivative of the error function with respect to a weight is extremely small, leading to a negligible update of the weight during each iteration of training. As a result, the weight remains effectively unchanged, which can make optimizing the neural network more difficult. To solve this problem, ResNet-50 uses residual connections, which allow information to bypass some layers and be directly passed to the deeper layers of the network. This helps to ensure that gradients are propagated more effectively and improves the network's ability to learn [13].

**Inception V3**

Inception V3 is a deep convolutional neural network architecture that was introduced by researchers at Google in 2015. The main goal of the Inception V3 model is to address the problem of the computational cost of deep neural networks by reducing the number of parameters in the model while maintaining high accuracy.

One of the main features of inception architecture is its use of **inception modules**, which are designed to capture spatial hierarchies of features in images. These modules allow for the network to capture both local and global patterns in the input data, and they are constructed using multiple convolutional layers with different filter sizes. However, this pre-trained model architecture is limited by its use of convolutional filters, which can only capture filters of specific size. Moreover, this architecture does not make use of residual connections, which have been shown to improve the performance of deep neural network architectures [30].

Inception V3 is an enhanced and optimized version of the Inception V1, which was first proposed in [32] in 2014. It also includes all the upgrades of the Inception-ResNet V2 model. It employs a range of techniques to improve the network's ability to adapt to different models. These techniques include improved regularization, factorized convolutions, and residual connections, which collectively make Inception V3 more proficient in handling complex image recognition tasks compared to its predecessor.

In Inception V1, the factorization strategy mainly focused on reducing the computational cost of the network by factorizing large convolutional filters into smaller ones. This was achieved through the use of a 5 x 5 convolutional filter that was factorized into two smaller filters - a 3 x 3 filter and a 1 x 1 filter. Similarly, a 3 x 3 filter was factorized into two 3 x 1 and 1 x 3 filters, which were stacked together. These

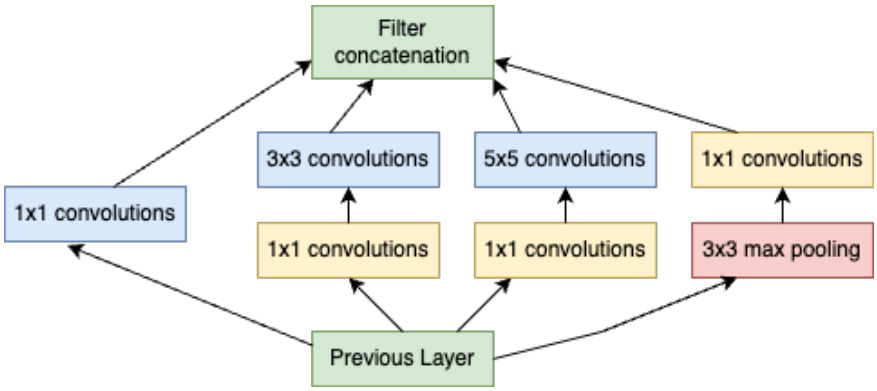


Figure 3.7: The dimension reduction within the inception module

factorizations helped with decreasing the number of parameters and computational cost of the network. In Inception V3, the factorization strategy was further improved with the introduction of efficient factorizations. These factorizations include a combination of depthwise separable convolutions and factorized 1 x 1 convolutions, which are more computationally efficient than traditional convolutions. Depthwise separable convolutions involve a depthwise convolutional operation that utilizes distinct convolutional filters for each input channel, succeeded by a pointwise convolutional operation that applies a 1 x 1 convolutional filter across all channels. This approach significantly reduces the computational cost of the network while maintaining high accuracy.

A fundamental principle behind the design of the Inception architecture is to leverage parallel filters of different sizes to detect features across various spatial scales. To address this problem, the architecture includes a dimension reduction step that reduces the number of input channels for the following filters. This is accomplished through the use of 1 x 1 convolutional filters that can compute linear combinations of the input channels. The use of 1 x 1 convolutional filters for dimension reduction helps to reduce the computational cost of the network by decreasing the number of parameters and computations needed to process the input. Additionally, the authors found that this technique enhances the accuracy of the network by increasing the representational power of the filters [32].

## 3.4 Multi-Task Learning

Multi-task learning (MTL) is an approach in machine learning in which a model is trained to simultaneously perform multiple tasks. Rather than training separate models for each task, a unified model is trained to collectively learn from and perform multiple tasks [33]. The idea is to leverage the shared information among the tasks to enhance the overall performance of the model. This approach proved to

be useful when there is limited data accessible for each task, as sharing information among tasks can enhance the generalization performance of the model. Additionally, multi-task learning can also be used to improve the efficiency of training, as training a single model for multiple tasks is often faster than training separate models for each task. MTL can be used in various scenarios where there are multiple related tasks to be performed, such as multi-class classification and multi-label classification [34]. MTL can be particularly useful when there is a shared representation that can be learned across the tasks, as it can lead to enhanced performance on all tasks compared to learning them separately. Additionally, MTL can also be beneficial when the tasks have a common underlying structure or when there is a need for feature sharing across tasks.

### 3.4.1 Single-task vs. multi-task learning

**Single task learning** (STL) is a type of machine learning approach in which a model is trained to perform a single task, such as predicting the price of a house or classifying an image. The objective of the model in single task learning is to minimize the error between the predicted and true output values. STL algorithms are designed to optimize the performance of the model on a specific task. They typically require a large amount of labeled training data to generalize well to new examples. Some common examples of single task learning algorithms include linear regression, decision trees, and support vector machines. STL can be useful in applications where there is a clear and well-defined task, and the goal is to optimize the performance of the model for that specific task. However, single task learning may not be as effective in situations where there are multiple related tasks, or where there is a need to learn representations that can be shared across multiple tasks. In these cases, multi-task learning may be a more suitable approach.

In contrast, **multi-task learning** involves training a machine learning model to perform multiple related tasks simultaneously, such as predicting the price, location, and size of a house given its features or classifying an image into multiple categories. MTL algorithms seek to learn a shared representation across the tasks, where the model can exploit the commonalities and differences between the tasks to improve performance on each task. The idea is that by jointly learning multiple tasks, the model can leverage the shared structure to improve generalization and reduce overfitting [35]. MTL has gained significant attention in recent years due to its potential to improve performance and efficiency in various areas, including natural language processing, computer vision and healthcare. However, MTL can also be challenging to implement, as it requires identifying related tasks, designing appropriate architectures, and balancing the trade-off between task-specific and shared

representations.

### 3.4.2 Multi-Task Learning in Convolutional Neural Network

In the context of CNNs, multi-task learning involves sharing the early layers of the network across multiple tasks while having separate output layers for each task. This approach can be particularly useful when the tasks share common features in their inputs, as the shared layers can learn these common features and use them to perform better on all tasks.

