الجمهورية الجزائرية الديمقراطية الشعبية وزارة التعليم العالي والبحث العلمي

UNIVERSITE BADJI MOKHTAR – ANNABA
BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــــــــة

Faculty: TECHNOLOGY

Department: ELECTRONIC

Field: SCIENCE AND TECHNOLOGY

Program: TELECOMMUNICATION

Specialization: SYSTEMS TELECOMMUNICATIONS

Thesis

Submitted to obtain the Master's Degree

Theme:

Plants Leafs Diseases Detection Web Application Using Convolutional Neural Networks

Presented by: *Sari Abd Elhalim and Boulfoul Iyad*

Supervisor: BOULMAIZ AMIRA M.C.A University Badji Mokhtar Annaba

Dissent Jury:

S.Lafifi	Professor	*University Badji Mokhtar Annaba*	President
N.Kouadria	Professor	*University Badji Mokhtar Annaba*	Examiner
A.Boulmaiz	MCA	*University Badji Mokhtar Annaba*	Supervisor

Academic Year: 2023/2024

Dedication

First and foremost, we thank ALLAH who has given us reconciliation and energy to successfully complete this work and our studies.

To my precious family members, especially my mother thank you so much for attending my college graduation, it meant so much that you were able to watch me walk across the stage and accept my degree. I couldn't have done it without your encouraging support for the past few years and my whole life. I look forward to continuing to make you proud.

To you my dear friends, I'm forever grateful for your support throughout my college experience. Thanks so much for being my study partner. I wouldn't have wanted to spend those late nights studying with anyone else! Here's too many more years of friendship as we enter this new phase of life together. I can't wait to watch you succeed in your career path.

Dear Professor B.Amira, I want to express how grateful I am for having you as my supervisor. Your interesting, well-explained lectures helped me excel in a very intimidating class. The knowledge you shared allowed me to succeed in other courses and is sure to help me in my future career. Thank you again, and I hope we can keep in contact.

Also special thanks also goes to the examiners professor B.Mohammed and T.Mahmoud for the time they spent on carefully reading this thesis and for their constructive comments.

Warmly, Sari Abd Elhalim and Boulfoul Iyad

Acknowledgement

First and foremost, we would like to take this opportunity to extend our thanks to our supervisors Boulmaiz Amira for creating a stimulating and positive working environment even in this difficult and challenging times, without her help, encouragement, support and guidance it would be impossible for us to complete this work.

A special thanks also goes to the examiners for the time they spent on carefully reading this thesis and for their constructive comments.

Last but not least we would like to thank our family and friends for their constant source of support.

ملخص

يؤدي التدهور في كمية ونوعية الإنتاج إلى خسائر اقتصادية. وبالتالي، فإن التعرف على الأمراض النباتية أمر مهم للغاية. تظهر أعراض المرض في أجزاء مختلفة من النباتات. ومع ذلك، فإن الأوراق هي الأكثر استخدامًا للكشف عن الإصابة بالأمراض. يستخدم العديد من الباحثين تقنيات الرؤية الحاسوبية للكشف عن الأمراض باستخدام صور الأوراق. تقوم دراستنا بتشخيص الأمراض النباتية باستخدام طريقة الشبكة العصبية العميقة (DNN) بناءً على هذه الأعراض المبكرة. تم استخدام عدة نماذج للشبكات العصبية التلافيفية (CNN) مثلAlexNet وVGG16 وResNet بالإضافة إلى نموذج سنقترحه لاحقًا لتحديد 14 فئة تضم عدة امراض. ثم قمنا ببناء واجهة ويب لتشخيص هذه الأمراض باستخدام أحد هذه النماذج.

كلمات مفتاحية: الرؤية الحاسوبية، تشخيص أمراض النبات، الشبكات العصبية التلافيفية

Summary

Degradation in the quantity and quality of production leads to economic losses. Thus, recognition of plant diseases is very important. Disease symptoms appear in different parts of the plants. However, it is the leaves that are most commonly used to detect infection. Many researchers use computer vision techniques to detect diseases using leaf images. Our study diagnoses plant diseases using the deep neural network (DNN) method based on these early symptoms. Several convolutional neural network (CNN) models such as AlexNet, VGG16 and ResNet were used in addition to a model we will propose later to identify 17 classes with 14 diseases. Then we built a web interface for the diagnosis of these diseases using one of these models.

Keywords: Computer vision, Plant diseases diagnosis, CNN.

Résumé

La dégradation de la quantité et de la qualité de la production entraîne des pertes économiques. Il est donc très important de reconnaître les maladies des plantes. Les symptômes des maladies apparaissent dans différentes parties des plantes. Cependant, ce sont les feuilles qui sont le plus souvent utilisées pour détecter les infections. De nombreux chercheurs utilisent des techniques de vision par ordinateur pour détecter les maladies à l'aide d'images de feuilles. Notre étude diagnostique les maladies des plantes à l'aide de la méthode des réseaux neuronaux profonds (RNP) basée sur ces symptômes précoces. Plusieurs modèles de réseaux neuronaux convolutifs (CNN) tels qu'AlexNet, VGG16 et ResNet ont été utilisés en plus d'un modèle que nous proposerons plus tard pour identifier 17 classes avec 14 maladies. Nous avons ensuite construit une interface web pour le diagnostic de ces maladies en utilisant l'un de ces modèles.

Mots-clés: Vision par ordinateur, diagnostic des maladies des plantes, CNN.

Contents

List of tables

List of figures

List of abreviations

AI	*Artificial intelligence.*
D.L.	*Deep learning.*
CNN	*Convolutional neural network.*
DNN	*Deep neural network.*
RGB	*Red Green Blue.*
FCN	*Fully connected network.*
ACP	*Principal component analysis.*
MLP	*Multi Layer Perceptron.*
SVM	*Vector Machine support.*
ReRead	*Rectified Linear Unit Function.*
SGD	*Stochastic gradient descent.*
Adam	*Adaptive Moment Estimation.*
GPU	*Graphics Processing Unit.*
TPR	*True positive rate.*
FPR	*False positive misses.*
TNR	*True negative rate.*
FNR	*False negative rate.*
API	*Application Programming Interface.*
VM	*Virtual Machine*

Introduction general

The occurrence of plant diseases has a negative impact on agriculture production. If plant diseases are not detected promptly, the risk of food insecurity will escalate [1] Pests detection is the basis for effective prevention and control of plant diseases, and they play a vital role in the management and decision-making of agricultural production. In recent times, the identification of plant diseases has become a crucial matter.

Plants infected with a disease typically exhibit noticeable markings or lesions on their leaves, stems, flowers, or fruits. Generally speaking, each disease or pest condition exhibits a distinct visible pattern that can be utilized to precisely diagnose anomalies. Plant leaves are the primary source for identifying plant diseases, and most of the symptoms of diseases may begin to appear on the leaves [2].

In the majority of cases, agricultural and forestry experts are used to identify on-site or farmers identify fruit tree diseases and pests based on experience. This method is not only subjective, but it is also time-consuming, laborious, and inefficient. During the identification process, farmers with less experience may misjudge and use drugs blindly. Quality and output will also lead to environmental pollution, which will lead to unnecessary economic losses. In order to address these obstacles, the investigation of image processing techniques for the detection of plant diseases has emerged as a highly contested research topic. So, researchers usually use leaf pictures to find disease. With advances in machine learning, computer vision applications have been successful. This success led to new ways of learning called deep learning. Many new ways of learning have been used in agriculture, and they are becoming more popular because they work well. Researchers have developed convolutional neural networks (CNNs) that can be used to recognize shapes in images.

The leaves have different appearances and textures that can help you identify the illness. Computers can help solve this problem by seeing things better. Finding plant ailments and preventing crop losses are the focus of this endeavor. This helps to improve production efficiency. Our study employs deep learning to identify plant diseases. We used different computer models, such as AlexNet, VGG16, ResNet. These models were used to create a website to diagnose plant diseases.

This dissertation is divided into four chapters which are structured as follows:

The first chapter: We introduce the treatment procedures that form a system for detecting and classifying ill plants, extending beyond the approaches suggested by the current state of art.

The second chapter: We explain how artificial neural networks evolved into convolutional neural networks used in this work to diagnose plant diseases.

The third chapter: we will explain the work methodology will be explained, as well as an analysis of the performance of different systems implemented for the identification and diagnosis of plant diseases.

The fourth chapter: We shall deliberate on the specifics of the implementation of the deep learning model in a web interface, with the aim of diagnosing plant diseases.

Finally, We end this dissertation with a conclusion and some ideas for future endeavors.

Chapter 1

State of the art on plant disease detection and classification systems

1.1 Introduction

Plant Agricultural pests have a severe impact on both agriculture production and the storage of crops. In order to mitigate the harm caused by agricultural pests, it is imperative to accurately identify the pest category and implement targeted control measures. Therefore, it is imperative to establish an agricultural pest identification system [3].

Plants infected with disease usually show obvious marks or lesions on leaves, stems, flowers, or fruits. Generally speaking, each disease or pest condition exhibits a distinct visible pattern that can be utilized to precisely diagnose anomalies. Typically, the leaves of plants serve as the primary means of identifying plant diseases, and the majority of the symptoms of diseases may manifest themselves on the leaves [4]. In **figure 1.1**, represent leaves diseased - PlantVillage.

In most cases, agricultural and forestry experts are used to find fruit tree diseases or pests on-site. This method is not only subjective, but it also takes a lot of time and effort, and doesn't work well. Farmers who don't have much experience might make mistakes and use drugs without knowing what they are. The way things are made will affect the environment, which will make people lose money. Research into the use of image processing techniques for plant disease recognition has become a hot research topic [4].

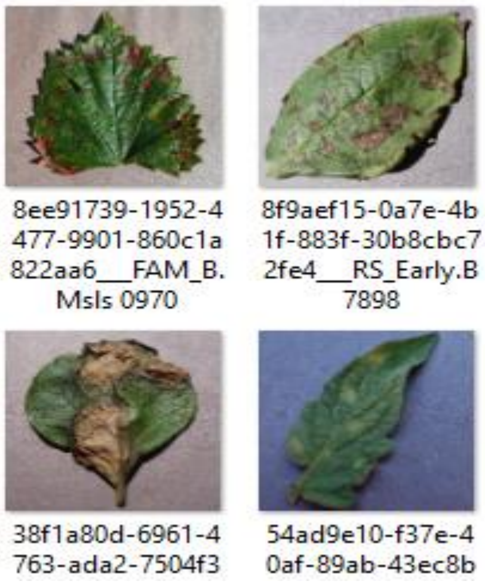

8ee91739-1952-4477-9901-860c1a822aa6___FAM_B.Msls 0970
8f9aef15-0a7e-4b1f-883f-30b8cbc72fe4___RS_Early.B 7898
38f1a80d-6961-4763-ada2-7504f3
54ad9e10-f37e-40af-89ab-43ec8b

Fig. 1.1: Examples of leaves representing diseased plants – PlantVillage

Groupe of data [5]

In this chapter, We shall explore this technique for detecting diseased plants. we introduce the treatment steps that constitute a system for detecting and classifying diseased plants by going beyond traditional methods proposed in the state of the art.

1.2 Plant diseases

Pests and pathogens play a large role in crop losses around the world [6].

Plant diseases can be attributed to a living organism (biotic) or to environmental factors (abiotic), such as hail, spring frosts, weather, chemical burns, and so forth. Since the latter are non-infectious and non-transmissible, they are less dangerous and can be avoided [7]. However, biotic diseases are the most dangerous and cause the greatest damage to crops. They are categorized into three primary categories, namely:

- **Fungal diseases:**
  Fungi or similar organisms are responsible for approximately 85% of plant ailments, and their tiny and light nature makes them capable of traversing the air to infect other plants or trees.
- **Bacterial diseases:**
  Bacteria can spread through insects, splashing water, or other diseased plants or tools.
- **Viral diseases:**

  Viruses are responsible for the rarest plant diseases. Nevertheless, once infected, there is no means of eliminating the virus, and all suspect plants must be eradicated to stop the spread of the disease. Insects are the most prevalent carriers, requiring physical entry into the plant.

  The figure1.2 indicates the 3 types of diseases introduced and the symptoms linked to each one between them :

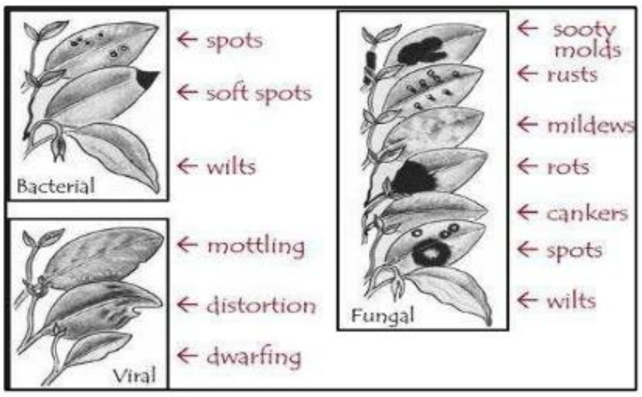


Fig. 1.2: Different types of plant diseases [8]

1.3 Agricultural disease diagnostic systems

Plant disease risk is susceptible to climate change. It is difficult to predict changes in risk under climate change because of the many biological interactions that result in disease. For instance, certain plant diseases arise when the phenology of the plant and the pathogen coincide, as in the case of Fusarium head blight, wherein spores are prepared to infect during the period of wheat flowering. [9].

According to this definition, the primary objective of a plant disease detection system is to accurately detect the occurrence of diseases in advance, employing several methods as illustrated in **Figure 1.3**, in order for producers to make informed choices regarding the use of phytosanitary products. The following outlines the fundamentals, prerequisites, and procedures used to evaluate plant ailments.

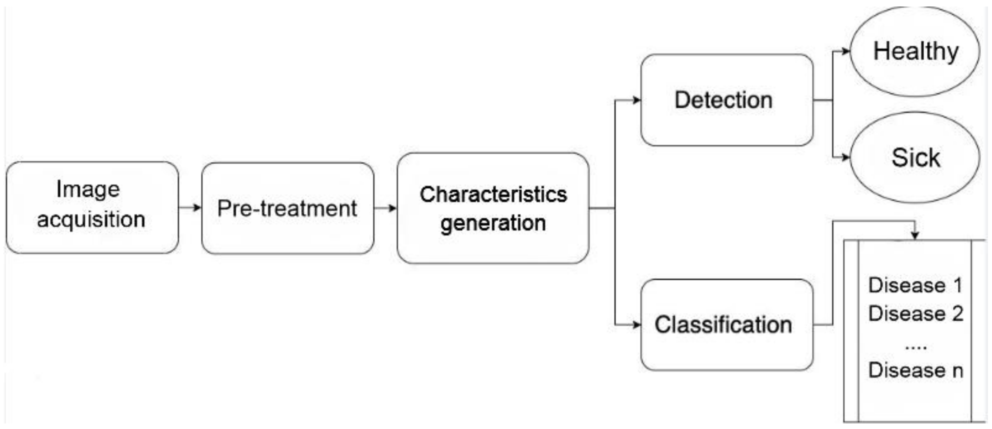


Fig. 1.3: Block diagram of a plant disease diagnosis system

Features of agricultural disease diagnosis system

According to Lucas [10] a prediction system is said to be effective if it presents the characteristics following essential characteristics:

- **Reliability**: Use reliable environmental data. collect climatic factors such as temperature or humidity precisely.
- **Simplicity**: The system needs to have a user-friendly interface, making it easy for farmers to exploit it on a large scale.

- **Importance**: The disease treated must have economic importance on the crop and be fairly sporadic i.e. the possibility of temporary treatment is not applicable.
- **Utility**: The diagnostic system must be useful, that is to say, it allows producers to be relieved of several crop monitoring activities by offering the necessary recommendations for the application of chemicals and control illnesses at the appropriate time.
- **Availability**: The interaction element data must be available in real time (plant types, plant varieties, climate data, etc.)
- **Profitable**: The diagnostic system must be affordable in terms of technical cost-health and disease management available.

1.4 Agricultural disease detection steps

The process of constructing a computer vision system involves several steps before reaching the desired outcome, as illustrated in the **Figure 1.4**.

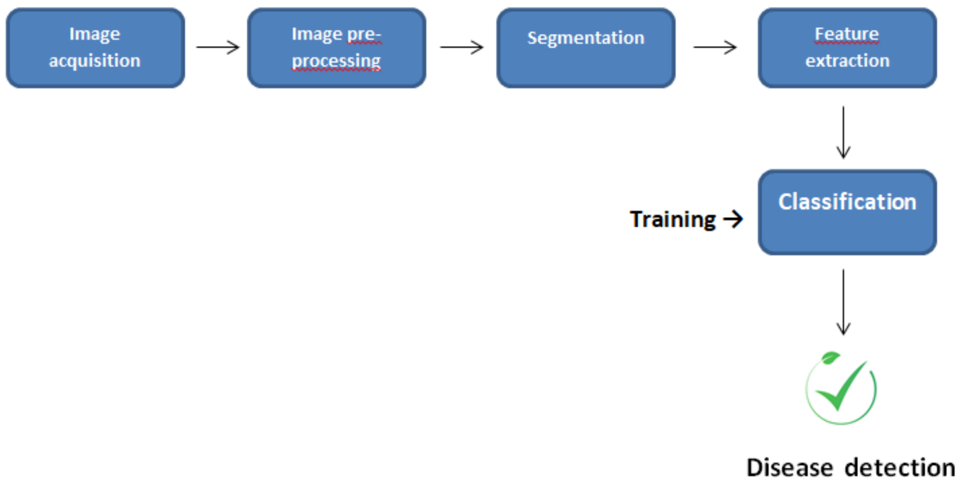


Fig. 1.4: Block diagram showing the steps of agricultural disease detection

1.4.1 Data collection: Image acquisition

As with any computer vision system, this is the initial step.
A variety of devices are used to collect images, including drones and cameras attached to robots. Recent years have seen the use of mobile devices in the creation of mobile applications for the detection of plant ailments. The nature of the control zone and treatment objectives affect the choice of acquisition tool. Smartphones and robots are sufficient for areas with limited area, such as greenhouses. For large areas, drones or satellite imagery are recommended.

1.4.2 Image pre-processing

The term "pre-processing" is commonly used to refer to operations that involve images at the lowest level of abstraction, where both the input and output are intensity images. These iconic images are of the same type as the original data captured by the sensor, with an intensity image usually represented by a matrix of image function values (brightness's) The purpose of pre-processing is to improve the image data by suppressing unwanted distortions or enhancing some image features. Major datasets typically collect images in real-time, frequently accompanied by inaccurate data. The images are pre-processed in order to enhance the computational precision of the plant disease detection system before they are extracted. Applying pre-processing actions, such as resizing and cropping, also reduces the duration of the process [11].
The pre-processing operations can be summarized as follows:

(a) Image normalisation
When the characterization method used generates descriptors that depend on the image size, it is important to standardize the single size.
The process of normalizing data is imperative as classification methodologies necessitate a uniform amount of information. It is also possible to reduce the size of the data in order to simplify processing. such as shown in the **figure1.5** :

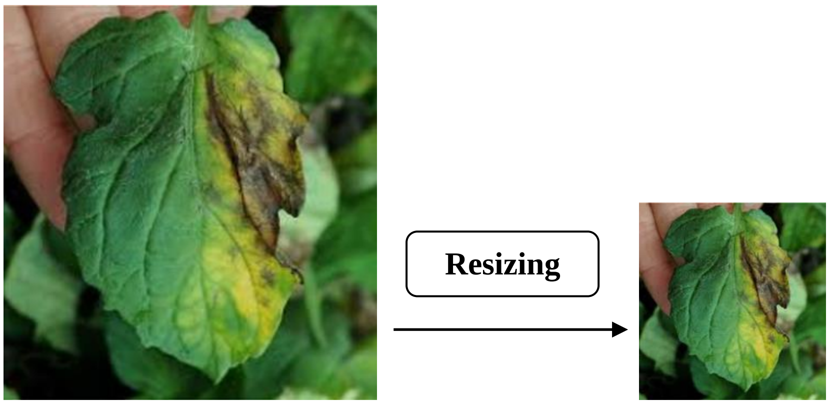


Fig. 1.5: Resizing an image

(b) **Noise cancellation**
Using low-pass filters removes noise from the scene, shooting conditions, or the camera's sensor. The figure1.6 shows an example of a filter called a Gaussian that removes unwanted noise without affecting the details in the shape.

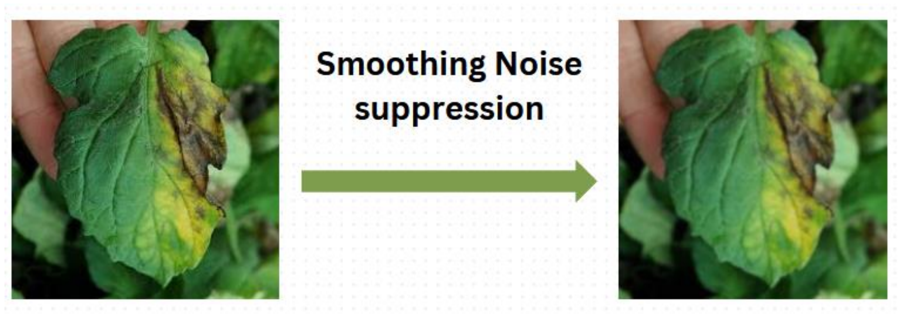


Fig. 1.6: Smoothing an image

(c) Edge detection

Edge detection refers to the process of identifying and locating sharp discontinuities in an image[12].

Edge detection helps make a leaf look better by changing its shape depending on how healthy the images are. Computer vision uses edge detectors called convolutional filtering to show areas with a lot of variation in intensity.

We make a distinction here of the Sobel, Laplacian filters [13].

For illustration purposes, the figure1.7 shows the detection of edges by the related Sobel and Laplacian 4 filters.

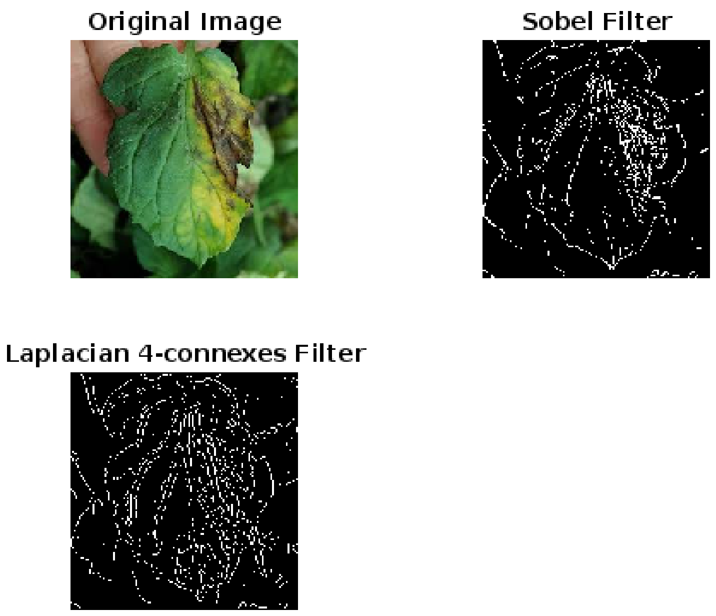


Fig. 1.7: Detecting image edges

(d) Image Segmentation

To locate areas of interest, the image is divided by segmentation. The goal is to pinpoint the area with atypical features: The simplified image representation makes it easier to analyze and more effective for distinguishing between infected and uninfected areas. The figure 1.8 illustrates two instances of a plant with a disease.

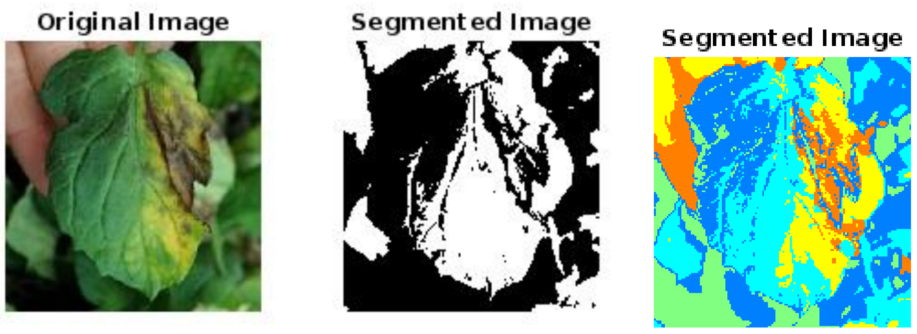


Fig. 1.8: Examples of images before and after segmentation [14]

1.4.3 Characteristics generation

Feature extraction techniques are applied to get features that will be useful in classifying and recognizing images.

They are also useful in various image processing applications, such as character recognition [15].

To extract abnormal tissues from plants leafs consists of three steps , presented in block diagram bellow [16].

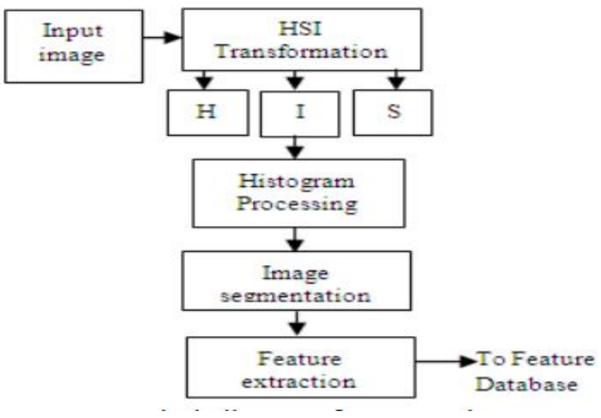


Fig. 1.9: Block diagram of Image analyzer [16]

STEP1: transform defected image to HSI color space.

STEP2: Examining the intensity image histogram.

STEP3: applying thresholds helps to adjust the image intensity. [16]

Image segmentation is the process of dividing an image into regions, subregions, or objects with the same characteristics. The two main properties used for image segmentation are discontinuity and similarity. Depending on these attributes, the image segmentation can be categorized into two distinct categories: edged-based segmentation and region-based segmentation [17].

Feature extraction is a fast and efficient way to use features learned by a pre-trained neural network. It propagates the input image to a layer of our own that defines it as the output feature. Using a pre-trained network to extract features is easy. The layer used to consider might change, but the process is still the same. An image is started as an input and its size is set by the pre-trained default input shape. The same image is sent through the network [18].

Computer vision deals with the automatic extraction, evaluation, and comprehension of useful data from a single image or a series of images. Automatic image classification systems have been developed using Convolutional Neural Networks [19].

Convolutional Neural Networks (CNN) are popular because they are widely used for unstructured data classification [20].

1.4.4 Disease Classification and Detection

Plant disease detection using computer vision and image processing involves the classification step. Given its importance in disease detection, the performance of this phase depends on previous stages, such as data acquisition, pre-processing stage, segmentation of the infected area, and the final feature extraction and selection.

Classic classifiers such as K-nearest neighbors and Bayesian classifier were discovered in early research in smart agriculture. Recent years have seen the widespread adoption of statistical vector machines and neural networks. As a result, the algorithms used in the current state of the art can be summarized as follows :

K Nearest Neighbor Classifier (KPVV)

K-nearest neighbor is an instance-based learning method. A model is built based on the training samples associated with the nearest neighbor class with a distance function and a class selection function, without requiring a learning phase. In [21], the authors conducted an evaluation of the efficacy of this classifier in detecting diseases caused by fungi in corn leaves.

Support Vector Machines (SVM)

SVM is a new type of machine learning that uses statistics. Because of its good reputation and higher accuracy, it has become the research focus of the machine learning community. This paper explains the basics of support vector machine and how it is used to classify things. How well an algorithm works and what the future holds for support vector machines in classification. Finally the prospect of support vector machines in classification applications. SVM has been extensively utilized in numerous research studies as an automated detection system for plants that are exhibiting disease. [21].

Bayes classifiers

The Bayes classifier is favored in pattern recognition for its exceptional performance in minimizing classification errors. It operates on the premise of comprehensive class knowledge, utilizing prior probabilities and class-specific patterns to predict future occurrences and assign labels to test patterns. Leveraging the Bayes theorem, it transforms prior probabilities into future probabilities based on pattern characteristics, using likelihood values as key indicators of predictive accuracy [22].

Random Forest

Random forests are groups of trees that work together, each using random values. Adding more trees reduces errors. They rely on strong individual trees and connections. Using random features for node division works better than Adaboost. Internal estimates help decide feature importance and response to increasing features, applying to regression as well [23].

Artificial neural networks

ANNs are a part of machine learning in the field of artificial intelligence. The goal is to make machine learning systems that are affected by the electrical activity of the brain. [24]. Digital image processing and artificial neural networks are important tools for keeping track of plants' health [25].

1.5 Related works

This section examines the latest research on the deep learning application to the plant disease detection, particularly the work done using the PlantVillage dataset [26], the results are summarized in the table1.1.

The authors of [27] suggested a method for detecting plant pathogens. There were 800 images of cucumber leaves that represented two distinct diseases and also healthy ones. The authors employed their own version of the CNN algorithm. A four-fold cross-validation approach led to a maximum classification precision of 94.9% for the proposed algorithm.

The same authors did another study [28] to find out about seven different types of viruses in cucumbers. They used a dataset of 7520 images that included viral diseases and healthy leaves. Classifiers were 82.3% accurate when tested four times to make sure they were correct.

A CNN helped the authors of [29] distinguish 13 distinct types of plant ailments from healthy leaves. More than 3000 original images were used to represent 13 different diseases in different crops and two further classes for healthy leaves and background images. The authors employed a CNN CaffeNet model that had been trained to achieve an accuracy range of 91% to 98%, for testing distinct classes, and an overall precision of 96.3%.

The people who wrote [30] used a computer program called LeNet to find two different types of diseases in pictures of banana leaves that were taken for the Plant Village project. The data had 3700 pictures, and the model was able to guess 99.72% of them. Deep learning was used to detect apple black rot disease severity [31] using the PlantVillage dataset. The portion of the data set they utilized contains more than 150 images, arranged in four different levels of severity. Four different models were used to compare the results of the CNN model (VGG16, VGG19, Inception-V3 andResNet50), which were fine-tuned and trained from scratch. The VGG16 model was adjusted to achieve an accuracy of 90.4%

There are many studies and articles about how to classify agricultural diseases. We will list them all in the table below :

Authors	Dataset			Model	Precision	Year
	Plant type	Number of classes	Number of images			
[32]	Maize	9	500	GoogLeNet, Cifar10	98.9%, 98.8%	2018
[33]	Tomato	6	13262	AlexNet, VGG1 6.	97.49%, 97.29%	2018
[30]	Banana	3	3700	LeNet	99.72%	2017
[31]	Apple	4	2086	VGG16, VGG19, Inceptio n- v3,and ResNet 50	90.4%with VGG 16 model.	2017
[28]	Cucumber	2	7520	CNN	82.3%	2017
[29]	crop	13	3000	CaffeNet	range of 91% to 98% overall precision of 96.3%	2016

Tab. 1.1: Summary of articles on using deep learning in detection plant diseases

Deep learning models were used to classify diseases from images, a few were used to segment the diseased areas, and one estimated the degree of the illness. The appropriate treatment for that case depends on classifying whether the plant is diseased and identifying that disease.

1.6 Conclusion

In this chapter, We talked about the general principles of plant ailment detection techniques and the various phases of the adapted systems, acquisition, pre-processing, segmentation, etc. We introduce the different classifiers explored in the study for the detection of plant diseases. Our method for detecting and classifying plant diseases is described in the next section.

The early detection of plants (before they start showing signs of illness) could provide valuable information for the implementation of effective pest control strategies and disease prevention measures to prevent the emergence and spread of infectious diseases. One of the solutions represents deep learning models that could potentially be integrated into a web interface.

In contemporary times, it is imperative to implement appropriate management strategies, such as the utilization of fungicides and the utilization of specific chemical agents to combat diseases. These strategies enable us to apply pesticides with instant information on plant health. This improves disease control and productivity.

Chapter 2

Convolutional Neural Networks

2.1 Introduction

The use of Artificial Intelligence (AI) has received a lot of attention in the media recently. This has made many people excited, but also scared. Some of these fears are based on unrealistic or unrealistic ideas about what AI can do. This strong interest in AI is largely linked to major technological advances that have greatly improved computer performance in many fields, such as automatic speech recognition and computer vision. These advances have opened up a lot of possibilities for AI in various forms, such as applications, robots, chatbots, etc. It's interesting that many different areas are involved, like industry, health, agriculture, finance, banking, insurance, and transportation. AI will become more important in organizations and systems that make things. The fields of application for AI in these sectors continue to grow. In this situation, people usually think of AI as a group of tools that can help with many things like making work faster, improving efficiency, and making work easier. Some even argue that by transferring these tasks to AI systems, it will be possible to redirect the work of employees to higher "added value" activities. Many reports and books have talked about the (potentially) positive and negative impacts of AI on work and employment.[1]

This chapter presents a theoretical overview of the evolution of artificial neural networks towards convolutional neural networks used in this work for diagnosis plant diseases.

2.2 Artificial Intelligence

Recent advancements have been made in artificial intelligence and machine learning have radically changed the way we process, analyze and manipulate images. This is largely due to the craze for deep machine learning, considered the new frontier of artificial intelligence, in which the most representative and discriminating features are acquired end-to-end. Convolutional neural networks have produced excellent results in segmentation, classification, detection and identification tasks. [2] [3]. A CNN comprises of two processing blocks, namely a convolutional block that generates features and a prediction block that executes the classification (or detection) step [4].

In this work, we propose to study CNNs as an end-to-end system and as a feature extractor. To make this system, we use different computer vision designs like VGG-16, AlexNet and ResNet.

2.3 Concepts on artificial neural networks

Neural networks have become popular recently and are being used successfully in many different areas like finance, medicine, engineering, geology, physics, and biology. The people are excited because these networks try to show what the human brain can do. From a statistical point of view, neural networks are interesting because they can be used in prediction and classification problems [5]. Now, ANNs can identify patterns between input data sets and target values. ANNs can be used to predict the outcome of new independent input data. ANNs mimic how humans learn and can handle problems with complicated data, even if it's not clear or clear. So they are great for modeling agricultural data that can be complicated and not always straight forward [5].

2.3.1 From biological neuron to artificial neuron

Artificial neural networks are like biological neural networks, which are systems of very simple processors and connections between them. Artificial models try to use the same ways that humans organize things. An artificial neural network (ANN) sees nodes like fake neurons. An artificial neuron is designed as a computational model of real neurons. Artificial neural networks are mostly used for processing information, like making models of real brains and studying how animals and machines behave. They can also be used in engineering to recognize patterns, predict what will happen, and compress data. These networks are mostly made up of inputs and weights. The weights add up the inputs and are figured out by a math trick that tells if a neuron is activated. Another function usually calculates the output of an artificial neuron. The neuron only adds up its inputs and multiplies its output by the weights [6]. Fig. 2.1: shows the biological neuron.

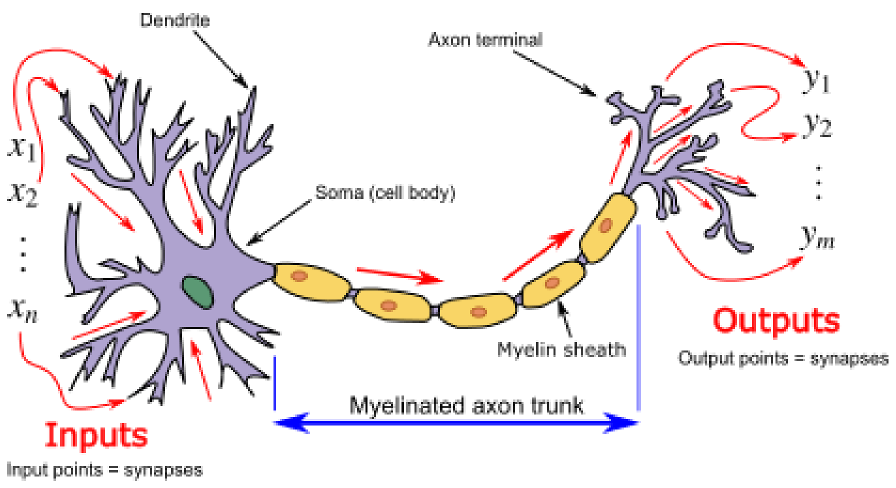


Fig. 2.1: Biological neuron [7]

2.3.2 Perceptron

Artificial neural network (ANN) is a type of machine learning technique. The name is inspired by the connections between neurons in the human brain. ANNs use data to learn and find hidden connections between things, even if they don't explicitly explain how they work.