Multi-task learning in CNNs has been applied to a variety of tasks, including object recognition, facial expression recognition, semantic segmentation and more [11]. By learning multiple related tasks simultaneously, the model can learn more robust features and generalize better to new examples. However, it is important to note that multi-task learning can be challenging, as it requires carefully balancing the objectives of the different tasks and preventing negative interference between them. One of the key benefits of MTL in CNNs is that it can improve the model's ability to generalize to new, unseen data by learning common features across multiple tasks. Additionally, MTL can help to reduce the amount of training data required for each individual task, as the model can leverage data from related tasks to improve its performance.

MTL can be effective because it imposes regularization by demanding that an algorithm performs well on related tasks, which may lead to better regularization than uniformly penalizing all complexities to avoid overfitting. In scenarios where tasks share significant similarities and are slightly under sampled, MTL may prove to be especially useful. However, as we will see below, MTL has been found to be advantageous even for tasks that are not related [36].

Parameter sharing is a technique used in MTL that allows the sharing of model parameters across multiple related tasks. This technique can be implemented in different ways, such as sharing the entire model architecture or sharing only some of the layers in the network. For example, in convolutional neural networks, the early layers that extract low-level features can be shared across tasks, while the later layers that extract task-specific features can be task-specific. It can reduce overfitting by regularizing the model, as it is forced to learn a common representation across tasks. It can also improve the efficiency of training and reduce the amount of data required for each task, as the model can leverage the information learned from other tasks. Additionally, sharing parameters can help transfer knowledge between tasks, which can be especially useful in settings where there is limited data for some tasks [11]. There are two main approaches to parameter sharing: hard parameter sharing and soft parameter sharing.

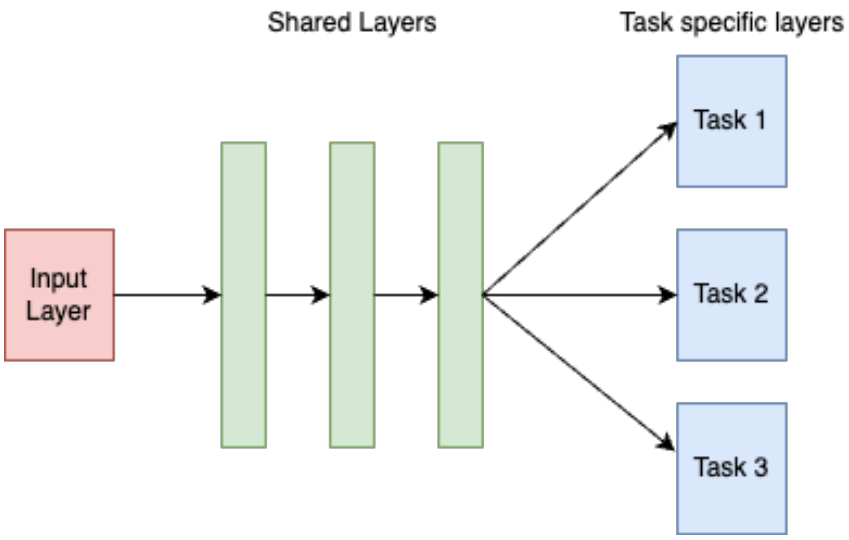


Figure 3.8: Hard parameter sharing in MTL

1. **Hard parameter sharing** involves sharing all of the parameters in the model across all tasks. This means that the same set of parameters is used to learn all the tasks, and the model is trained on all tasks simultaneously. The advantage of hard parameter sharing is that it forces the model to learn a shared representation across all tasks, which can improve generalization performance. However, the downside is that it can be difficult to balance the needs of all tasks, especially if they have different levels of complexity [11].

2. **Soft parameter sharing**, on the other hand, allows for some degree of flexibility in the shared representation. In this approach, the model learns a separate set of task-specific parameters for each task, while still sharing some of the parameters across tasks. The degree of sharing can be controlled by a hyperparameter, allowing the model to learn a balance between task-specific and shared information. The advantage of soft parameter sharing is that it can handle tasks with different levels of complexity and allow for more flexibility in the shared representation [11].

Overall, the choice between hard and soft parameter sharing depends on the nature of the tasks and the trade-off between generalization performance and flexibility. Hard parameter sharing is more restrictive but can lead to better generalization, while soft parameter sharing allows for more flexibility but may require more careful tuning.

# Chapter 4

# Leaf Disease Detection Application

## 4.1 Machine Learning Model

The proposed methods for detecting leaf diseases and classify leaves in this study were implemented using Python programming language [37] and the TensorFlow [38] and Keras framework [39]. TensorFlow is a widely used open-source software library developed by Google and designed for machine learning and artificial intelligence. It provides a variety of tools and functionalities for building, training, and evaluating deep neural networks. Keras is an open-source neural network library which is written in Python and runs on top of Tensorflow. It is designed to provide a user-friendly interface for building and training deep learning models, making it easier and faster to construct complex models.

To facilitate the experiments, the models were trained on Google Colaboratory [40], which is a cloud-based platform that offers free access to a high-performance GPU. The use of Google Colaboratory enabled us to take advantage of its computing power and reduce the computational cost associated with training deep neural networks.

### 4.1.1 Data

For this study, the PlantVillage dataset was used [1]. The dataset contains 61,486 images of 39 unique categories of plant leaf diseases affecting 14 different plant classes. The dataset was augmented with six different techniques, including rotation, scaling, image flipping, noise injection, Gamma correction and PCA color augmentation, resulting in a substantial increase of 19,972 images in size. Each image has a square shape with dimensions of 256 x 256 pixels and features a single leaf against a homogeneous background. Considering this, the model is expected to attain a high accuracy.

A total of 18,000 images were used from the dataset, encompassing 20 distinct

plant leaf diseases affecting six diverse plant classes. Due to the fact that the photos have a homogeneous background and in order to avoid the possible overfitting problem, 800 images have been randomly chosen for training for each category and 100 images for validation. In Fig. 4.1 are displayed some samples from the dataset. For the classification problem, the same dataset has been used, albeit with altered labels that correspond with the plant species. However, it can be seen from Table 4.1 that the diseases are not equally distributed for each plant species, so, the dataset will not be balanced for the classification task and will have a bias that might affect the performance.

| Plant | Training images | Validation images |
|---|---|---|
| Apple | 3200 | 400 |
| Corn | 2400 | 300 |
| Grape | 3200 | 400 |
| Potato | 2400 | 300 |
| Strawberry | 1600 | 200 |
| Tomato | 3200 | 400 |

Table 4.1: The images distribution between the training and validation sets for the classification problem

Figure 4.1: Training images from the PlantVillage dataset [1]

### 4.1.2 Experiments and Results

The experiments were conducted using the datasets mentioned in the previous section. To normalize the pixel values between 0 and 1, the images used for training and validation were resized to the dimensions of 224 x 224 and rescaled. This was achieved using the ImageDataGenerator class from TensorFlow. Mainly, three models have been used.