They have many different shapes, but they all have one thing in common: the neuron. Neurons are parts that work together and have different connections. In every neuron, input has a weight and a bias: data goes to the next level through an activation function. An artificial neural network is made up of many neurons that are connected together to solve linear or non-linear problems [8].

The figure 2.2 shows the structure of an artificial neuron.

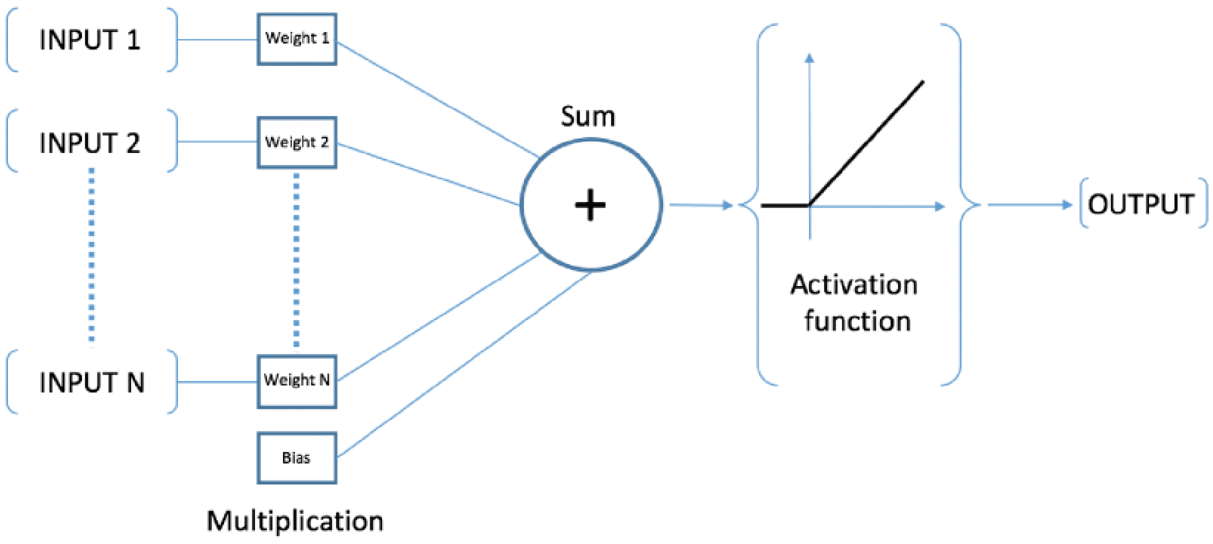


Fig. 2.2: Structure of an artificial neuron [8]

which proceeds by summation weighted by its input vector:

$$I = (\text{Input}\mathbf{1}; \text{Input}\mathbf{2}; \ldots ; \text{Input}\mathbf{N}) \in \Re^M \tag{2.1}$$

And the synaptic weight vector:

$$W = (\text{Weight}\mathbf{1}; \text{Weight}\mathbf{2}; \ldots ; \text{Weight}\mathbf{N}) \in \Re^M \tag{2.2}$$

An adder (sum) computes the linear combination of inputs and their weights.

An activation function controls the neuron's output intensity, collectively mimicking biological neuron behaviour for effective information processing in artificial neural networks [9].

2.3.3 The multilayer perceptron

The Rumelhart introduced the multilayer perceptron in 1986, extending the previous single-layer perceptron. The structure consists of three types of layers, with each neuron being linked to all the neurons in the next layer, resulting in a fully connected network, as shown in figure2.3. The dimensions of the components used to carry out the task determine the nature of the issue to be tackled. In other words, the number of neurons in the input layer determines the dimension of the processed information, whereas the number of neurons in the output layer determines the number of classes. As for the hidden layers, their numbers and the number of neurons that constitute them are design problems. The model's ability to grasp intricate decision boundaries is limited by the absence of numerous hidden neurons. Over fitting the model results in too many neurons reducing its generalization. Thus, a rigorous experimentation is required to establish a satisfactory balance between the number of omitted nodes and the network's capacity for generalization [10].

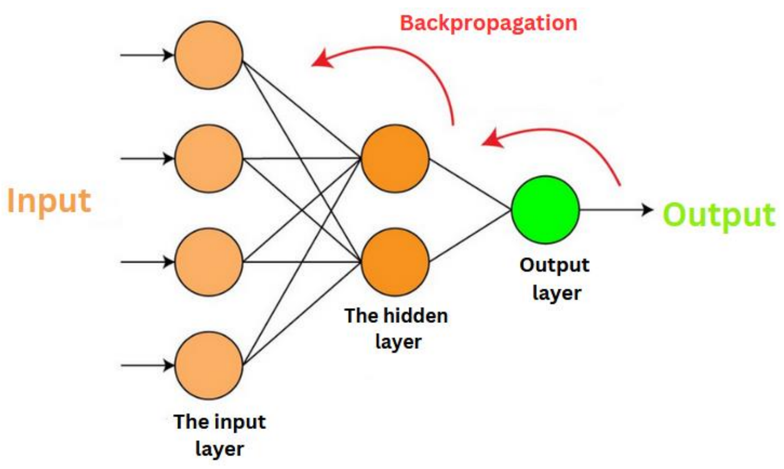


Fig. 2.3: Schematic representation of an MLP with a single hidden layer

The PMC was trained by the error gradient back propagation algorithm proposed by Rumelhart [10] , This allows information to flow in the opposite direction in the network in order to calculate the gradient. The mean square error E between the estimated output vector y and the real output vector yr is used to adjust the synaptic weights of all the neurons of the different layers by the following equation:

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(y_r^i - y^i\right)^2 \tag{2.3}$$

The synaptic weights are modified such that :

$$w(t+1) = w(t) + \Delta w(t+1)$$

$$\Delta w(t+1) = -\alpha \frac{\partial E}{\partial w} + \mu \Delta w(t) \quad (2.4)$$

t : current iteration (corresponding to the passage of data through the network).

Δw(): Change in weight with each iteration.

$\boldsymbol{\frac{\partial E}{\partial w}}$: error gradient with respect to weight w.

It should be noted that the objective is to minimize a cost function represented by the mean square error E down to its local minimum, which is called descent of the gradient.

The parameter α shows how fast or slow the gradient descends in the direction of the local minimum, which determines how fast or slow the local minimum is approached. The picture shows how much the learning step weaving is important and how much it affects things.

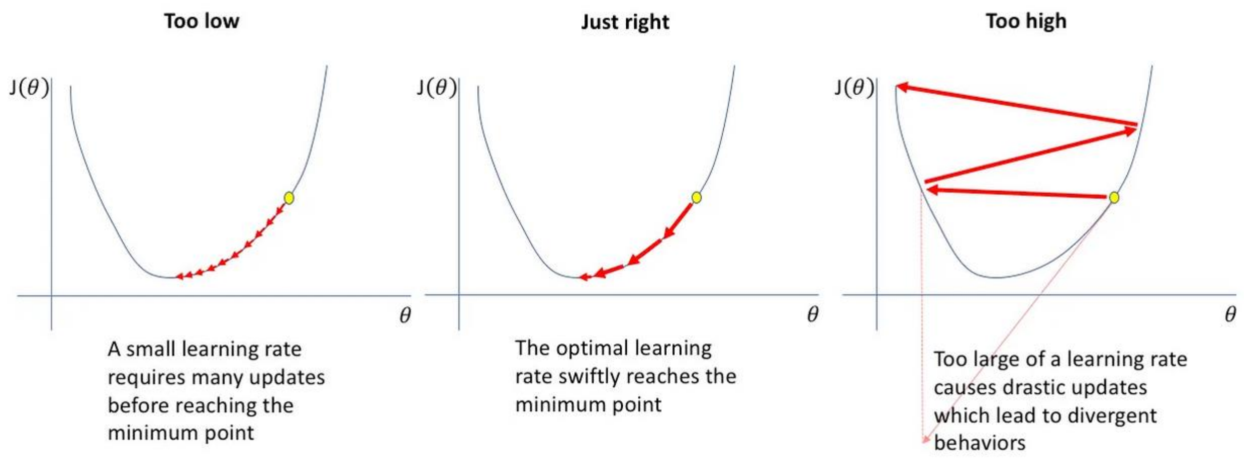

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Fig. 2.4: the impact of the learning step weaving [11]

The parameter $\boldsymbol{\mu}$ momentum lets us keep track of the last update of the synaptic weights so that we can use it in the current update (iteration t+1)

2.4 CNN Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep learning architecture that is inspired by the natural visual perception mechanism of living creatures. In 1959, Hubel and Wiesel [12] discovered that cells in animals' eyes can detect light in their eyes.

CNN is considered to be the predecessor of CNN. In 1990, LeCun [13] and his team wrote the first paper about CNN. They improved it later in [14]. It can show the original picture well and recognize patterns right away without any extra work. Zhang et al. [15] used a shift-invariant artificial neural network (SIANN) to recognize characters in an image.

CNN makes it easier to extract features manually. Actually, it takes out important parts of the picture and gives them different weights and biases to make them stand out. Each input image will go through two blocks: the convolution block and the classification block.

Figure 2.5 represent a simplified illustration of a neural network based on a fixed-size image of wheat spike part. Each convolutional layer automatically extracts useful features, such as edges or corners, and outputs a number of feature maps. Pooling actions reduce the dimensions of the feature maps to boost effectiveness. The number of data representations is progressively augmented within the network to enhance classification precision. Standard neural network layers are used to output probabilities for each class. [16]

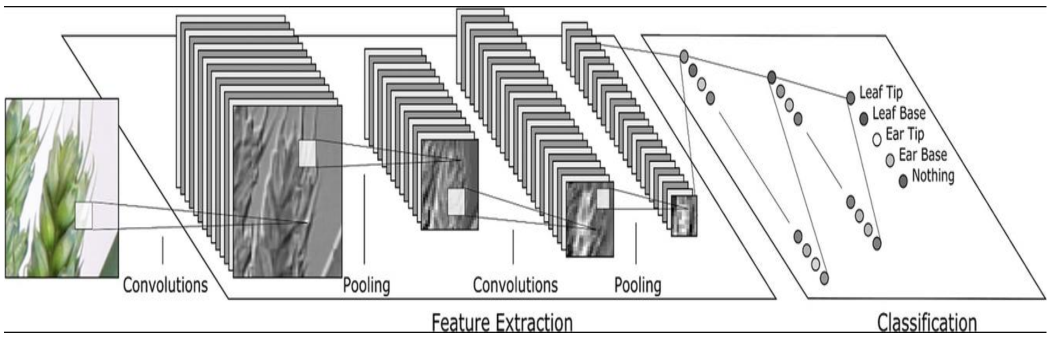


Fig. 2.5: Illustration of the CNN architecture

2.4.1 Convolutional Layer

The convolution layer is made up of multiple feature maps that are made by combining the convolution kernel with the input signal. Each convolution kernel is a weight matrix, which can be a 3×3 or 5×5 matrix for a two-dimensional (2D) single channel image. figure2.6 illustrates an example of the 2D convolution.

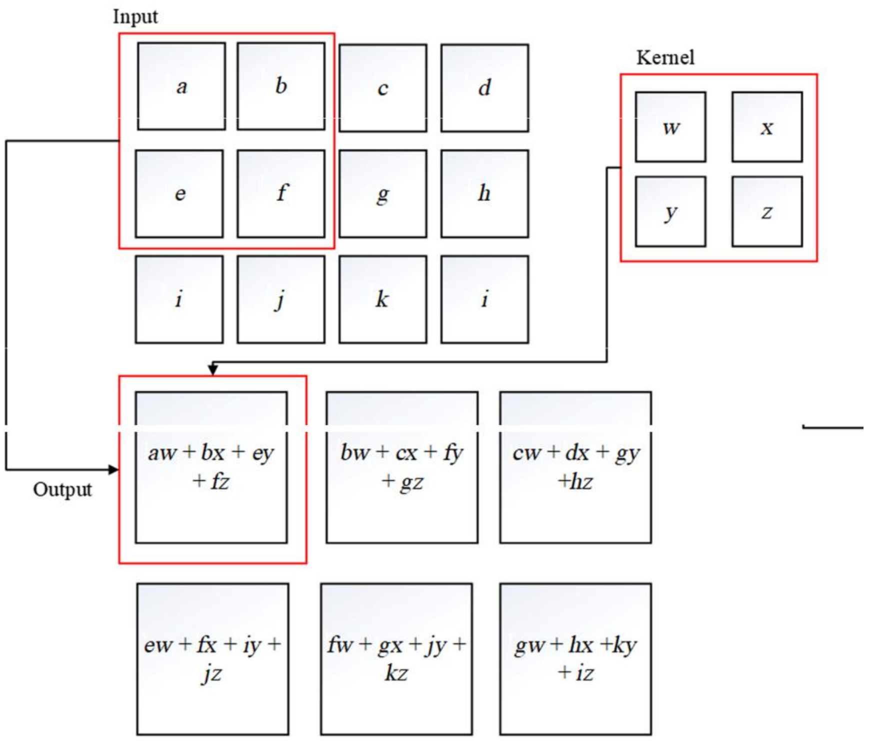


Fig. 2.6: Example of the 2D convolution.

The convolution procedure permits the processing of variable-sized inputs by employing convolution kernels, and diverse input characteristics are uncovered by the convolution procedure in the convolution layer. Lower-level features such as edges, end points and corners are extracted from the first layer The next layer extracts more intricate and higher-level characteristics by processing the lower-level ones. Sparse interactions and weight sharing are the predominant characteristics of the convolution layer [17].

2.4.2 Pooling Layer

In a convolutional neural network, a pooling layer is often used. The objective is to gradually reduce the size of the representation. It therefore reduces the number of features and the computational complexity of the network. The pooling layer operates on each feature map independently.[18] It is similar to reducing the resolution in the image processing domain. Number of kernels is not affected by pooling.[19]

The choice of kernel size and stride is also relevant for the pooling phases. There are three types of pooling:

(1) max pooling: selects the maximum element from the region of the feature map covered by the kernel.

(2) average pooling: calculating the average for each patch of the feature map.

(3) min pooling: selects the minimum element from the region [20].

The figure2.7 represented an example of max-pooling and average pooling operations.

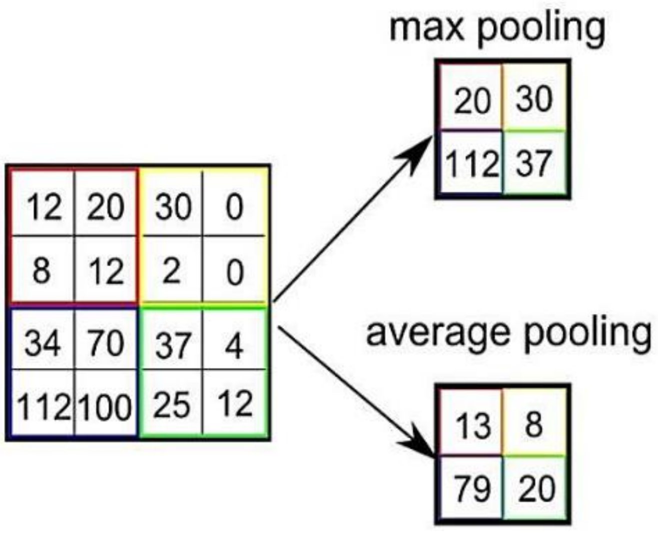


Fig. 2.7: Example of a pooling operation.

{ The max-pooling with 2×2 kernel and stride 2 lead to down-sampling of each 2×2 blocks is mapped to one block.}

A max pooling is the most popular approach used in the pooling operation A kernel of 2 by 2 would traverse over the entire matrix with a stride of 2 and pick the largest element from the window to be included in the next representation map.

2.4.3 Flatten Layer

After passing the convolution and pooling layers and before entering the fully connected layers, the output of the earlier layers is passed for flattening. By this, it is meant that the dimensions of the input array from previous phases are flattened out into one large dimension. For instance, a 3D array with a shape of 10×10×10 when flattened would become a 1D array with 1000 elements, which is rendered in Figure 2.8. [21]

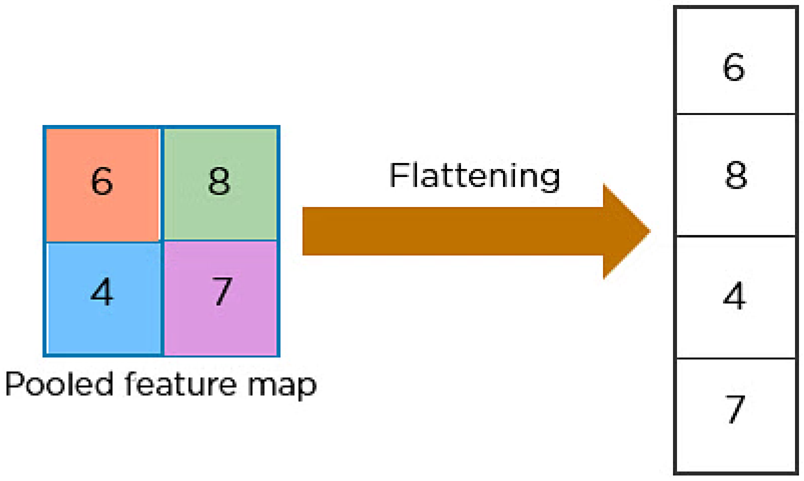


Fig. 2.8: Flattening mechanism

2.4.4 CNN Settings

Filtered

The convolution is a mathematical technique used to extract features from an image. The convolution is determined by the image kernel. The image kernel is simply a small matrix. A 3x3 kernel matrix is very common [22]. Nine filtered matrixes called "feature maps" are created by this procedure can be shown Figure 2.9 .

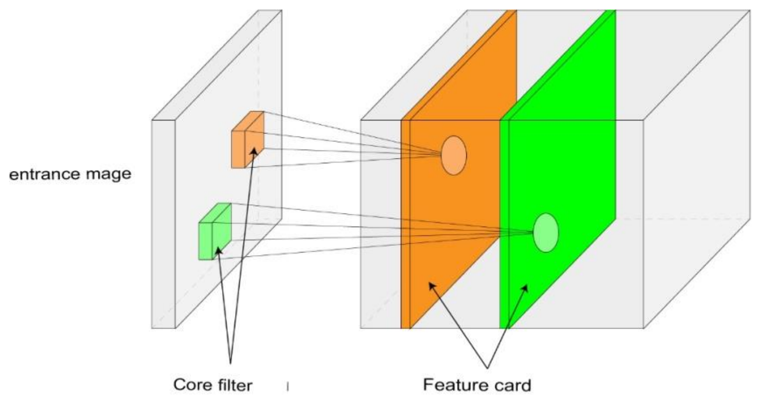


Fig. 2.9: Feature maps

Stride

Stride is the number of pixels shifts over the input matrix. For padding p, filter size $f*f$ and input image size $n*n$ and stride 's' our output image dimension will be:

$$[\{(n + 2p - f + 1) / s\} + 1] * [\{(n + 2p - f + 1) / s\} + 1]$$

Zero Padding

After convolution, our original image gets smaller. This happens because there are many layers of processing involved in image classification. If we do many convolutions, our original image will get smaller, but we don't want it to shrink every time.

The second problem is that when the kernel moves over the original images, it touches the edges less often and touches the middle of the image more often. It also overlaps in the middle. We don't use the corners or edges of any picture much in the final result.

Activation functions

Artificial neural networks use activation functions to transform input signals into output signals, which are fed as input to the next layer in the stack. In a neural network constructed from artificial neural cells, we calculate the sum of the product of inputs and their respective weights, then employ an activation function to obtain the output of that particular layer and use it as the source of input for the following layer [23]. we cite the most used:

- **Sigmoid**
  It is widely utilized as an activation function due to its non-linear nature. The sigmoid function transforms values between 0 and 1. It's defined as:

$$F(x) = \frac{1}{e^{-x}} \tag{2.5}$$

- **Softmax**

  The Softmax function combines many sigmoid functions. Since a sigmoid function returns values from 0 to 1, these can be treated as probabilities of a particular class of data points.

  The softmax function, unlike the sigmoid functions used for binary classification, can be used for multiclass classification problems. The function returns a value for each data point in all the classes. It can be expressed as:

$$F(x) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}} \tag{2.6}$$

The output layer of a network or model for classification of multiple classes will have the same number of neurons as the classes in the target when we build it.

- **Rectified Linear Unit (ReLU)**

  Rectified linear unit is a non-linear activation function , widely used in neural networks. A certain number of neurons are activated at a time, which makes ReLU more efficient than other functions. The weights and biases are not updated during the back- propagation step in neural network training when the value of gradient is zero. described by the following equation:

$$R(z) = \max(0, z) \quad [24] \tag{2.7}$$

Batch normalization

Batch normalization (BN) is a technique for normalizing activations in deep neural networks. BN is a popular technique for deep learning because it helps improve accuracy and speeds up training [25].

Dropout

Dropout is a new algorithm for training neural networks that relies on dropping out neurons during training to avoid feature detectors co-adapting [26].

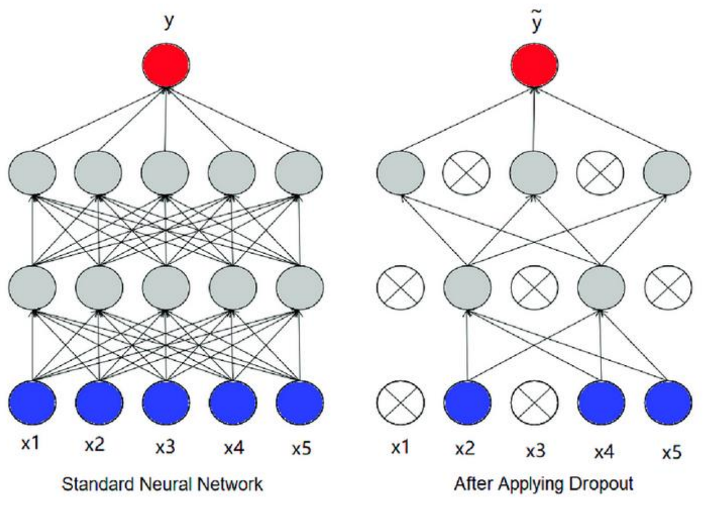


Fig. 2.10: Dropout mechanism in a multilayer neural network.

2.5 Architectures used

In our work, we are interested in the use of models (**ResNet152v2**, **ResNet18**, **VGG16**, **AlexNet**) which constitute reference architectures of CNNs, for classification and detection of agricultural diseases.

2.5.1 Model 01: AlexNet

AlexNet was designed by Hinton, winner of the 2012 ImageNet competition, and his student Alex Krizhevsky. The name comes from Alex Krizhevsky, in 2012 with a top-5 error rate of 15.3%, it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [27]. The most important features of the AlexNet paper are:

- Since the model had to train 60 million parameters (which is quite a lot), it was prone to overfitting. The paper found that Dropout and Data Augmentation significantly helped reduce overfitting. The first and second fully connected layers in the structure thus utilized a 0.5-point dropout for their respective purposes. By augmenting the number of images artificially, the dataset expanded dynamically during execution, allowing the model to generalize better.

- Another important thing was using ReLU activation instead of tanh or sigmoid, which made training faster (6 times faster) . Deep Learning Networks usually use ReLU non-linearity to train faster because other methods get saturated when they reach higher activation values.

Architecture

The architecture has eight layers, five of which are convolutional and three of which are fully connected as shown in the figure 2.11. According to [28] here are some of the characteristics used which constitute new approaches to convolutional neural networks:

- **ReLU nonlinearity**: The tanh function, which was the standard at the time, is replaced by Rectified Linear Units (ReLU) in AlexNet. A CNN trained using ReLU reached a 25% error on the CIFAR-10 dataset six times faster than a CNN trained using Tanh.

- **Multiple GPUs:**
  Back in the day, graphics cards GPUs were still rolling around with 3 gigabytes of memory (nowadays those kinds of memory would be rookie numbers) The training set had 1.2 million images, which made this especially bad. AlexNet allows for multi-GPU training by placing half of the model's neurons on one GPU and the other half on another GPU. This allows for training with multiple GPUs. This not only allows for the development of a larger model, but it also shortens the duration of the instruction.

- **Overlap Pooling** (Overlapping Pooling):
  CNNs usually "pool" the output of neighboring groups of neurons without overlapping. But when the authors added overlap, they saw a reduction in error of about 0.5% and found that models with overlapping pooling are harder to overfit.

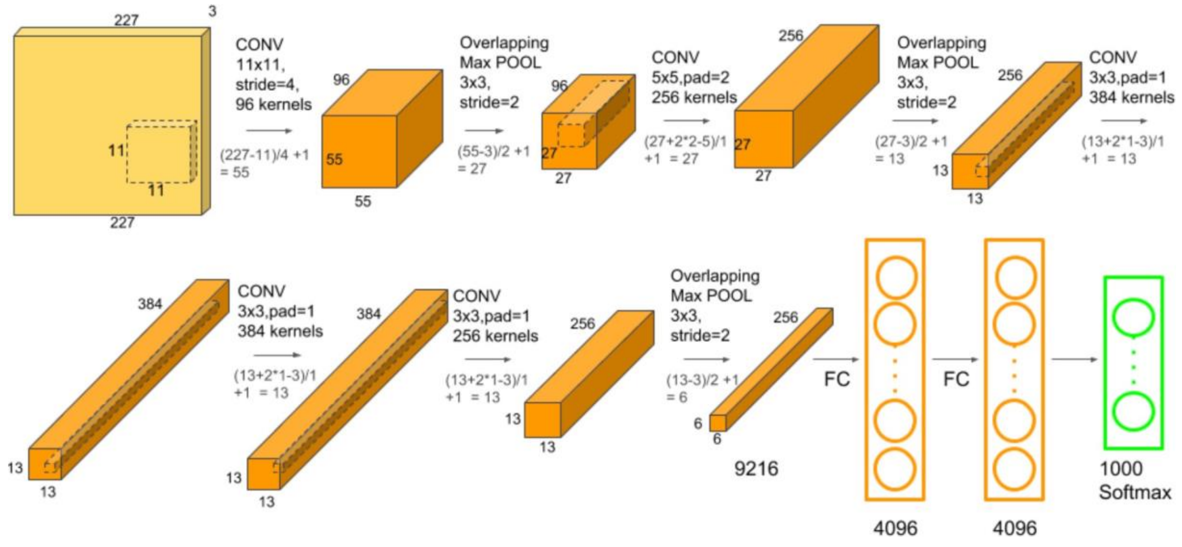


Fig. 2.11: Architecture of the AlexNet model

2.5.2 Model 02: VGG16

VGG is a convolutional neural network. proposed by K. Simonyan and A. Zisserman from Oxford University. It became popular after winning the ILSVRC (ImageNet Large Scale Image Recognition Competition) in 2014. The model was accurate by 92.7% on Imagenet, one of the best results [29].

Architecture

There are two algorithms: VGG16 and VGG19. In this work, we'll use the **VGG16** only. Both designs are similar, but VGG19 has more layers for convolution. The architecture of VGG-16 is shown in Figure2.12.

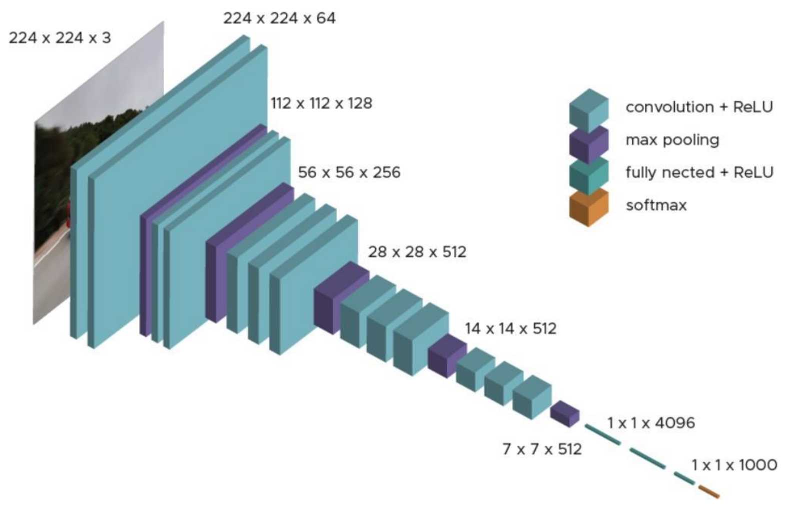


Fig. 2.12: Architecture of the VGG16 model

This model improves AlexNet by replacing large filters with a stack of3x3 size filters.

The model only requires specific pre-processing, which consists of subtracting the average RGB value, calculated over the training set, from each pixel.

The first convolution layer's input during training is a RGB image with dimensions of 224 x 224. The size of the convolution kernel is the smallest dimension to capture the notions of top, bottom, left, right, and center. The model had this feature at the time of publication. Until VGG16, many models were oriented towards convolution kernels of larger dimensions (size 11 or size 5, for example) Only discriminating information such as atypical geometric shapes can be retained by these layers.

The convolution layers are accompanied by Max-Pooling layers, each of size 2×2, to minimize the size of the filters during training.

There are 3 layers of fully-connected neurons at the output of the pooling layers. The first two are composed of 4096 neurons and the last of 1000 neurons, each with a softmax activation function to determine the image class.

2.5.3 Model 03: ResNet152v2

Deep neural networks are getting deeper and more complicated. It has been proven that adding more layers to a neural network can make it more robust for image-related tasks. But it can also cause them to lose their accuracy. That's where Residual Networks come into play. Deep learning practitioners tend to add so many layers in order to extract important features from complex images. There may be edges and recognizable shapes at the end of the first layers. Adding more than 30 layers to the network affects its performance and results in low precision. The notion that enlarging a neural network improves it is unfounded. This is not due to over fitting, because in that case, one could use dropout and regularization techniques to solve the issue altogether. The problem is caused by the popular vanishing gradient problem [30].

The ResNet152 model with its 152 layers topped the ILSVRC Imagenet 2015 test, despite having fewer parameters than the VGG19 network, which was a huge hit at the time. A residual network is composed of residual units or blocks that have no connections, also known as identity connections.

The output from the previous layer is added to the output of the layer after it in the residual block. There could be 1, 2 or even 3 hops or skips. A reduction of its dimensions may be caused by the convolution process when adding. Thus, we include an additional 1 x 1 convolution layer to alter the dimensions of x.

The figure2.13 illustrates a residual block with a convolution layer3×3, a normalization layerbatch, and a ReLU activation function. The latter is continued by a 3×3 convolution layer and a batch normalization layer. The skip connection effectively skips these two layers and is added prior to the ReLU activation function. A residual network is formed by repeating such residual blocks.

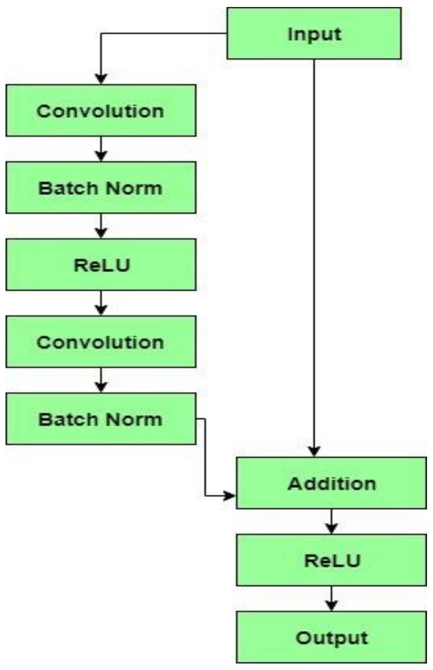


Fig. 2.13: A residual block

After comparing all the different CNN designs, ResNet had the lowest error rate of 3.57% for classification tasks, which was better than all the other designs. Even humans don't have much lower error rates than machines [30].

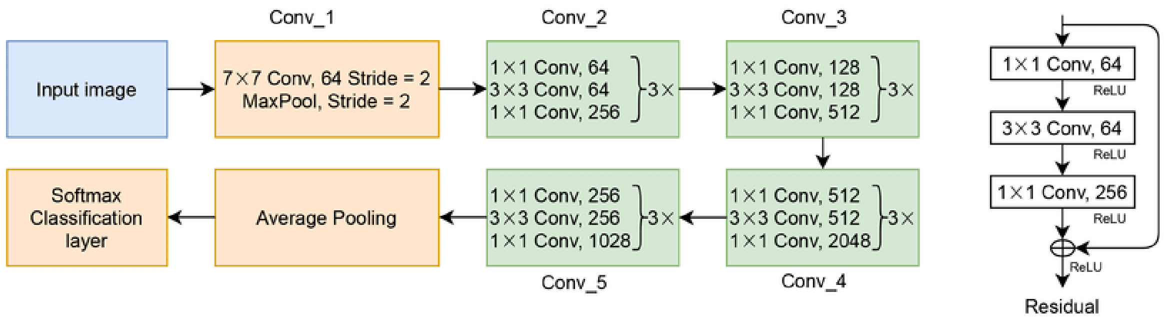


Fig. 2.14: The basic architecture of ResNet.

It can be said that residual networks have become very popular for image recognition and classification tasks because they can fix vanishing gradients when adding more layers to a deep neural network with a total number of parameters of about 60 million. Right now, the thousand-layer ResNet is not very useful.

2.6 Transfer Learning

Transfer learning is a technique in machine learning where a model that has been trained on one task is used to train another one. This can be useful when the second task is similar to the first task or when there is limited data available for the second task. This can be useful when there is limited data available for the second task. The model can learn more quickly and efficiently on the second task by applying the lessons learned from the first task. The model will already be aware of the general characteristics that are likely to be useful in the second task, which can prevent overfitting [31].

Why learning by transfer ?

In the early layers of a deep neural network trained on images, a deep learning model tries to learn a low level of features, like detecting edges, colors, variations of intensity, etc. No matter what type of image we are processing for detecting a lion or car, such kind of features appear to be specific to a particular dataset or task. Both scenarios require us to uncover these nitty-gritty details. All of these features occur regardless of the cost function or image dataset. Thus, acquiring these attributes in the case of detecting lions can be applied to other tasks, such as detecting individuals, as well [31].

The figure 2.15 shows the principle of transfer learning.

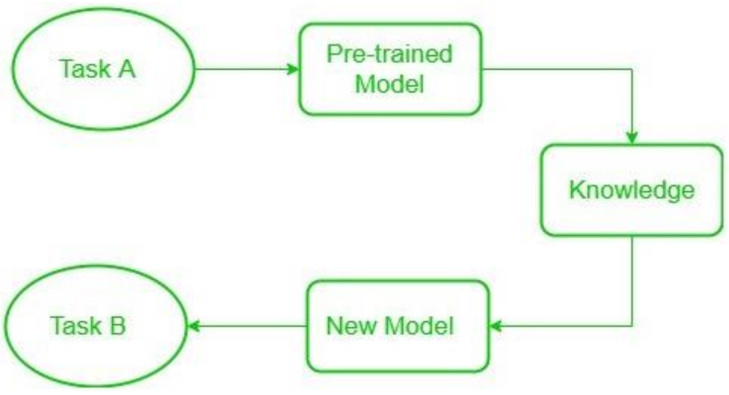


Fig. 2.15: Diagram showing the simplified process of transfer learning

When dealing with transfer learning, we encounter a phenomenon called "Freezing of layers". A layer, whether it is a CNN layer, a hidden layer, a block of layers or any subset of all layers, is said to be fixed when it is no longer available for learning. Therefore, the weights of frozen layers will not be updated during training. While layers that are not frozen follow the normal training procedure (see figure2.16).