The first one was a custom model, which was trained from scratch. This model was comprised of three convolutional layers, which constituted the feature extrac-

tor. The first convolutional layer was followed by a MaxPooling layer. A Dropout layer with a dropout rate of 0.35 was applied after the second layer, followed by a MaxPooling layer. The third layer was followed by a Dropout layer, but with the rate 0.5 and then by a MaxPooling layer. The output would be passed to the classification network, which consists of a Flatten layer, followed by a Dense layer with 1024 units. This is followed by a Dropout layer with a rate of 0.3, a Dense layer with 512 units, another Dropout layer with a rate of 0.5, and a Dense layer with 50 units. Finally, there is a Dense layer with six units for the classification task, representing the six plant species, and a Dense layer with 20 units for the disease detection task, representing the 20 different types of diseases. The activation function used in all the convolutional layers and all Dense layers, except the final one, is ReLU. For the final Dense layer, which was used for classification, the Softmax function was utilized.

The second model used a pre-trained network, namely InceptionV3. All the layers from the pre-trained model were frozen. The extracted features were then fed to a classification network which was comprised of a GlobalAveragePooling layer which was then followed by the same classification network as the first model, except the Flatten layer.

The third model employed a Multi-task Learning approach. To address both the classification and disease detection problems, the model initially employs the pre-trained InceptionV3 model with frozen layers to extract features. These features are subsequently passed through a Flatten layer, followed by a Dense layer with 1024 neurons. A Dropout layer with a dropout rate of 0.35 is then applied, and the resulting output is fed into separate classification networks for each task. The same classification network was used for both tasks. It was composed of a Dense layer with 512 units, followed by a Dropout layer with a dropout rate of 0.35. Subsequently, a Dense layer with 50 units was employed, and finally, a Dense layer was utilized for classification. The Dense layer had six units for the plant classification problem and 20 units for the disease detection problem.

The networks that were obtained were trained for 20 epochs using the Adam optimizer with a learning rate of 0.0001. The loss function used in these experiments was the categorical cross-entropy loss function. During the training process, a batch size of 32 was utilized. Both the custom model and the one that uses InceptionV3 were trained separately on the leaf classification task and disease detection task. The results obtained by these models are shown in Fig. 4.2, 4.3 and 4.4.

There were also conducted some experiments with the models that uses the InceptionV3 pre-trained model. The models were trained again but with the last four layers of the InceptionV3 architecture unfrozen. We can see in Fig. 4.5 and Fig. 4.6 that the performances were similar to those where all the layers of the InceptionV3

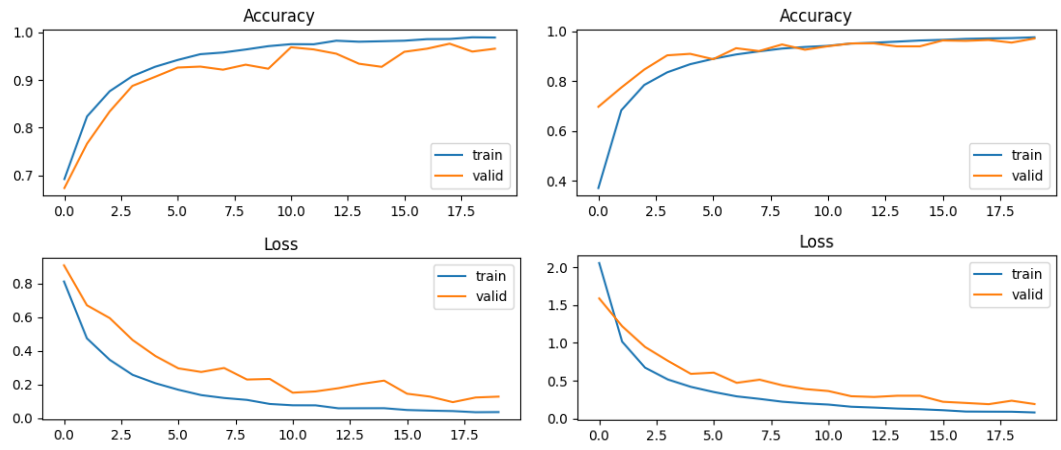


Figure 4.2: The obtained results on the classification task and disease detection task using the custom model (left - the training accuracy and loss; right - the validation accuracy and loss)

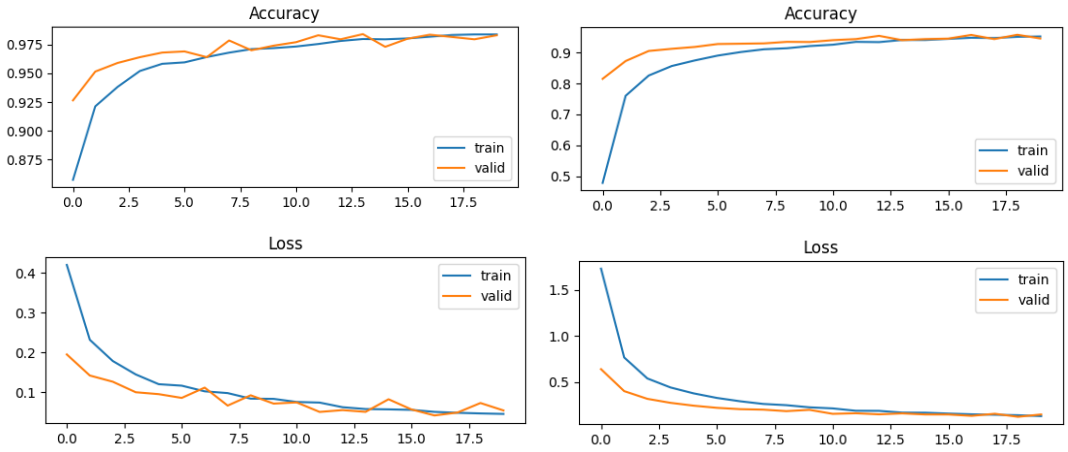


Figure 4.3: The obtained results on the classification task and disease detection task using the InceptionV3 model with all layers frozen (left - the training accuracy and loss; right - the validation accuracy and loss)

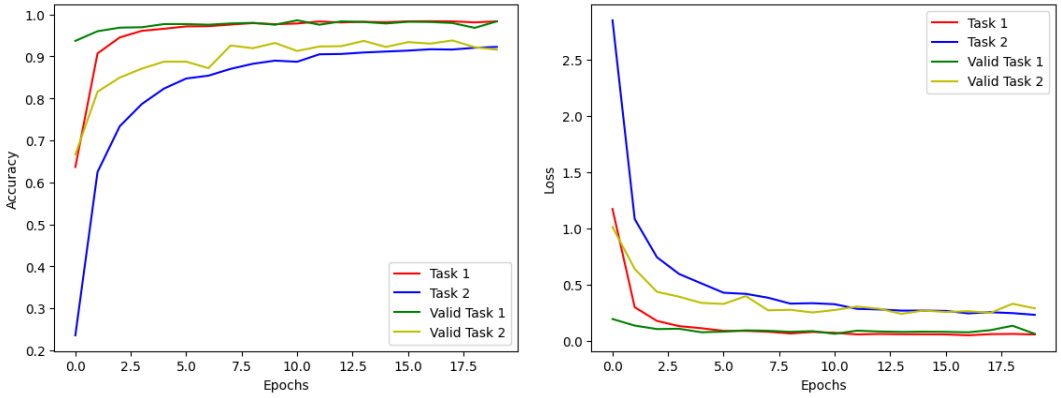


Figure 4.4: The accuracy (left) and loss (right) obtained on the classification task (Task 1) and disease detection (Task 2) by the model that uses the MTL approach and the pretrained Inceptionv3

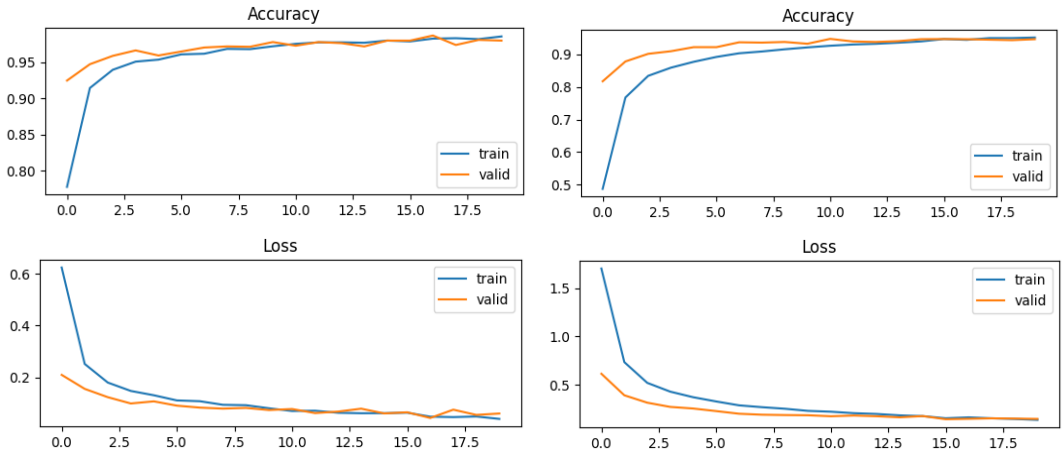


Figure 4.5: The obtained results on the classification task and disease detection task using the InceptionV3 model with the last four layers unfrozen (left - the training accuracy and loss; right - the validation accuracy and loss)

were frozen. From Fig. 4.5 it can bee seen that the models that use the InceptionV3 architecture and not employ the MTL approach have similar performances. From Fig. 4.6, it can be observed that the models that employed a MTL approach have similar performances on the classification task. However, on the disease detection problem, the model that has all the layers frozen from the InceptionV3 performed slightly better.

Another experiment for the MTL approach has been conducted using a custom model which has been trained from scratch. The feature extractor part which is shared for both tasks, the classification and the disease detection, is comprised of four convolutional layers. The first and the forth layers are both followed by a MaxPooling layer. After the second and third layers, a MaxPooling layer is applied, followed by a Dropout layer with dropout rates of 0.35 and 0.5, respectively. The output features are then fed to the same classification network that the previous models that employed a MTL approach. The obtained results of this model are depicted in Fig. 4.7.

It can be observed that the models that yielded the best results for both tasks were the two models that used InceptionV3 with all its layers frozen. For further reference, we will refer to them as Model 1 and Model 2 respectively. The extended results provided by these models are shown in Table 4.3 for classifying the leaves classes for disease detection respectively.

From the extended results it can be observed that the two models have similar performances on the plant classification task. However, for the disease detection task, we can see that Model 1 has the best results. The architecture of the model 1 and model 2 are shown in Fig. 4.8 and Fig. 4.9 respectively.

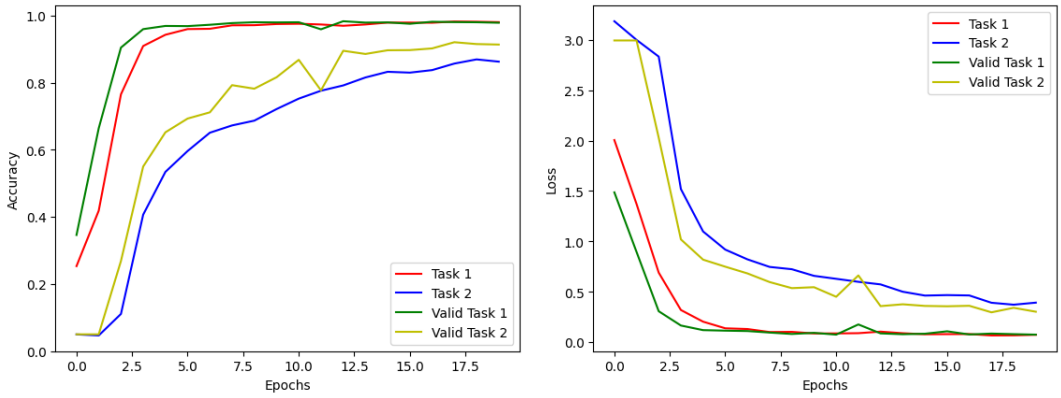


Figure 4.6: The accuracy (left) and loss (right) obtained on the classification task (Task 1) and disease detection (Task 2) by the model that uses the MTL approach and the pre-trained model Inceptionv3 having the last four layers unfrozen.

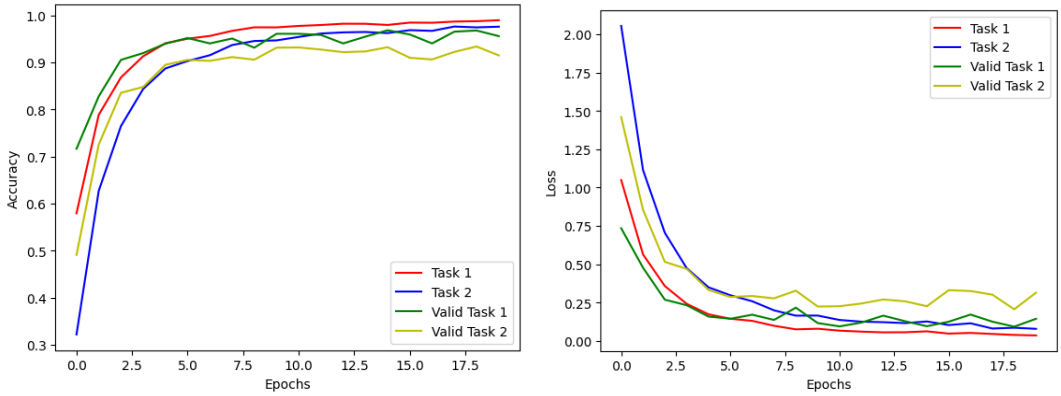


Figure 4.7: The accuracy (left) and loss (right) obtained on the classification task (Task 1) and disease detection (Task 2) by the custom model with MTL