When we use transfer learning to solve a problem, we choose a pre-trained model. There are two ways to use the information from the model that has already been trained. The first way is to keep some parts of the model we already trained and then train other parts on a new set of data for the new job. The other way is to create a new model and use some features from the model that was already trained. In both situations, we take out some of the things we learned and try to teach the rest of our model. This makes sure that the only feature that may be the same for both tasks is taken out of the pre-trained model and the rest of it is changed to fit the new dataset by training [31].

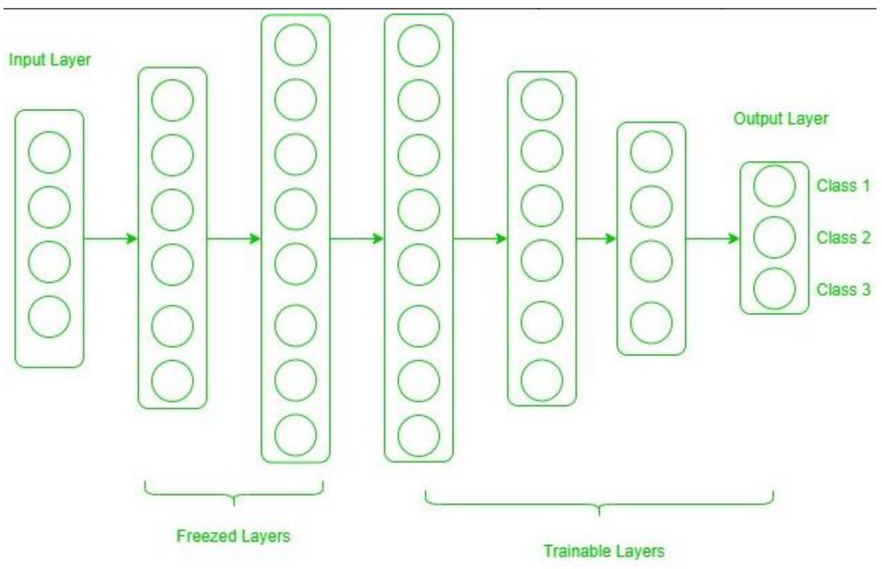


Fig. 2.16: Fixed layers and trainable layers

The idea is therefore to fix the weights of certain layers during training and to finish the rest to deal with the problem. This strategy makes it possible to reuse knowledge in terms of the overall architecture of the network and to use its state as a starting point for training. Therefore, it can achieve better performance with shorter training time.

The figure2.17 summarizes the main transfer learning methods commonly used in deep learning:

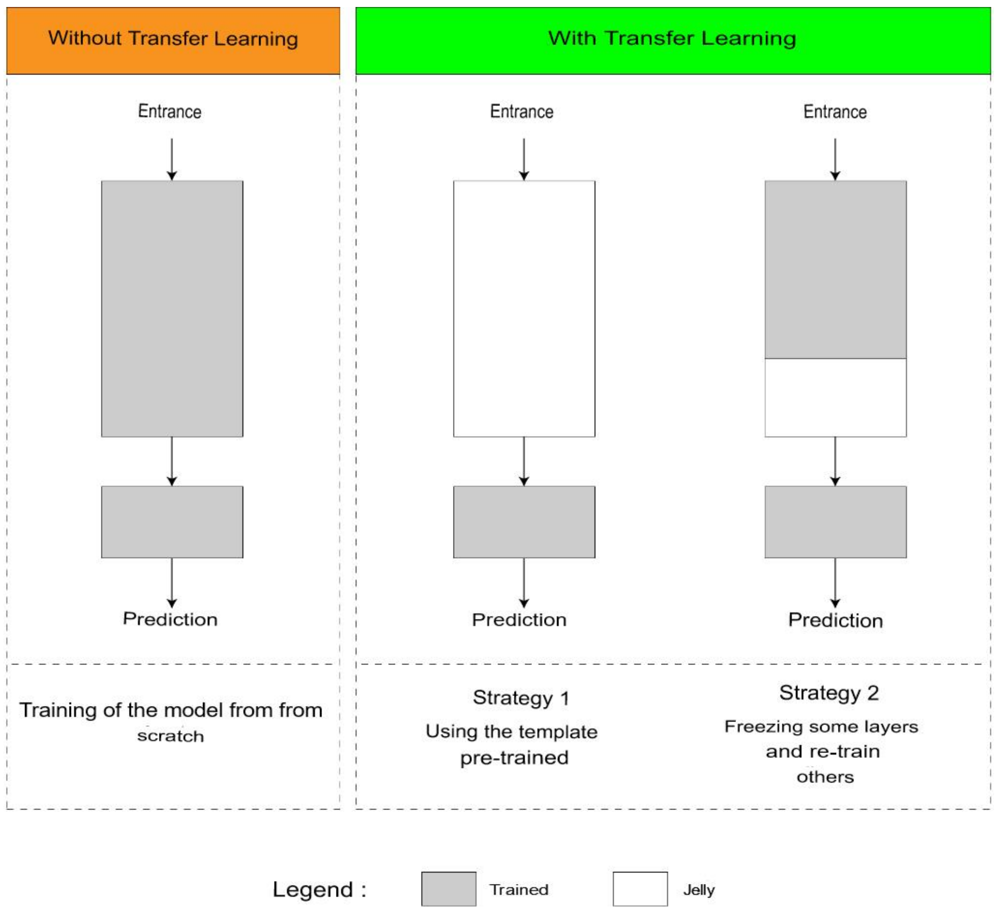

Without Transfer Learning
With Transfer Learning
Entrance
Entrance
Entrance
Prediction
Prediction
Prediction
Training of the model from from scratch
Strategy 1
Using the template pre-trained
Strategy 2
Freezing some layers and re-train others
Legend :
Trained
Jelly

Fig. 2.17: Transfer learning approach in Deep Learning

Therefore, one of the basic requirements of transfer learning is the existence of a model that performs well on the source task. Today, many state-of-the-art deep learning architectures are now freely available shared by their respective teams.

2.7 Conclusion

In this chapter, we started by discussing the history of artificial neural networks, then we detailed convolutional neural networks and their main parameters. We also presented the pre-trained architectures which are the CNN models adopted for plant disease classification.

In the following chapter we will discuss the methodology followed to create the plant disease diagnosis system, as well as the experimental results achieved and the different analyzes obtained when using the different types of the architectures.

Chapter 3

Methodology and experimental results

3.1 Introduction

This chapter is dedicated to explaining the methodology adopted and analyzing the performance of various systems implemented for the classification of agricultural diseases. Our analyses were conducted on a selected portion of the dataset available on Plant Village [1], which stands as a significant reference in the field of smart agriculture. Initially, we examine and compare pre-trained architectural models utilizing transfer learning techniques. In the second part, we evaluate the proposed CNN architecture for diagnosing plant diseases.

The systems were implemented in a Python development environment under Anaconda. Given that training Convolutional Neural Networks (CNNs) requires substantial computational resources, the programs were executed on virtual machines through Google Colab. This service is particularly suitable for deep learning applications that necessitate the use of multiple specialized libraries such as Keras, PyTorch, and TensorFlow. It provides access to high-performance GPUs, including Nvidia Tesla graphics cards, with a generous memory allocation, allowing for the efficient handling of compute-intensive tasks.

3.2 Dataset

All CNN models were trained on a subset of 14 categories including(Tomato, Potato, Grape, Orange, Soybean, Squash, Corn (maize), Strawberry, Peach, Apple, Blueberry, Cherry (including sour), Raspberry, and Pepper, bell) from a total of 70,295 images. This data is derived from a publicly accessible database known as Plant Village, which contains a total of 54,306 images featuring 38 different healthy/diseased leaves across 14 plant species.

The entire database is available on Kaggle, showcasing various plant diseases as illustrated in figure 3.1.

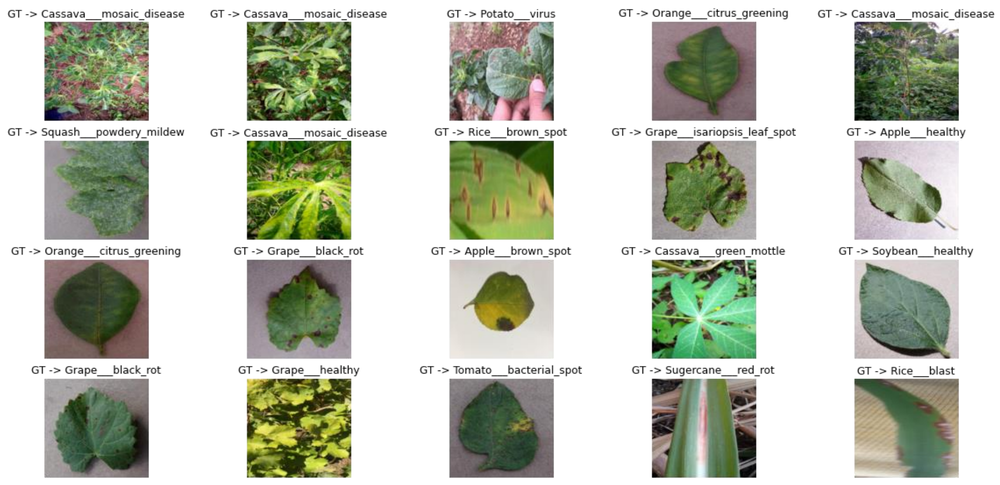

GT -> Cassava__mosaic_disease
GT -> Cassava__mosaic_disease
GT -> Potato__virus
GT -> Orange__citrus_greening
GT -> Cassava__mosaic_disease
GT -> Squash__powdery_mildew
GT -> Cassava__mosaic_disease
GT -> Rice__brown_spot
GT -> Grape__isariopsis_leaf_spot
GT -> Apple__healthy
GT -> Orange__citrus_greening
GT -> Grape__black_rot
GT -> Apple__brown_spot
GT -> Cassava__green_mottle
GT -> Soybean__healthy
GT -> Grape__black_rot
GT -> Grape__healthy
GT -> Tomato__bacterial_spot
GT -> Sugercane__red_rot
GT -> Rice__blast

Fig. 3.1: Some examples of plant diseases from the PlantVillage database

All images in the Plant Village database were captured at experimental research stations associated with Land Grant Universities in the United States (such as Penn State, Florida State, Cornell, among others). The collection of images is ongoing, and the database is expected to expand over time. These experimental research stations, both public and private, provide the opportunity to capture a large number of images in a short period. The majority of the images were taken by two technicians working as a team. During field trials of crops infected with a disease, the technicians collected leaves by detaching them from the plant. The leaves were then placed on a sheet of paper serving as a gray or black background. All images were captured outdoors, in natural light, which could vary from bright sunlight to cloudy conditions. This variety was intentional to mimic the range of conditions under which the end user (a farmer with a smartphone) might take pictures. For each leaf, 4 to 7 images were collected using a standard digital camera (Sony DSC Rx100/13, 20.2 megapixels) in automatic mode. The leaf was rotated 360 degrees during the photo shoot.

Distribution some samples

In this study, we used a subset consisting of 70,295 images from 14 different plant types, namely: Apple, Blueberry, Cherry (including sour), Corn (maize), Grape, Orange, Peach, Pepper, bell, Potato, Raspberry, Soybean, Squash, Strawberry, and Tomato. The analysis of multi-class pathologies was conducted on the infections affecting these plants, where the distribution of classes is as detailed on the table3.1.

Type of food	Associated disease	Number of images
Grape	Black rot	1888
	Esca (Black Measles)	1920
	Leaf blight (Isariopsis Leaf Spot)	1722
	Healthy	1692
Potato	Early blight	1939
	Late blight	1939
	Healthy	1824
Tomato	Bacterial spot	1702
	Early blight	1920
	Late blight	1851
	Leaf Mold	1882
	Septoria leaf spot	1745
	Spider moths Two-spotted spider moth	1741
	Target Spot	1827
	Yellow Leaf Curl Virus	1961
	Mosaic viruses	1790
	Healthy	1926
Strawberry	Leaf scorch	1774
	Healthy	1824
Apple	Cedar apple rust	1760
	Black rot	1987
	Apple scab	2016
	Healthy	2008
Blueberry	Healthy	1816
Cherry	Powdery mildew	1683
	Healthy	1826
Corn	Common rust	1907
	Gray leaf spot	1642
	Northern Leaf Blight	1908
	Healthy	1859
Orange	Haunglongbing	2010
Pepper bell	Bacterial spot	1913
	Healthy	1988
Peach	Bacterial spot	1838
	Healthy	1728
Raspberry	Healthy	1781
Soybean	Healthy	2022
Squash	Powdery mildew	1736

Tab. 3.1: Distribution of samples from the Plant Village dataset

This classification provides a comprehensive overview of the dataset's diversity in plant health conditions, serving as a foundation for the detailed multi-class analysis of plant pathologies.

It's observed that the dataset is significantly imbalanced, meaning there are classes represented with more instances than others. In such cases, CNN models tend to create a biased learning model that has poorer predictive accuracy on the minority classes compared to the majority classes. In rare instances, such as fraud detection or disease prediction, as in our case, it is crucial to correctly identify the minority classes. Therefore, the model should not be biased to only detect the majority class but should give equal weight or importance to the minority class. Consequently, the number of data per class needs to be increased. Today, a very popular technique called "data augmentation" is used to increase the amount of data in the training set by adding slightly modified copies of existing data or newly created synthetic data from existing data. This technique is explained as follows.

Data augmentation

Data augmentation enhances the diversity of our training dataset by applying random (yet realistic) transformations, such as image rotation, flipping, and zooming, as illustrated in the figure 3.2 .

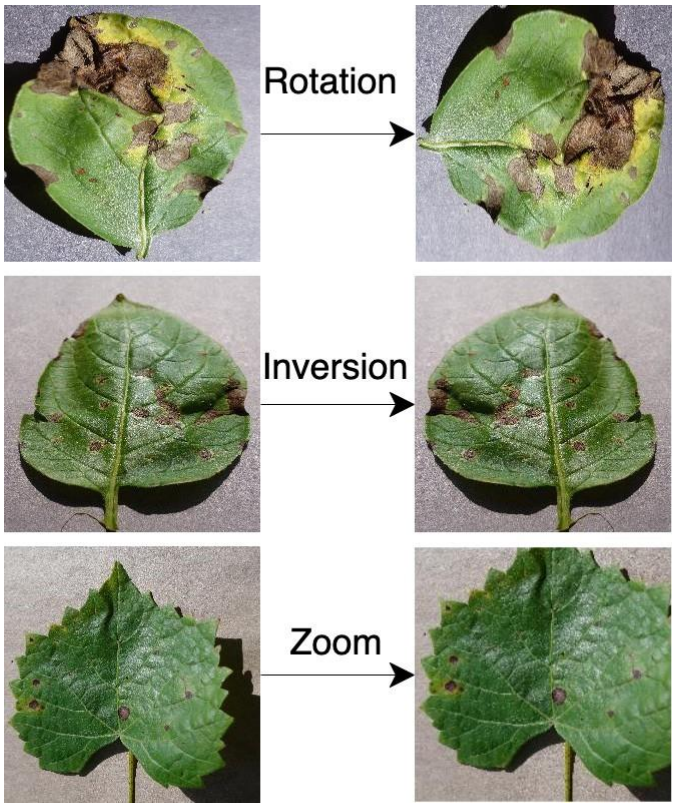


Fig. 3.2: Examples of transformation carried out on images

Table 3.2 shows the distribution of samples across the 17 classes after data augmentation. It is noted that 80% of the samples are used for the training phase, 10% for validation, and 10% for the testing phase.

Fruit/Vegetable	Disease	Training	Validation	Test
Apple	Apple_scab	2016	504	504
	Black_rot	1987	497	497
	Cedar_apple_rust	1760	440	440
	healthy	2008	502	502
Blueberry	healthy	1816	454	454
Cherry	Powdery_mildew	1683	421	421
	healthy	1826	456	456
Corn	Gray_leaf_spot	1642	410	410
	Common_rust	1907	477	477
	Northern_Leaf_Blight	1908	477	477
	healthy	1859	465	465
Grape	Black_rot	1888	472	472
	Esca	1920	480	480
	Leaf_blight	1722	430	430
	healthy	1692	423	423
Orange	Haunglongbing	2010	503	503
Peach	Bacterial_spot	1838	459	459
	healthy	1728	432	432
Pepper bell	Bacterial_spot	1913	478	478
	healthy	1988	497	497
Potato	Early_blight	1939	485	485
	Late_blight	1939	485	485
	healthy	1824	456	456
Raspberry	healthy	1781	445	445
Soybean	healthy	2022	505	505
Squash	Powdery_mildew	1736	434	434
Strawberry	Leaf_scorch	1774	444	444
	healthy	1824	456	456
Tomato	Bacterial_spot	1702	425	425
	Early_blight	1920	480	480
	Late_blight	1851	463	420
	Leaf_Mold	1882	470	400
	Septoria_leaf_spot	1745	436	405
	Spider_mites	1741	435	520
	Target_Spot	1827	457	456
	Yellow_Leaf	1961	490	485
	mosaic_virus	1790	448	452
	healthy	1926	481	450

Tab. 3.2: Sample distribution of our dataset after data augmentation

3.3 Software, libraries and hardware

In this section, we give a brief overview of the programming language and software tools used in our work, namely Python, Keras, TensorFlow as well asof the material used.

3.3.1 Software, booksellers

Python

Python is a high-level, interpreted programming language known for its easy-to-learn syntax that emphasizes readability, which helps reduce maintenance costs. It supports dynamic typing and binding, making it ideal for rapid application development and as a scripting language to connect existing components. Python's built-in data structures, along with its support for modules and packages, promote code reuse and program modularity. Available freely across major platforms, Python's extensive standard library and interpreter can be distributed without charge, enhancing its appeal[2].

Programmers often prefer Python due to the productivity boost it offers. The absence of a compilation step speeds up the edit-test-debug cycle. Debugging is straightforward, with exceptions raised for errors instead of segmentation faults, and a detailed stack trace is provided if exceptions are uncaught. Python also includes a source-level debugger for thorough inspection and evaluation of code. Additionally, Python's capability for introspection and the simplicity of adding print statements for debugging underscore its convenience and effectiveness in programming tasks. Python also stands out when compared to other languages for these reasons.