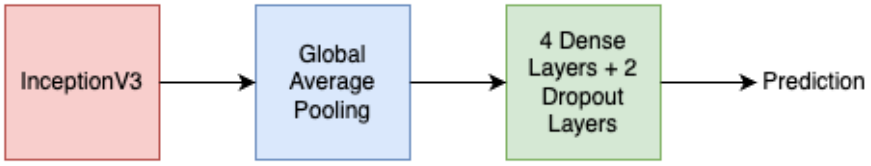


Figure 4.8: The architecture of the Model 1

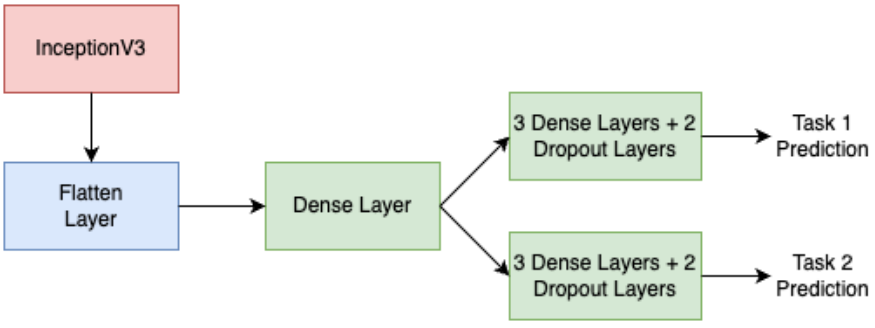


Figure 4.9: The architecture of the Model 2

| Metrics | Model 1 | Model 2 |
|---|---|---|
| Accuracy | 98.37% | 98.35% |
| Loss | 0.04595 | 0.04645 |
| Precision | 98.49% | 99.18% |
| Recall | 98.25% | 97.91% |
| F1-score | 98.36% | 98.54% |

| Metrics | Model 1 | Model 2 |
|---|---|---|
| Accuracy | 98.29% | 98.35% |
| Loss | 0.05467 | 0.09779 |
| Precision | 98.34% | 98.13% |
| Recall | 98.29% | 97.64% |
| F1-score | 98.31% | 97.88% |

Table 4.2: The results obtained for the plant classification problem (training - left, validation - right) by Model 1 and Model 2

| Metrics | Model 1 | Model 2 |
|---|---|---|
| Accuracy | 95.20% | 92.26% |
| Loss | 0.13391 | 0.29598 |
| Precision | 95.88% | 93.69% |
| Recall | 94.53% | 87.90% |
| F1-score | 95.20% | 90.70% |

| Metrics | Model 1 | Model 2 |
|---|---|---|
| Accuracy | 94.55% | 91.60% |
| Loss | 0.15194 | 0.29274 |
| Precision | 95.53% | 94.87% |
| Recall | 94.19% | 90.45% |
| F1-score | 94.85% | 92.60% |

Table 4.3: The results obtained for the disease detection problem (training - left, validation - right) by Model 1 and Model 2

In order to evaluate the performance of these models on unseen data, 20 images for each class from the PlantDoc dataset [41] have been used. The PlantDoc dataset consists of images taken in various environments. The backgrounds in these images tend to be more visually complex and noisy compared to the PlantVillage dataset. The presence of various elements in the background adds additional contextual information to the images. From the results presented in Table 4.4 it can be observed that training only on the Plant Village dataset caused the models to not perform very good on unseen data. Because of the similarity between the images, the models might have learned an excessive amount of information from them. As a consequence, the models became susceptible to overfitting, which hampered their performance on unseen data. Model 2 performed similar to Model 1 on the classification task, however, it did not perform well on the disease detection task. Beside the dataset problem, another one that hampered the performance of Model 2 might be the task prioritization. During the training process the model might have been prioritizing the plant classification task over the disease detection task, leading to insufficient updates to effectively learn the diseases.

| Metrics | Model 1 | Model 2 |
|---|---|---|
| Testing Accuracy | 70.04% | 72.26% |
| Testing Loss | 1.2467 | 1.1459 |

| Metrics | Model 1 | Model 2 |
|---|---|---|
| Testing Accuracy | 51.23% | 23.34% |
| Testing Loss | 2.7953 | 4.2742 |

Table 4.4: The results obtained by models during the testing process using the PlantDoc dataset (plant classification task - left; disease detection task - right

At the end, I aim to perform a comparative analysis between the proposed models and the existing approaches that were presented in the subchapter 2.2. The ar-

ticles proposed models only for the disease detection problem, thus for this comparison we will compare only the results of the models that were trained for this task.

| Model | Dataset | Accuracy |
|---|---|---|
| CNN with 3 convolutional layers [2] | PlantVillage | 91.20% |
| LVQ [12] | Tomato images from PlantVillage | 86.00% |
| R-CNN[3] | Grape images from PlantVillage | 82.96% |
| Model 1 | PlantVillage | 94.55% |
| Model 2 | PlantVillage | 91.60% |

Table 4.5: Comparison between the existing results in the literature and the proposed models

## 4.2 Software Application

The models have been integrated in a web application which consists of both a frontend and a backend. The frontend is responsible for presenting the user interface, while the backend is responsible for generating predictions. An overview of the interaction between the user and the system, an UML use case diagram has been used. Fig. 4.10 reflects all the operations and functionalities accessible to the user. The interaction between the user and the components of the application is described in Fig. 4.11. In the following chapters, I will provide detailed explanations of these components.

### 4.2.1 Backend

The backend of the application was implemented using FastAPI [42], a modern and efficient Python web framework specifically created for developing high-performance APIs. On the server, three classes have been implemented which are shown in Fig 4.12. The PredictionUtils class is a class which contains only static methods. It stores the labels for each species of plant and each disease.

The Prediction class stores data about the predicted species or the predicted disease, and the score of the prediction. It inherits the BaseModel class from the pydantic package, which is a class that provides built-in serialization and deserialization capabilities, allowing easy conversion between Python objects and serialized formats such as JSON or YAML.