It enables developers to create machine learning applications using various tools, libraries, and community resources. We worked with version: 2.9.1

Torchvision

The torchvision library consists of popular datasets, model architectures, and image transformations for computer vision. It consists of:
• Training recipes for object detection, image classification, instance segmentation, video classification and sematic segmentation.
• 60+ pretrained models to use for fine-tuning (or training afresh).
• Dataset loaders for popular vision datasets such as ImageNet, COCO, Cityscapes and more [3]

Pytorch

PyTorch is an open-source machine learning framework that extends the Python programming language and Torch library, initially developed as an internship project by Adam Paszke under the guidance of Soumith Chintala, a Torch developer. It is designed for both deep learning research and application, offering over 200 mathematical operations and simplifying the creation of artificial neural network models. PyTorch is widely used by data scientists for research and artificial intelligence applications, enjoying growing popularity due to its ease of use for prototyping and deployment. The framework is under a modified BSD license and has strong ties to Meta (formerly Facebook), where Chintala works as a researcher and which uses PyTorch for all AI workloads.[4]

3.3.2 Material

For the hardware used, we conducted all training on the cloud to save time since Convolutional Neural Networks (CNNs) require significant hardware resources. Therefore, we opted for the Paperspace Gradient platform. Training a CNN requires a substantial amount of hardware resources, so we chose to use a cloud service from the Paperspace Gradient platform.

Google Colab

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.[5]

Caractéristique	Description
Model	NVIDIA Tesla K80/T4/P100
GPU memory	12GB GDDR5 (K80) / 16GB GDDR6 (T4, P100)
Memory bandwidth	240 GB/s (K80) / 320 GB/s (T4) / 732 GB/s (P100)
Processing units	4992 Cœurs CUDA (K80) / 2560 Cœurs CUDA (T4) / 3584 Cœurs CUDA (P100)
Peak performance	8.73 TFLOPS (Simple Precision, K80) / 8.1 TFLOPS (T4) / 9.3 TFLOPS (P100)
CUDA Version	Compatible with the latest versions
Availability	Depending on the type of subscription (free or Pro) to Google Colab

Tab. 3.3: Characteristics of GPUs used in training

3.4 Evaluation Metrics

Evaluation metrics are crucial for assessing the performance of a model. One of the key aspects of deep learning is understanding how to evaluate our model effectively. When developing a model, it's vital to measure how accurately it predicts the expected outcome.

Without properly evaluating the model using appropriate evaluation metrics, there's a risk of generating inaccurate predictions, especially if the dataset is imbalanced. Therefore, we suggest employing various evaluation metrics as described in the following sections.

3.4.1 Classification Accuracy

The simplest metric for model evaluation is accuracy. It represents the ratio of the number of correct predictions to the total number of predictions made for a dataset.

$$\text{Accuracy (\%)} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions made}} \times 100 \tag{3.1}$$

This metric is used when the dataset is balanced.

3.4.2 Confusion Matrix

A confusion matrix, or error matrix, is a table that displays the number of correct and incorrect predictions made by the model compared to the actual classifications in a dataset, thereby informing us about the mistakes (errors) made by the model. This matrix describes the performance of a classification model on test data for which the true values are known. It is an n×n matrix, where n is the number of classes. Figure 3.3 shows a confusion matrix for a 2-class problem. As can be seen, the results of a confusion matrix are classified into four major categories: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

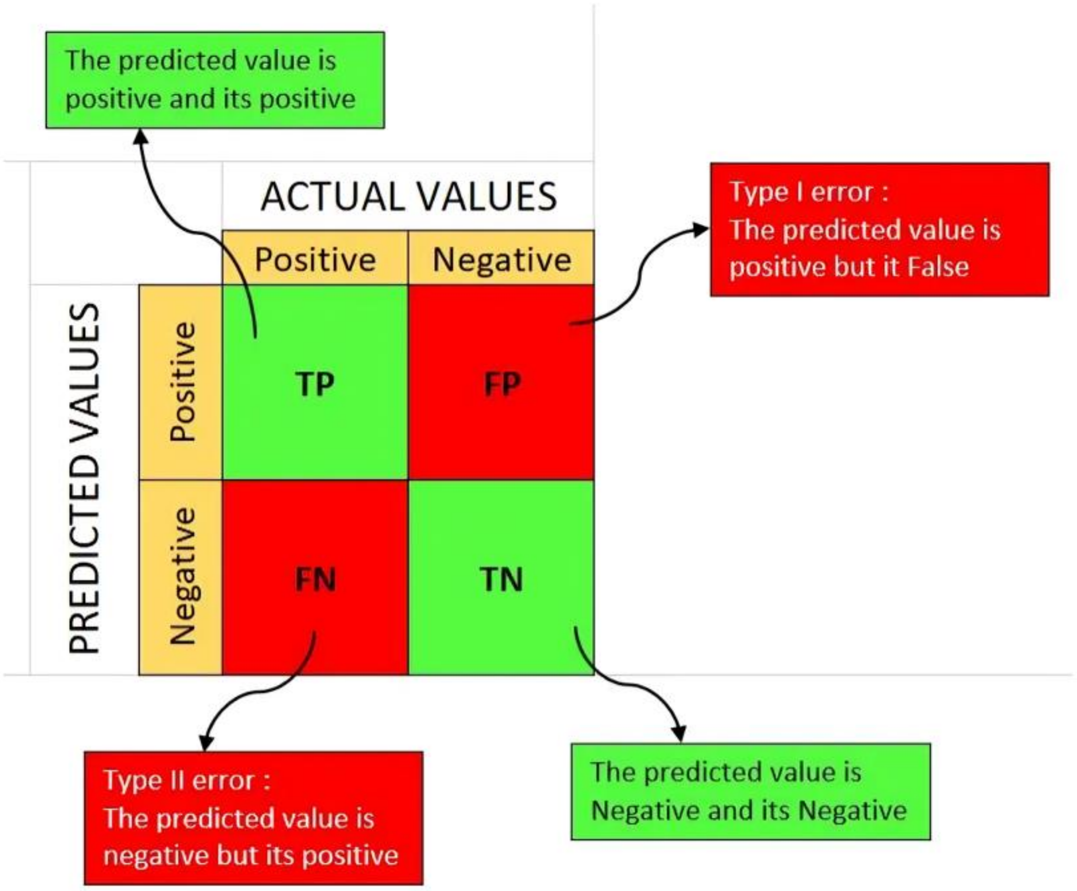

The predicted value is positive and its positive
ACTUAL VALUES
Positive
Negative
PREDICTED VALUES
Positive
Negative
TP
FP
FN
TN
Type I error : The predicted value is positive but it False
Type II error : The predicted value is negative but its positive
The predicted value is Negative and its Negative

Fig. 3.3: Basic Confusion Matrix Model

- **True positives (TP):**Number of samples that are truly positive and that are predicted to be positive.
- **True negatives (TN):**Number of samples that are truly negative and that are predicted to be negative.
- **False positives (FP):**Number of samples which are actually negative but predicted positive. These errors are also called type 1 errors.
- **False negatives (FN):**Number of samples that are actually positive but predicted negative. These errors are also called type 2 errors.

From the confusion matrix, we can derive 4 classification metrics:

Precision

Precision answers the following question: What proportion of positive predictions is actually correct? It then defines the number of true positives in relation to the number of true positives and false positives.

The equation can be stated as follows:

stated as follows:

$$Precision\ (\%) = \frac{\mathrm{TP}}{\mathrm{TP+FP}} \times\ 100 \tag{3.2}$$

Recall or Sensitivity or Recall

The recall answers the following question: what proportion of true positives were correctly identified? It then represents the ratio between the total number of examples correctly classified positives and the total number of positive examples.

The equation can be stated as follows:

$$Recall\ (\%) = \frac{\mathrm{TP}}{\mathrm{TP+FN}} \times\ 100 \tag{3.3}$$

F1-score

The F1 score gives an overall estimate of the precision and recall of a sample. This is the harmonic average of the precision and recall of a sample, the score F1 can be defined as follows

$$F1 - score(\%) = \frac{2\times precision\times Recall}{precision\times Recall} \times\ 100 \tag{3.4}$$

3.1 Experimental protocol

The development of a plant disease diagnostic system requires a very specific experimental protocol in order to achieve satisfactory results. In this section, we describe the adopted experimental protocol in terms of methodology, data distribution, and CNN architecture settings.

3.1.1 Methodology

Our primary goal is to develop a model capable of identifying 38 classes of plant leaves based on the type of crop and the diseases associated with that crop. We will use two methods for this purpose:

To clarify, this work focuses on 14 different crops. We compare two methods for diagnosing a plant leaf. Firstly, we classify based on the crop type, and then according to the diseases associated with that crop. Thus, for this first method, we have 4 classification models.

The second method involves creating a single model for all 38 classes, which is more suitable for the web application we will introduce in the next chapter. Figure 3.4 illustrates the two methods used for plant disease classification.

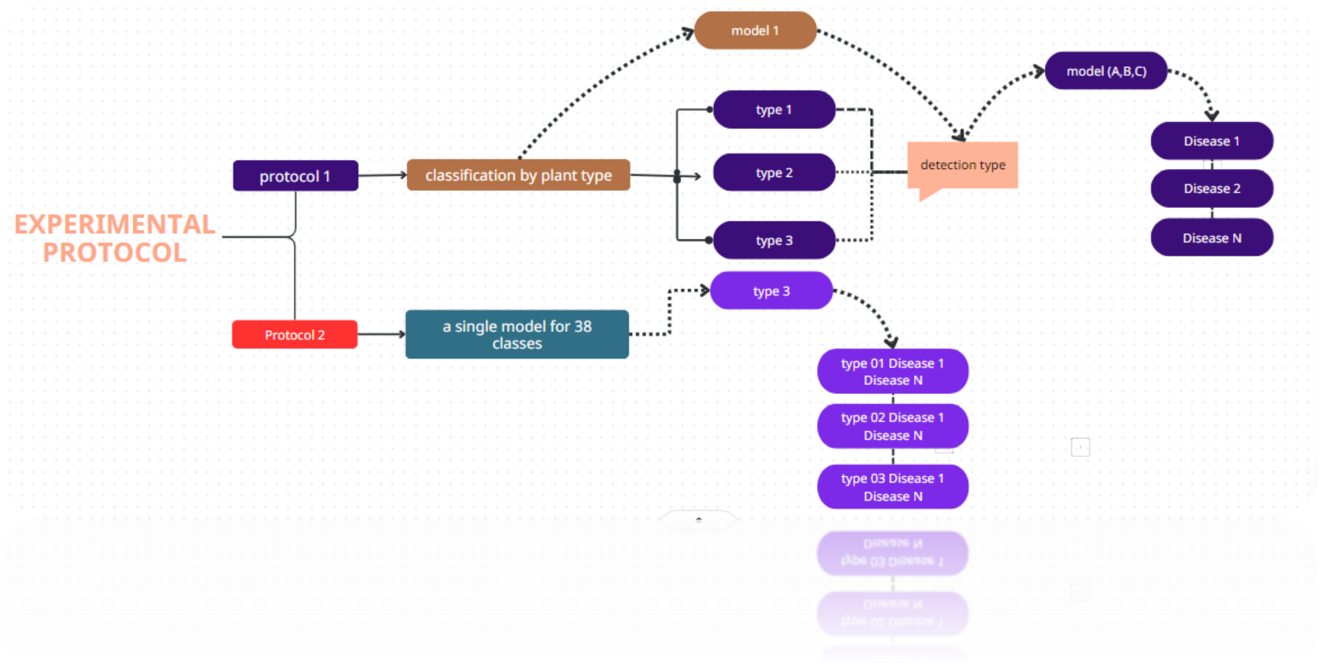

model 1
model (A,B,C)
type 1
type 2
type 3
detection type
Disease 1
Disease 2
Disease N
EXPERIMENTAL PROTOCOL
protocol 1
classification by plant type
type 3
Protocol 2
a single model for 38 classes
type 01 Disease 1 Disease N
type 02 Disease 1 Disease N
type 03 Disease 1 Disease N

Fig. 3.4: Methods adopted for the diagnosis of plant diseases

3.1.2 Data distribution

The images collected from the Plant Village database are distributed according to the two protocols explained previously.

Protocol 01

For the classification by type of plants, namely potato, grape, or tomato, we selected samples from each category to ensure the highest diversity possible. Notably, the dataset includes various classes with different numbers of images per class, as follows:

- Potato has 3 diseases with a total of 5702 images (Early blight: 1939, Late blight: 1939, Healthy: 1824).
- Grape has 4 diseases with a total of 7222 images (Black rot: 1888, Esca (Black Measles): 1920, Leaf blight (Isariopsis Leaf Spot): 1722, Healthy: 1692).
- Tomato has 10 diseases with a diverse number of images per class, contributing to a significant portion of the dataset.

It's important to note that 80% of the samples are utilized for the training phase of the model, 10% for validation, and 10% for testing. This structured approach ensures a comprehensive evaluation of the model's performance across a variety of plant diseases.

Category	Training (80%)	Validation (10%)	Test (10%)	Total
Apple	6049	756	757	7562
Blueberry	1453	181	182	1816
Cherry	3009	376	377	3759
Corn	6316	789	790	7895
Grape	7222	903	904	9029
Orange	1608	201	201	2010
Peach	2566	321	322	3209
Pepper	3901	488	489	4878
Potato	5702	713	714	7129
Raspberry	1425	178	178	1781
Soybean	1618	202	202	2022
Squash	1389	174	173	1736
Strawberry	2598	325	325	3248
Tomato	25455	3182	3183	31820
Total	**70006**	**8751**	**8752**	**87509**

Tab. 3.4: Data distribution for plant type classification

Subsequently, for the classification of diseases of each type of plant, the data were distributed into the 3 sets; training, validation And test according to THE tables 3.6 ,3.7 and 3.5.

Disease	Training	Validation	Test
Apple___Apple_scab	2016	504	100
Apple___Black_rot	1987	497	100
Apple___Cedar_apple_rust	1760	440	100
Apple___healthy	2008	502	100

Tab. 3.5: Data distribution
Type: Apple

Cherry (including sour)

Disease	Training	Validation	Test
Cherry_(including_sour)___Powdery_mildew	1683	421	100
Cherry_(including_sour)___healthy	1826	456	100

Tab. 3.6: Data distribution Type: Cherry (including sour)

Corn (maize)

Disease	Training	Validation	Test
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	1642	410	100
Corn_(maize)__*Common_rust*	1907	477	100
Corn_(maize)___Northern_Leaf_Blight	1908	477	100
Corn_(maize)___healthy	1859	465	100

Tab. 3.7: Distribution of data Type: Corn

Potato

Disease	Training	Validation	Test
Potato___Early_blight	1939	485	100
Potato___Late_blight	1939	485	100
Potato___healthy	1824	456	100

Tab. 3.8: Distribution of data Type: Potato

Tomato

Disease	Training	Validation	Test
Tomato___Bacterial_spot	1702	425	100
Tomato___Early_blight	1920	480	100
Tomato___healthy (example)	-	-	Multiple files with different sizes

Tab. 3.9: Distribution of data Type: Tomato

Grape

Disease	Training	Validation	Test
Grape___Black_rot	1888	472	100
Grape___Esca_(Black_Measles)	1920	480	100
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1722	430	100
Grape___healthy	1692	423	100

Tab. 3.10: Data distribution Type: Grape1

Orange

Disease	Training	Validation	Test
Orange___Haunglongbing_(Citrus_greening)	2010	503	100

Tab. 3.11: Data distribution Type: Orange

Peach

Disease	Training	Validation	Test
Peach___Bacterial_spot	1838	459	100
Peach___healthy	1728	432	100

Tab. 3.12: Data distribution Type: Peach

Pepper, bell

Disease	Training	Validation	Test
Pepper,_bell___Bacterial_spot	1913	478	100
Pepper,_bell___healthy	1988	497	100

Tab. 3.13: Data distribution Type: Pepper, bell

Raspberry

Disease	Training	Validation	Test
Raspberry___healthy	1781	445	100

Tab. 3.14: Data distribution Type: Raspberry

Soybean

Disease	Training	Validation	Test
Soybean___healthy	2022	505	100

Tab. 3.15: Data distribution Type: Soybean

Squash

Disease	Training	Validation	Test
Squash___Powdery_mildew	1736	434	100

Tab. 3.16: Data distribution Type: Squash

Strawberry

Disease	Training	Validation	Test
Strawberry___Leaf_scorch	1774	444	100
Strawberry___healthy	1824	456	100

Tab. 3.17: Data distribution Type: Strawberry

Blueberry

Disease	Training	Validation	Test
Blueberry___healthy	1816	445	100

Tab. 3.18: Data distribution Type: Blueberry

Protocol 02

This time a classification of 38 classes at a time is carried out using the distribution of data indicated in the following table:

Fruit/Vegetable	Disease	Training	Validation
Apple	Apple_scab	2016	504
	Black_rot	1987	497
	Cedar_apple_rust	1760	440
	healthy	2008	502
Blueberry	healthy	1816	454
Cherry	Powdery_mildew	1683	421
	healthy	1826	456
Corn	Gray_leaf_spot	1642	410
	Common_rust	1907	477
	Northern_Leaf_Blight	1908	477
	healthy	1859	465
Grape	Black_rot	1888	472
	Esca	1920	480
	Leaf_blight	1722	430
	healthy	1692	423
Orange	Haunglongbing	2010	503
Peach	Bacterial_spot	1838	459
	healthy	1728	432
Pepper bell	Bacterial_spot	1913	478
	healthy	1988	497
Potato	Early_blight	1939	485
	Late_blight	1939	485
	healthy	1824	456
Raspberry	healthy	1781	445
Soybean	healthy	2022	505
Squash	Powdery_mildew	1736	434
Strawberry	Leaf_scorch	1774	444
	healthy	1824	456
Tomato	Bacterial_spot	1702	425
	Early_blight	1920	480
	Late_blight	1851	463
	Leaf_Mold	1882	470
	Septoria_leaf_spot	1745	436
	Spider_mites	1741	435
	Target_Spot	1827	457
	Yellow_Leaf	1961	490
	mosaic_virus	1790	448
	healthy	1926	481