The PredictionService class has two fields, namely image and modeltype. The predictclasses method makes the predictions for both tasks based on the value of the modeltype attribute. For the classification task it will generate the top-six predictions and for the disease detection task, it will generate the top three predictions

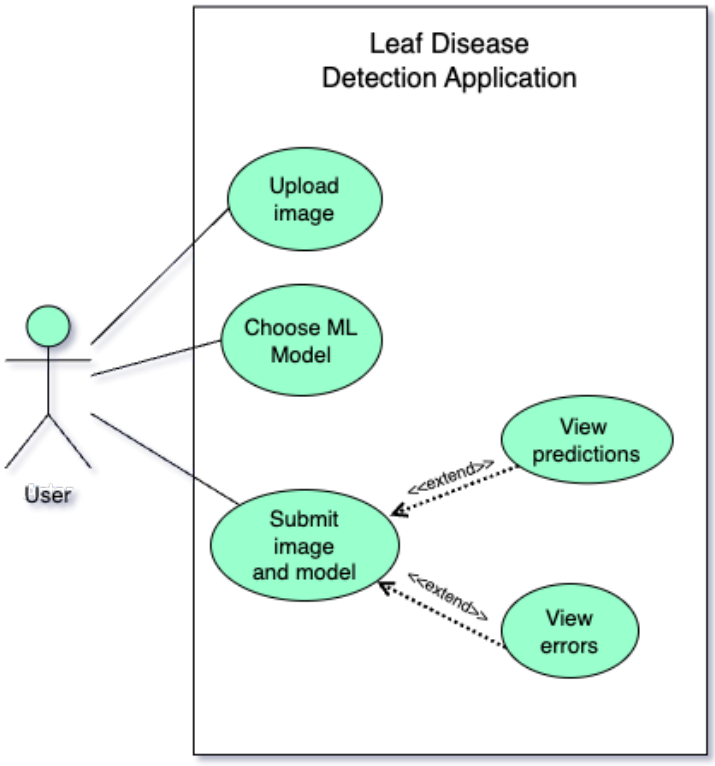


Figure 4.10: Use case diagram of the developed web application

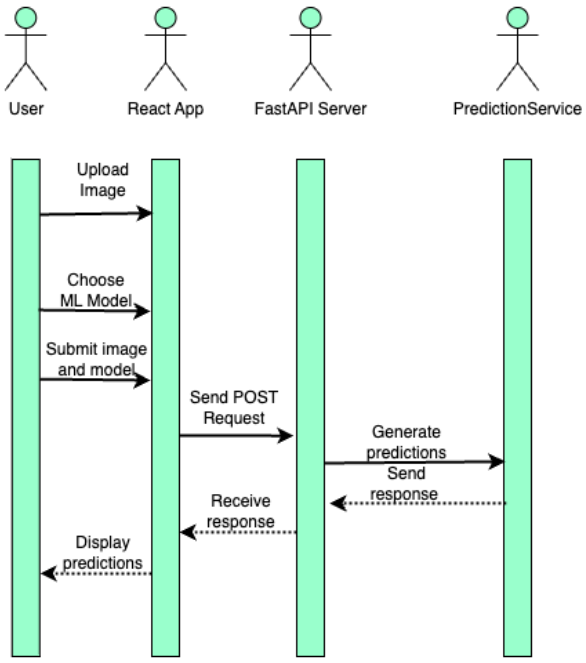


Figure 4.11: The Sequence diagram illustrating the flows of the application

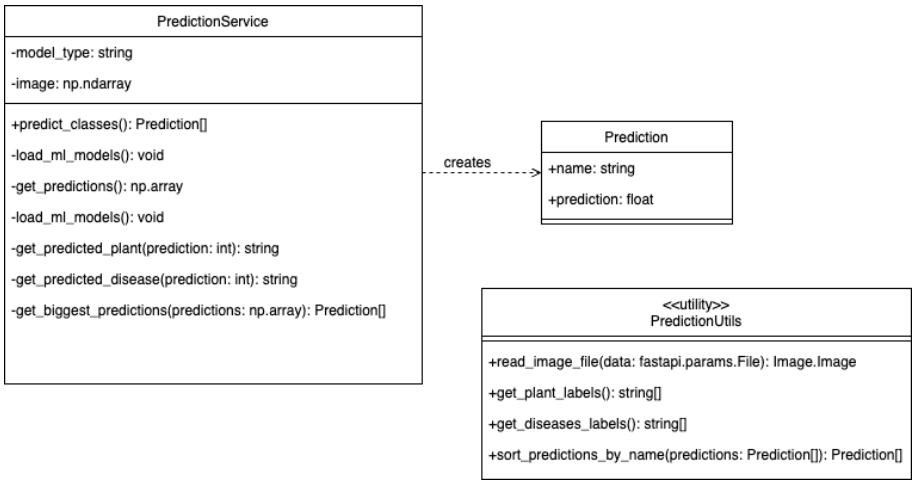


Figure 4.12: PredictionService, PredictionUtils and Prediction classes

with the highest probability for a given image. Before being fed to the model, the image is resized to 224 x 224.

The request for the prediction is initiated via a POST method at the route "/predict". The request requires two fields: "file" which corresponds to the associated image, and "modeltype" to specify the name of the desired model that will be used for generating the prediction. At the end, the response will contain 2 arrays, one for the classification problem, and one for the disease detection problem,

### 4.2.2 Frontend

The front end has been implemented using ReactJS [43], a popular JavaScript library widely utilized for building user interfaces in modern web applications, known for its component-based architecture. The application contains only one component for the main page. The user can upload an image and can also receive a prediction for the uploaded photo. In Fig. 4.13, 4.14 and 4.15 are depicted some screenshots from the main page. The webpage contains an input field for images, a dropdown to select the desired machine learning model and a submit button to make the call to the server. The request sent to the server is of type FormData and includes both the file and the designated model. Once the server responds, the received response triggers the display of two bar charts, visually representing the obtained results. The module recharts was used for the bar charts. Also the elements from the page, such as the Submit button and the other fields were used from the MaterialUI library [44].

As previously stated, the user has the ability to upload an image and select a machine learning model. Upon submitting the necessary information, the application will process the input and present the corresponding predictions to the user.

The user uploads an image and then selects the machine learning model. After that, upon pressing the submit button, the information is sent to the server, which

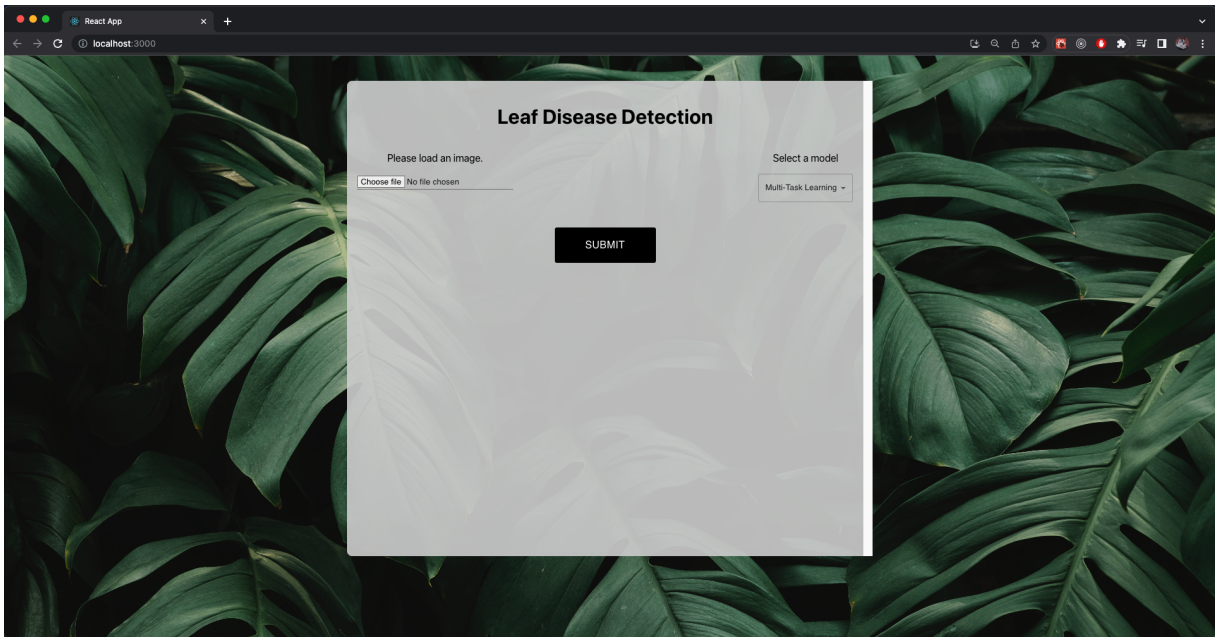


Figure 4.13: Main page of the developed web application

generates the prediction and sends it back React page which displays the response using charts.

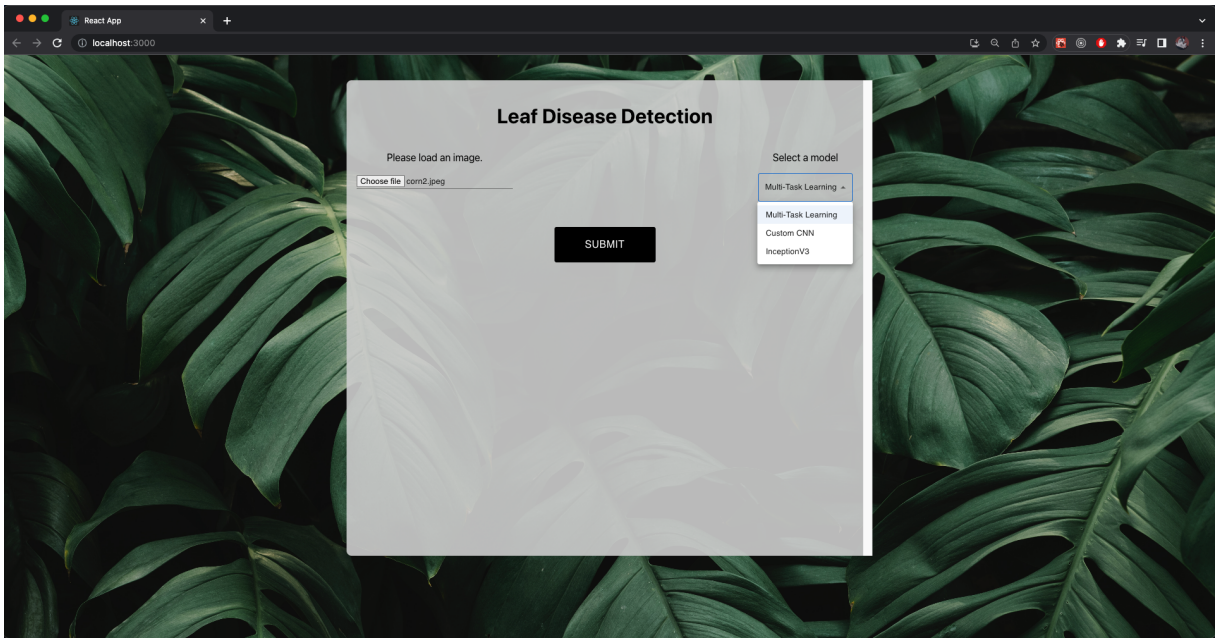


Figure 4.14: Selection of the desire model

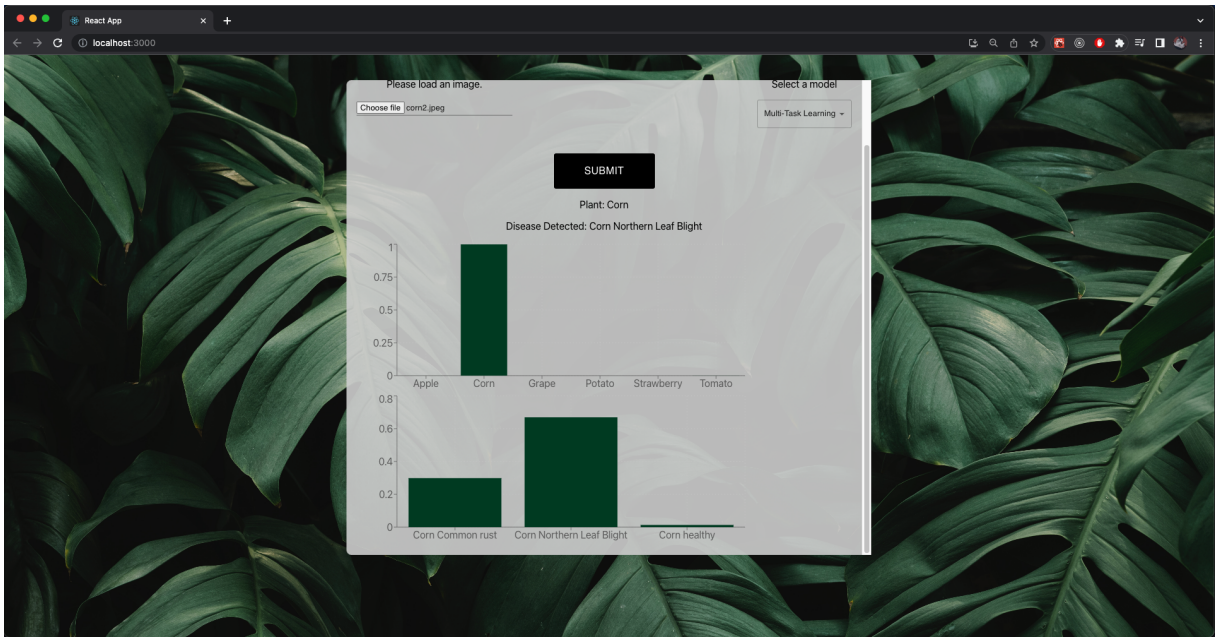


Figure 4.15: The display of the results received from the server

# Chapter 5

# Conclusions and Future Work

Machine learning and computer vision have become quite popular in the research area nowadays due to their ability to train the computer to obtain information from data that is represented in a visual form, such as images and videos, in order to understand it. This thesis proposes a relevant solution for classifying various species of plants and detecting various types of plant diseases from the leaves that are found in images.