Tab. 3.19: Distribution of data from the 17 classes

3.1.3 Training

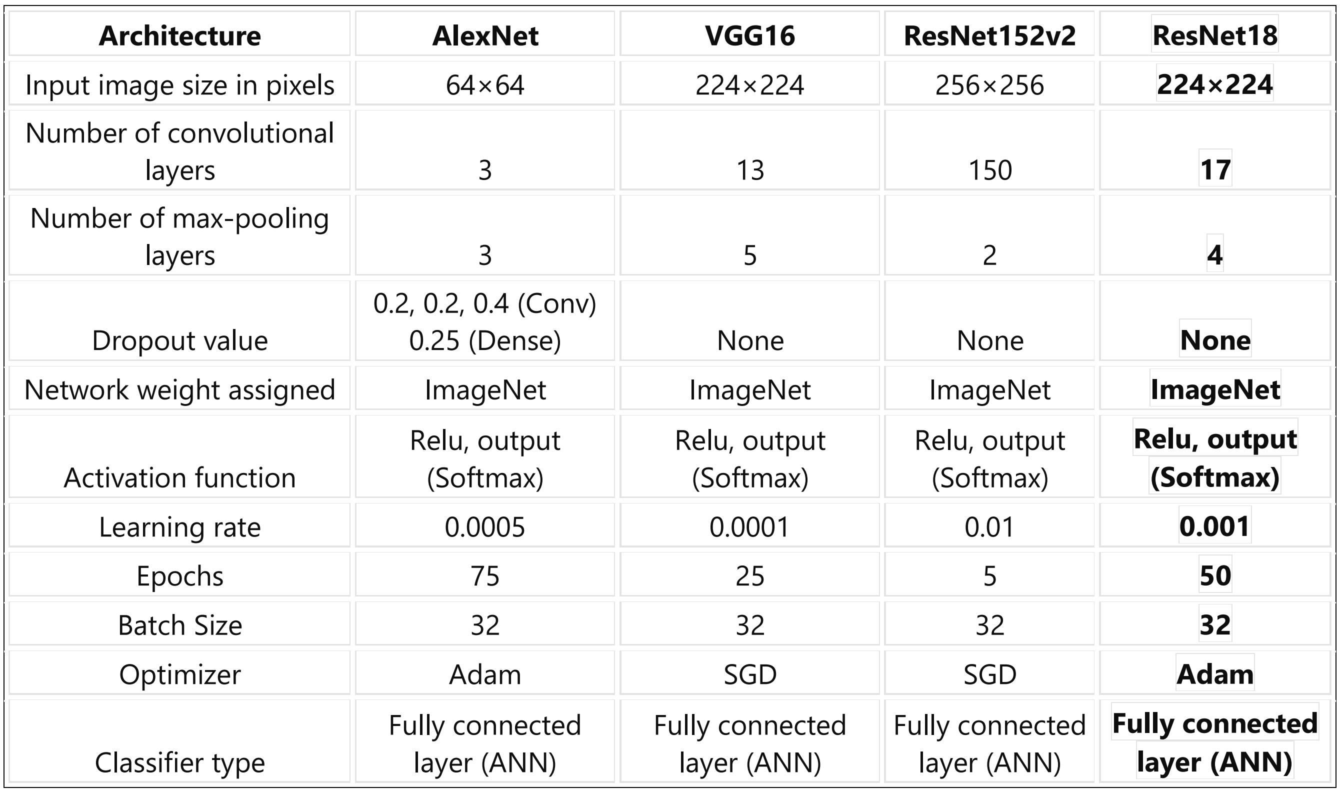
- Constructing an effective model requires a thorough analysis not only of the network design but also of the input data format. Therefore, plant leaf images were preprocessed so the CNN model could extract the appropriate features from them. Two preprocessing steps were applied to our dataset: normalization of the RGB values between 0 and 1, and resizing the images according to the model used.

- It was challenging to load all images at a resolution of 256 x 256, hence a data generator was used to reduce memory consumption.

- During the training phase, we utilized ImageNet pre-trained weights for the ResNet-18 architecture. The size of the input leaf images depends on the model we use. The dataset was divided into three subsets: training, validation, and test, with respective proportions of 80%, 10%, and 10%, as shown in the previous tables. The training subset refers to the data used to train the deep learning model. The validation dataset is used to measure the accuracy of the deep learning model during training. Finally, the test data set is considered the final set of data used to measure the model's performance on entirely new data.

- Regarding the update of the weights in the fully connected layer of the CNN, Adam and SGD (Stochastic Gradient Descent) optimizers are used with different learning rates for the ResNet-18 model as indicated in table 3.9. A batch size of 32 is used for training all the models presented in section 2.5.

Table3.9 summarizes all parameters used during training based onof the model.



Architecture	AlexNet	VGG16	ResNet152v2	ResNet18
Input image size in pixels	64×64	224×224	256×256	**224×224**
Number of convolutional layers	3	13	150	**17**
Number of max-pooling layers	3	5	2	**4**
Dropout value	0.2, 0.2, 0.4 (Conv) 0.25 (Dense)	None	None	**None**
Network weight assigned	ImageNet	ImageNet	ImageNet	**ImageNet**
Activation function	Relu, output (Softmax)	Relu, output (Softmax)	Relu, output (Softmax)	**Relu, output (Softmax)**
Learning rate	0.0005	0.0001	0.01	**0.001**
Epochs	75	25	5	**50**
Batch Size	32	32	32	**32**
Optimizer	Adam	SGD	SGD	**Adam**
Classifier type	Fully connected layer (ANN)	Fully connected layer (ANN)	Fully connected layer (ANN)	**Fully connected layer (ANN)**

Tab. 3.20: Training parameters adopted

3.2 Performance achieved

Firstly, we will evaluate the different CNN models adopted for the diagnosis of plant diseases according to the 2 protocols presented previously.

3.2.1 Protocol 01

Remember that in this first protocol, we first carry out a classification by type of plant before moving on to the classification of the disease presented by the leaf depending on the type of plant found. For this, several evaluation criteria are taken into account notably ; the size of the model, the time required for its training as well as time inference of a single piece of data since we are considering a Web implementation of the final system. The quantitative evaluation of the performance of the different models is mainly based on the precision (accuracy) in the different training sets, validation and testing.

The results of the classification by plant type are given in Table 3.10. We notice,

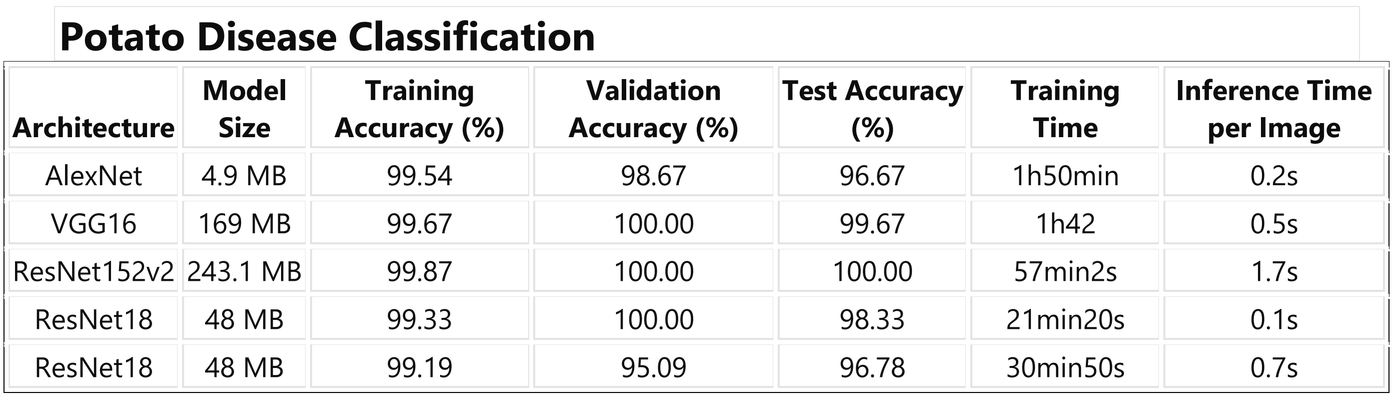
Architecture	Model Size	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)	Training Time	Inference Time per Image
AlexNet	4.9 MB	98.66	98.78	99.80	12d1h (30 epochs)	0.7s
VGG16	180.6 MB	95.32	95.59	100.00	140h24min (10 epochs)	0.8s
ResNet152v2	203 MB	98.40	98.46	100.00	50h10min (22 epochs)	1.75s
ResNet18	**48 MB**	**98.73**	**99.50**	**99.85**	**3h45min (24 epochs)**	**0.9s**

Tab. 3.21: Results of classification by plant type

The results of the plant type classification show perfect accuracy of 100% for the VGG16 and ResNet152 models, followed by the proposed model which presents an accuracy of 99.85%. Also, the training times remain reasonable with the shortest time of 3h45min with 24 epochs offered by ResNet18. In the 2nd position comes ResNet152v2 model with a training time of 50h10min with 22 epochs. As for the inference time, our model, as well as AlexNet, offer the shortest time of 0.7s for a single image. ResNet152 presents a relatively high inference time of 1.75s or 2min55s. Moreover, and unsurprisingly, ResNet152 and VGG16 occupy the most memory space with sizes of 203MB and 180.6MB, respectively. The proposed model and AlexNet are the least memory-intensive as they are the least deep. Overall, the proposed model presents the best performance, even though it has an error of 0.15% on the test data, which corresponds to 2 images incorrectly classified out of a total of 762 images.

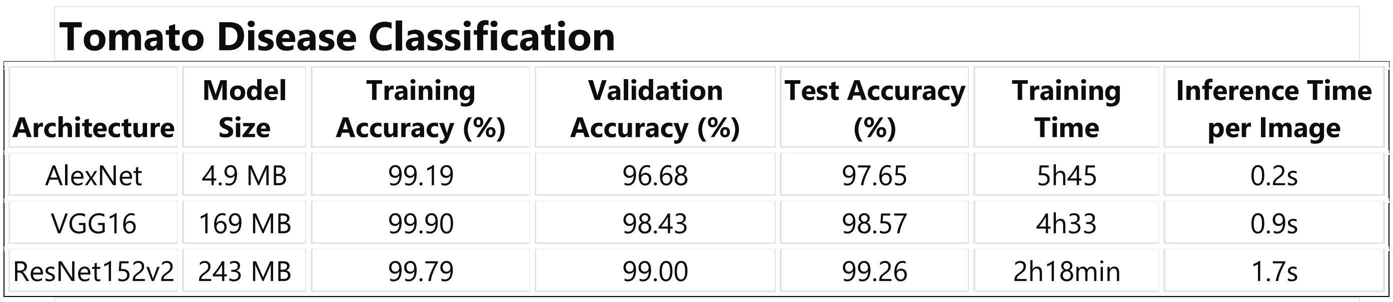
For For the second part of this experiment, which concerns the classification of diseases associated with each type of plant:

the results obtained are illustrated in table 3.11:



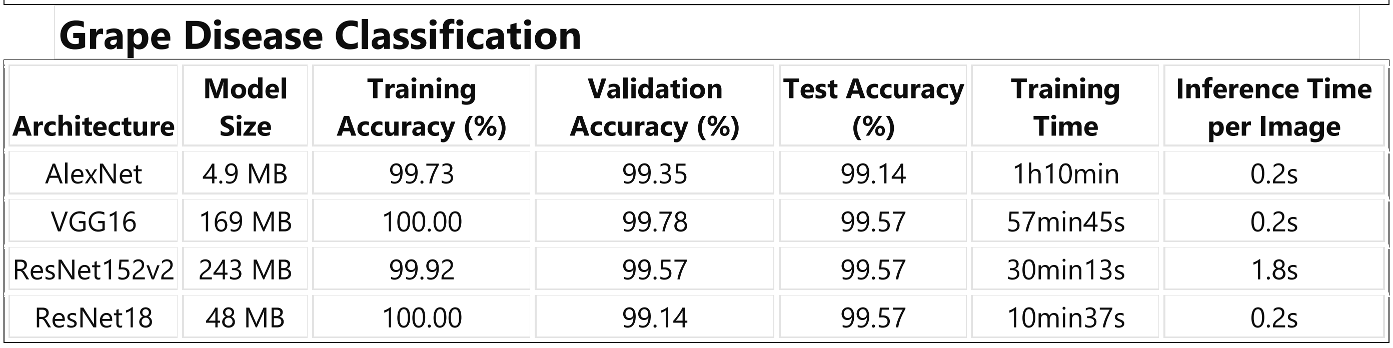
Potato Disease Classification

Architecture	Model Size	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)	Training Time	Inference Time per Image
AlexNet	4.9 MB	99.54	98.67	96.67	1h50min	0.2s
VGG16	169 MB	99.67	100.00	99.67	1h42	0.5s
ResNet152v2	243.1 MB	99.87	100.00	100.00	57min2s	1.7s
ResNet18	48 MB	99.33	100.00	98.33	21min20s	0.1s
ResNet18	48 MB	99.19	95.09	96.78	30min50s	0.7s



Tomato Disease Classification

Architecture	Model Size	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)	Training Time	Inference Time per Image
AlexNet	4.9 MB	99.19	96.68	97.65	5h45	0.2s
VGG16	169 MB	99.90	98.43	98.57	4h33	0.9s
ResNet152v2	243 MB	99.79	99.00	99.26	2h18min	1.7s



Grape Disease Classification

Architecture	Model Size	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)	Training Time	Inference Time per Image
AlexNet	4.9 MB	99.73	99.35	99.14	1h10min	0.2s
VGG16	169 MB	100.00	99.78	99.57	57min45s	0.2s
ResNet152v2	243 MB	99.92	99.57	99.57	30min13s	1.8s
ResNet18	48 MB	100.00	99.14	99.57	10min37s	0.2s

Tab. 3.22: Results of multiclass classification

The tables present a comparison of four deep learning architectures—AlexNet, VGG16, ResNet152v2, and ResNet18—across three different plant disease classification tasks: potato, tomato, and grape. Each architecture's performance is evaluated based on model size, training accuracy, validation accuracy, test accuracy, training time, and inference time per image. Here are some observations and insights based on the data provided:

1. Model Size and Efficiency: There's a clear trade-off between model size and performance. VGG16 and ResNet152v2, being larger models, generally offer higher accuracy, especially in validation and test phases. However, ResNet18 demonstrates that a significantly smaller model (48 MB compared to VGG16's 169 MB and ResNet152v2's 243.1 MB) can still achieve high accuracy with much less training time, making it an efficient choice for these tasks.

2. Accuracy Across Tasks: ResNet152v2 stands out with perfect test accuracy in the potato disease classification task, showcasing its strong generalization ability. However, in tomato and grape disease classification tasks, the differences in test accuracy among the models are narrower, indicating that smaller models like AlexNet and ResNet18 are also capable contenders for these tasks.

3. Training Time: Training time varies widely across models and tasks, with ResNet152v2 generally requiring less time to train despite its larger size. This might be attributed to its deeper and more optimized architecture that can learn more efficiently. ResNet18, in particular, shows an impressive balance between training time and performance, especially in the grape disease classification task where it achieves high accuracy in just over 10 minutes.

4. Inference Time: Inference time is crucial for real-world applications where decisions need to be made quickly. All models perform well in this aspect, but ResNet18 and AlexNet stand out with the shortest inference times across all tasks. This makes them attractive for deployment in scenarios where computational resources are limited or when fast decision-making is critical.

Overall, while larger models like VGG16 and ResNet152v2 offer slightly higher accuracy, the efficiency and effectiveness of ResNet18 make it an excellent choice for practical applications in plant disease classification. Its balance of size, speed, and accuracy demonstrates the advancements in designing compact yet powerful neural network architectures suitable for a wide range of applications.

3.2.2 Protocol 02: 38 classes at the same time

For this second protocol, we approached the classification of diseases by doing combine 18 types of plants at once, to make a single model capable of distinguish between 38 classes the criteria for evaluating the performance of the models, we used, precision, recall, F1-score as well as analysis of the confusion matrix to understand how confused the model is when making predictions. Not only does this will allow us to know what errors are being made, but above all, the types of confusion precisely in relation to food and different diseases. The results obtained are summarized in the table3.12.

Architecture	Model Size	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)	Training Time	Inference Time per Image
AlexNet	4.9 MB	97.67	97.79	99.80	20d1h (30 epochs)	0.7s
VGG16	180.6 MB	93.11	93.23	100.00	192h24min (10 epochs)	0.8s
ResNet152v2	203 MB	97.71	97.25	100.00	90h10min (22 epochs)	1.75s
ResNet18	**48 MB**	**97.73**	**99.34**	**99.85**	**3h45min (24 epochs)**	**0.9s**

Tab. 3.23: Classification results of 38 classes at the same time

The table presents the classification results for a dataset with 38 classes using four different architectures: AlexNet, VGG16, ResNet152v2, and ResNet18. Each model's performance is evaluated based on several criteria, including model size, training accuracy, validation accuracy, test accuracy, training time, and inference time per image.

- **AlexNet** shows impressive performance with the highest test accuracy of 99.80% among all models. Despite its smaller model size (4.9 MB), it achieves high accuracy levels, making it an efficient choice for applications where model size and inference speed are critical. However, its training time is significantly longer than the others.

- **VGG16** has a perfect test accuracy of 100.00%, indicating its capability to generalize well on unseen data. However, it has the largest model size (180.6 MB) and relatively low training and validation accuracies, which might suggest overfitting to the test set or a particularly well-suited architecture for the specific test data distribution. Its training time is also the longest, which could be a drawback for iterative development cycles.

- **ResNet152v2** also achieves perfect test accuracy (100.00%) with high model complexity (203 MB). Its training time is considerably less than VGG16, making it a more efficient choice for achieving high accuracy. However, the inference time per image is the longest, which might be a limitation for real-time applications.