Three models have been implemented, one using a pre-trained model, InceptionV3, and two models who have been trained from scratch, but one of them following the multi-task learning approach. Each model has been trained on both tasks separately, the leaf classification task and the disease detection task respectively. Exception from this are the models that were implemented using the MTL approach, because they have been trained on both task simultaneously. The dataset that was used to conduct this research was PlantVillage. Due to the similarity of the images from this dataset, the results these model yielded were quite high, with some of them beating some existing solutions from the literature. The highest accuracy that was obtained on the testing sets during this research was 98.35% for the leaf classification task and 94.55% for the disease detection task.

One of the limitations that was encountered during this research was the lack of hardware resources. The solutions for this was using online notebooks that offered free high-performance GPU. This gave the possibility to train models with different configurations in much less time. Another limitation was the lack of relevant data across the internet. The only relevant dataset for disease detection is PlantVillage, but the downside of this dataset is that the images are homogeneous, making the neural networks not perform well on unseen data. Moreover, in order to make the results much more accurate and avoid the overfitting problem, in the future experiments one improvement would be to create a custom dataset that contains images of plant leaves taken in different environments.

The presented solutions can be implemented in different environments: one of

them would be the web application that was proposed in this thesis, which can help farmers and gardeners combat the appearances of disease right from the early stages, hence saving their crops. Another possible situation in which these models could be integrated would be monitoring the crops all the time using surveillance cameras, in order to detect disease from the early stages, allowing the farmers to save the most out of their crops and minimizing the money loss. The proposed models prove that the artificial intelligence can be of great importance in many areas, such as agriculture, and the fact that it is in a continuous development, better algorithms are expected to show up facilitating further experiments in this domain.

# Bibliography


[1] G. J, ARUN PANDIAN; GOPAL, "Data for: Identification of plant leaf diseases using a 9-layer deep convolutional neural network,"

[2] M. Agarwal, A. Singh, S. Arjaria, A. Sinha, and S. Gupta, "Toled: Tomato leaf disease detection using convolution neural network," *Procedia Computer Science*, vol. 167, pp. 293–301, 2020. International Conference on Computational Intelligence and Data Science.

[3] R. Dwivedi, S. Dey, C. Chakraborty, and S. Tiwari, "Grape disease detection network based on multi-task learning and attention features," *IEEE Sensors Journal*, vol. 21, no. 16, pp. 17573–17580, 2021.

[4] G. Roffo, "Ranking to learn and learning to rank: On the role of ranking in pattern recognition applications," *Computer Vision and Image Understanding*, vol. 172, pp. 1–9, 2018.

[5] E. S. Olivas, J. D. M. Guerrero, M. Martinez-Sober, J. R. Magdalena-Benedito, L. Serrano, *et al.*, *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques*.

[6] A. Lewington, *Plants for People*. London: Natural History Museum, 2007.

[7] P. D. Malanski, B. Dedieu, and S. Schiavi, "Mapping the research domains on work in agriculture. a bibliometric review from scopus database," *Journal of Rural Studies*, vol. 81, pp. 305–314, 2021.

[8] H. Park, J.-S. Eun, and S.-H. Kim, "Image-based disease diagnosing and predicting of the crops through the deep learning mechanism," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 129–131, 2017.

[9] R. E. G. Christine L. Carroll, Colin A. Carter and C.-Y. C. L. Lawell, "Crop disease and agricultural productivity,"

[10] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[11] Y. Zhang and Q. Yang, "An overview of multi-task learning," vol. 5, pp. 30–43, 09 2017.

[12] A. T. Melike Sardogan and Y. Ozen, "Plant leaf disease detection and classification based on cnn with lvq algorithm," 2018.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[14] S. H. Lee, H. Goëau, P. Bonnet, and A. Joly, "Conditional multi-task learning for plant disease identification," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 3320–3327, 2021.

[15] "Digipathos dataset." `https://www.digipathos-rep.cnptia.embrapa.br/`, accessed on 10.05.2023.

[16] "A pl@ntnet dataset for machine learning researchers." `https://plantnet.org/en/2021/03/30/a-plntnet-dataset-for-machine-learning-researchers/`, accessed on 10.05.2023.

[17] "Ipm images for crop diseases." `https://www.ipmimages.org/`, accessed on 10.05.2023.

[18] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[19] A. G. Chigozie Enyinna Nwankpa, Winifred Ijomah and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning,"

[20] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011.

[21] J. Feng and S. Lu, "Performance analysis of various activation functions in artificial neural networks,"

[22] S. H. Han, K. W. Kim, S. Kim, and Y. C. Youn, "Artificial neural network: Understanding the basic concepts without mathematics," *Dementia and Neurocognitive Disorders*, vol. 17, no. 3, pp. 83–89, 2018.

[23] F.-F. Li, J. Johnson, and S. Yeung, "Convolutional neural networks for visual recognition." Online, 2018.

[24] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] S. University, "Convolutional neural networks for visual recognition." `http://cs231n.stanford.edu/`, accessed on 05.04.2023.

[27] R. Feris, "Spottune: Transfer learning through adaptive fine-tuning,"

[28] G. D. H. S. J. K. K. K. R. P. R. H. Andrei A. Rusu, Neil C. Rabinowitz, "Progressive neural networks,"

[29] A. Z. Karen Simonyan, "Very deep convolutional networks for large-scale image recognition,"

[30] C. Szegedy, S. Ioffe, V. Vanhoucke, *et al.*, "Rethinking the inception architecture for computer vision," pp. 2818–2826, IEEE, 2016.

[31] M. B. Y. H. A. J. H. Hasin Rehana, Nur Bengisu Çam, "Evaluation of gpt and bert-based models on identifying protein-protein interactions in biomedical text,"

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[33] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[34] T. P. L. R. Carlo Ciliberto, Youssef Mroueh, "Convex learning of multiple tasks and their structure,"

[35] A. K. M. Z. Ehsan Hajiramezanali, Siamak Zamani Dadaneh and X. Qian, "Bayesian multi-domain learning for cancer subtype discovery from next-generation sequencing count data," *arXiv preprint arXiv:1810.09433*, 2018.

[36] B. Romera-Paredes, H. H. Aung, M. Pontil, and N. Bianchi-Berthouze, "Exploiting unrelated tasks in multi-task learning," 2012.

[37] "Python programming language." https://www.python.org/doc/, accessed on 29.03.2023.

[38] "Tensorflow: An open source machine learning framework for everyone." https://www.tensorflow.org/, accessed on 29.03.2023.

[39] "Keras: The python deep learning library." https://keras.io/, accessed on 1.04.2023.

[40] "Google colaboratory." https://colab.research.google.com/, accessed on 2.04.2023.

[41] P. J. P. K. S. K. N. B. Davinder Singh, Naman Jain, "Plantdoc: A dataset for visual plant disease detection," *arXiv preprint arXiv:1911.10317*, 2019.

[42] "Fast api framework." https://fastapi.tiangolo.com/lo/, accessed on 28.04.2023.

[43] "React: A javascript library to build modern applications." https://react.dev/, accessed on 28.04.2023.

[44] "Materialui: A react component library." https://mui.com/, accessed on 28.04.2023.