- **ResNet18** stands out for its balance between model size (48 MB), accuracy, and efficiency. It achieves nearly perfect test accuracy (99.85%) with much shorter training time compared to the more complex models and has a moderate inference time per image. Its high validation accuracy suggests good generalization capability. Given its efficiency and effectiveness, ResNet18 appears to be the most balanced choice among the evaluated architectures, especially for scenarios where both accuracy and operational efficiency are important.

Overall, the choice of architecture should be guided by the specific requirements of the application, including the acceptable trade-offs between accuracy, model size, training time, and inference speed. ResNet18 seems to offer the best balance for most applications, providing high accuracy with reasonable computational requirements.

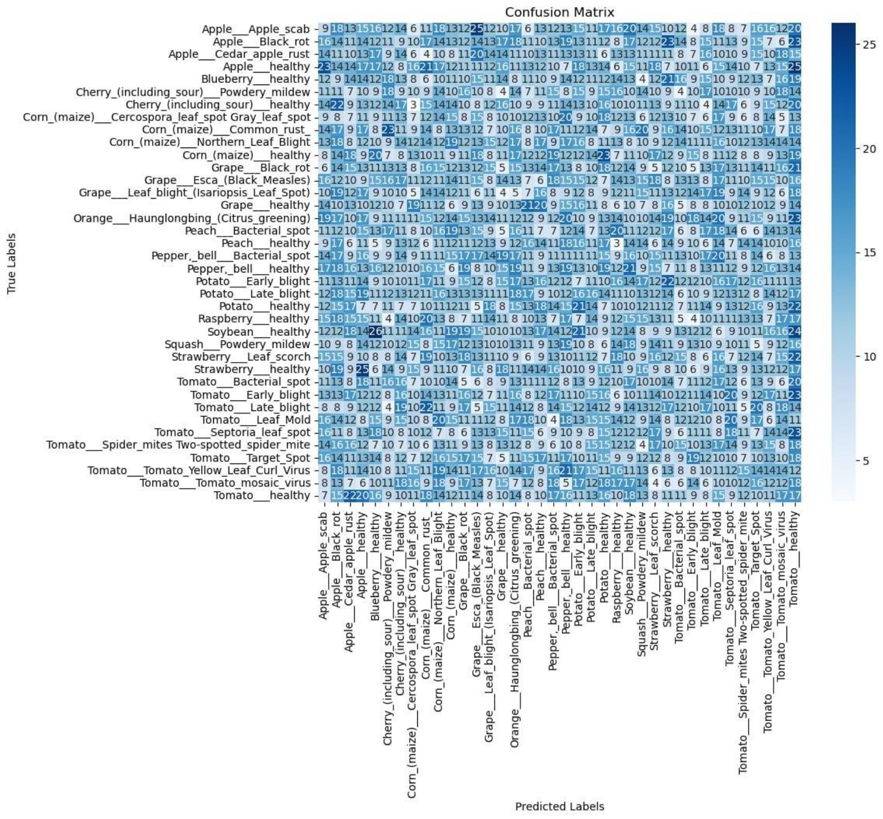


Fig. 3.5: Confusion matrix of the proposed model

Analysis of the confusion matrix shows confusion between potato and tomato for the same “late blight” disease. In fact, 8 out of 100 images of potato late blight have were classified tomato late blight, and 7 images out of 232 tomato late blight were classified potato late blight. Also, we notice significant confusion in diseases of the tomato plant, particularly between late blight, early blight, septoria leaf diseases spot, leaf mold, spider mites two spotted spider mite and target spot, such as:

- 10 tomato late blight images are categorized tomato early blight

- 8 images of tomato septoria leaf spot are categorized tomato leaf mold

- 11 images of tomato spider mites two spotted spider mite are categorized tomato target spot

This analysis can be confirmed by the precision and recall values given in table 3.13, the higher the precision, the more the model minimizes the number of false positive, while the higher the recall, the more the model maximizes the number of true positive. To have a more global view of the performance of the model we consider the F-1 score, the higher it is, the more efficient the model.

The best results achieved by the proposed CNN model are summarized in the figure 3.6

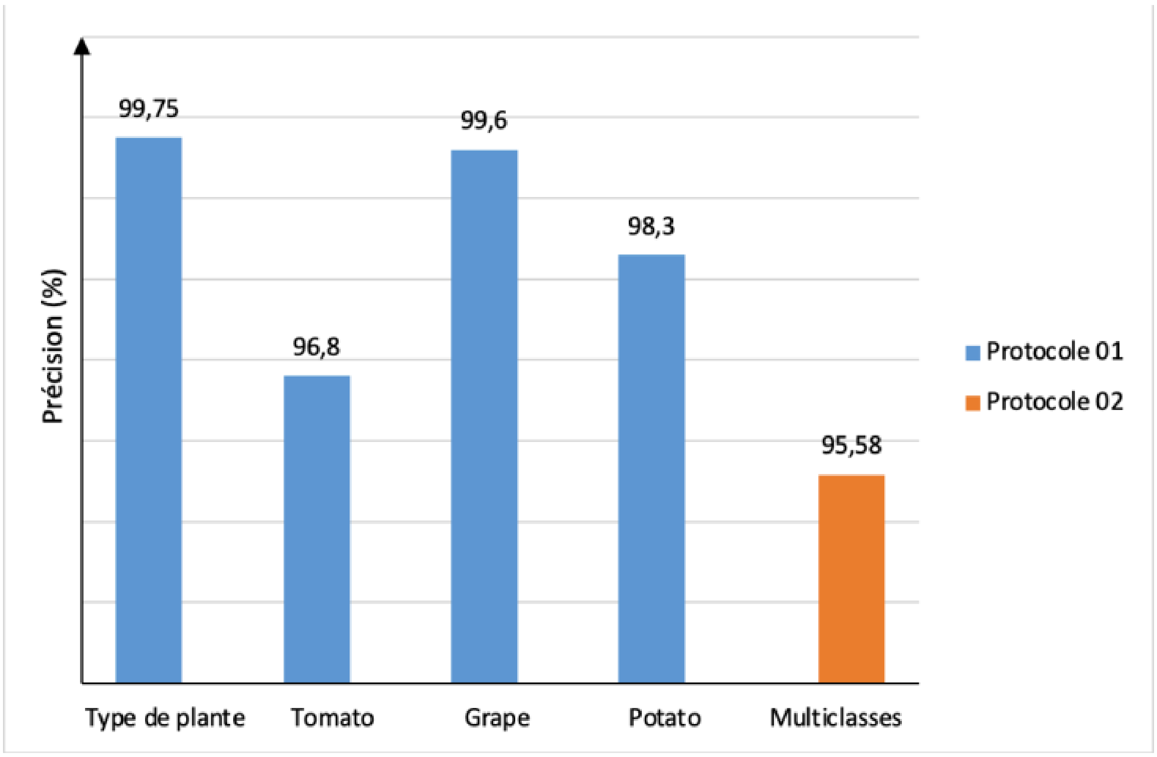


Fig. 3.6: Comparison of results obtained with the two protocols

Disease Category	Precision (%)	Recall (%)	F1-score (%)	Number of Images	Files Count
Apple___Apple_scab	94.50	95.00	94.75	150	2016
Black rot	95.90	99.15	97.50	118	1888
Esca (Black Measles)	100.00	99.28	99.64	139	1920
Leaf blight (Isariopsis Leaf Spot)	99.02	92.66	95.73	109	1722
Healthy	96.12	99.00	97.54	100	Multiple
Early blight	96.00	96.00	96.00	100	Multiple
Late blight	89.00	89.00	89.00	100	Multiple
Bacterial spot	97.26	99.53	98.38	214	Multiple
Leaf Mold	92.86	99.15	95.90	236	1882
Septoria leaf spot	94.69	89.50	92.02	219	1745
Spider mites Two-spotted spider mite	97.63	94.06	95.81	219	1741
Target Spot	91.70	96.51	94.04	229	1827
Yellow Leaf Curl Virus	97.18	97.97	97.57	246	1961
Mosaic virus	97.38	99.11	98.24	225	1790
Accuracy/Macro Avg/Weighted Avg	95.57	95.57	95.57	3068	

Tab. 3.24: Precision, recall and F1-score by class of the proposed model

3.3 Discussion

The experiments conducted for the development of a plant disease diagnosis system focused on leveraging convolutional neural networks (CNNs). The tests, carried out on a subset of the Plant Village database following two protocols, revealed the following insights:

- Pre-trained architectures like ResNet152v2 and VGG16 showed superior performance. However, their deployment demands significant computational resources, necessitating the use of powerful hardware (GPUs). This led us to utilize Google Colab's virtual machines. Moreover, these models are memory-intensive due to their complex structures, which is not economical for web-based implementation of the proposed solution. Despite their impressive capabilities, we had to set them aside due to their dem

- anding resource and time consumption, and instead, sought to develop a simpler, less resource-intensive model that still delivers satisfactory performance.

- The CNN model we proposed, **ResNet18**, is notably smaller in size compared to the pre-trained models, yet it achieves high accuracy. Additionally, our experiments with VGG16 and AlexNet underscored the significance of incorporating Dropout and additional hidden layers in enhancing the CNNs' classification efficacy.

- The implemented model effectively distinguishes between the classes associated with each type of plant, achieving test data accuracy ranging from 96% to 99%. It's important to note that the differentiation among the three plants in protocol 01 is relatively straightforward. However, a visible similarity between certain diseases affecting the "tomato" class was observed, as illustrated in the confusion matrix in figure 3.5. This matrix highlighted confusion among several images between the "late blight" and "early blight," "septoria leaf spot" and "leaf mold," as well as between "mites two spotted spider mite" and "target spot." Figure 3.7 presents some examples of these classes.

- Confusion rates for the second method were low, often justified by diseases sharing similar characteristics, such as the two classes “potato late blight” and “tomato late blight.” Figure 3.8 showcases examples of this confusion.

- The proposed model successfully identifies late blight disease but confuses the two plant types. The second protocol proved to be simpler compared to the first, offering shorter execution times and facilitating the model's implementation for web applications while still providing satisfactory performance, as depicted in Figure 3.6.

- Comparing our results with those in the literature, particularly in table 1.1, for the Tomato category, our model outperformed the AlexNet architecture used by the authors of [37] by 1.2% in accuracy. Similarly, with the ResNet152v2 model, we surpassed the accuracy achieved by the authors of [38] using the GoogleNet architecture for the same Tomato category, as shown in Figure 3.9. For other categories, there are yet no benchmarks for comparison.

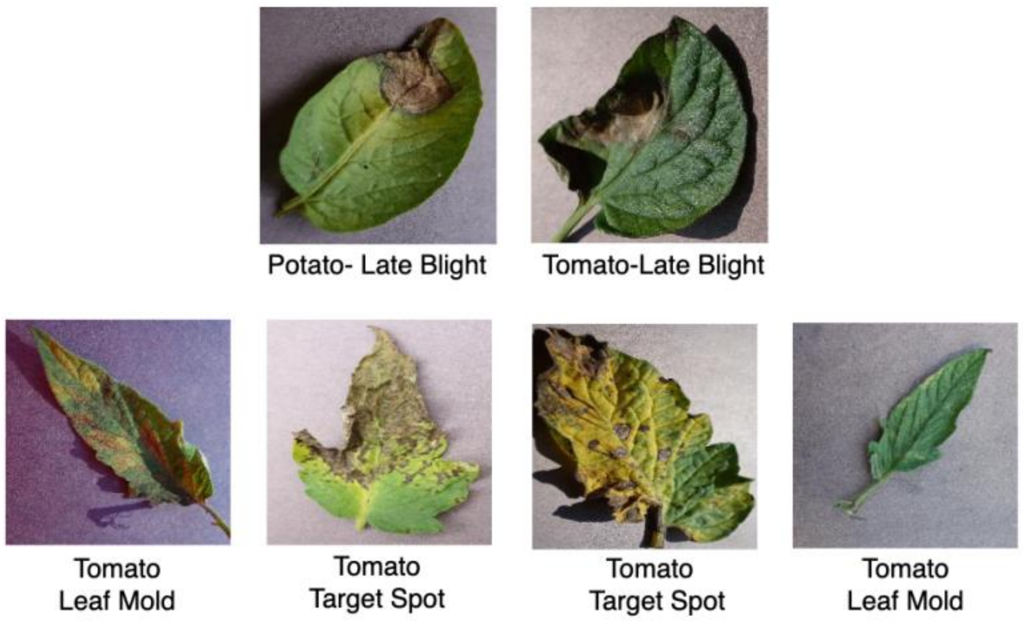


Fig. 3.7: Examples of misclassified images

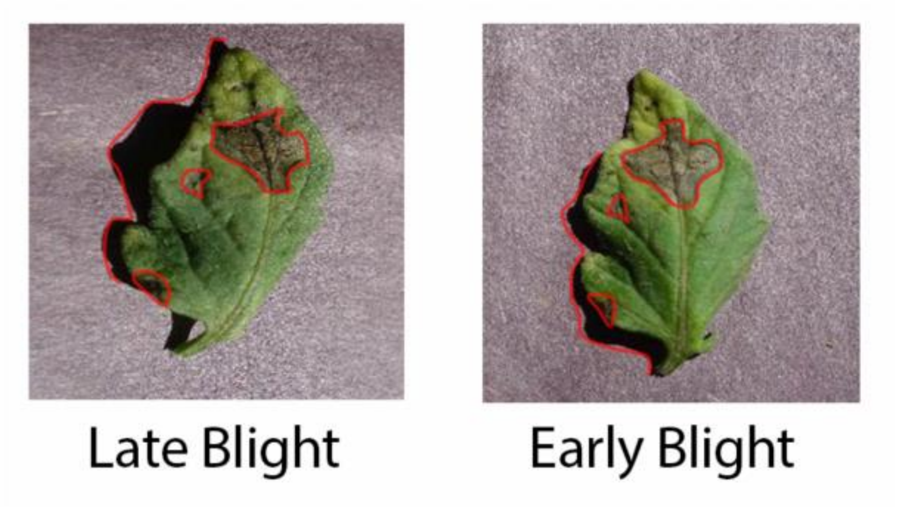


Fig. 3.8: Comparison between the two pathologies (early blight and late blight) of Tomato

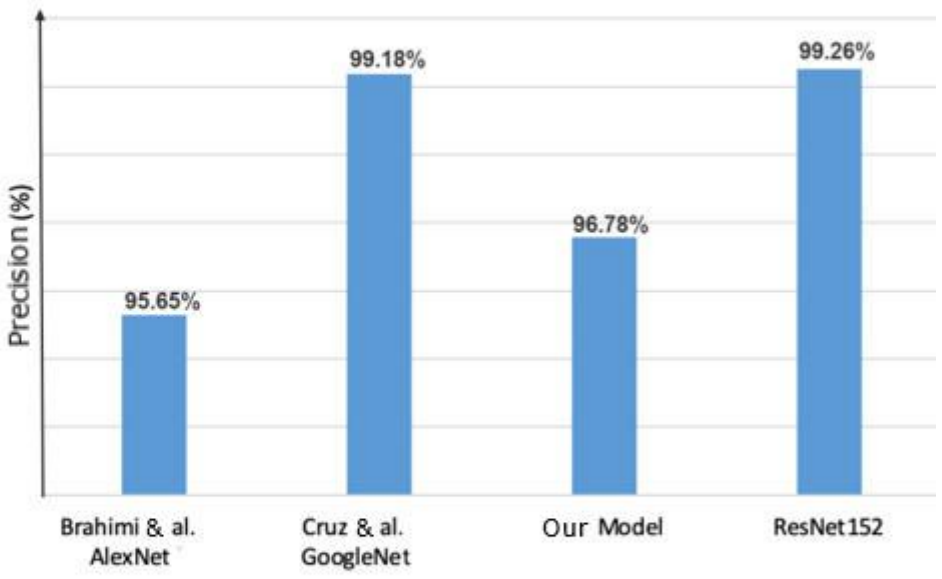


Fig. 3.9: Comparison of the results obtained with those of the state of the art (Tomato)

3.4 Conclusion

In this chapter, we have presented the tests conducted to evaluate the plant disease diagnostic system, based on convolutional neural networks (CNNs). Our work focused on studying three models that have demonstrated their effectiveness in the field of computer vision, namely ResNet152, AlexNet, and VGG16. From the experiments and tests carried out, we can conclude that successful use of CNNs requires robust computational resources and consumes memory space. Therefore, as our ultimate goal is to develop a web application, we have proposed a CNN architecture, ResNet 18, that is adapted to web requirements while still offering good performance.

In the following chapter, we will proceed with the implementation of our model on a web application, aiming to come up with a system ready to be deployed for diagnosing plant diseases.

Chapter 4

Implemention of the solution

4.1 Introduction

To start utilizing a model for practical decision-making, it must be effectively deployed in production. If reliable, practical insights cannot be consistently derived from our model, its impact is significantly limited.

Deployment is the process through which a machine learning model is integrated into an existing production environment to make practical business decisions based on data. It is one of the final stages in the machine learning lifecycle and can be among the most challenging.

Deploying the model requires coordination between data specialists, IT teams, software developers, and business professionals to ensure the model operates reliably within the organization's production environment.

The aim of this project is to develop a web application for diagnosing plant diseases based on a deep learning model. Therefore, in this chapter, we discuss the details of implementing our previously developed model into a web interface to enable effective use, simplifying the diagnosis of plant diseases and making it more accessible to users. For implementation, we opted for protocol 02 as it is faster and simpler.

4.2 Libraries and Frameworks Used

To be able to integrate the model into a web application, there are many popular frameworks that can be used to perform this task, such as Flask, Django and FastAPI [1].

Django is generally used for large applications. scale and takes a long time to set up, while Flask and Fast API are typically used to quickly deploy the model to a web application.

In this work, we have chosen the following software infrastructures (Framework):

4.2.1 FastAPI

FastAPI[2], is a modern web framework and powerful for building APIs with Python based on standard type indices. He owns the following key features:

- **Fast:** Very high performance, similar to NodeJS and Go. One of the frameworks. The fastest Pythons on the market.

- **Fast to code:** It considerably increases the development speed.

- **Reduction in the number of bugs:** It reduces the possibility of human-made errors.
- **Intuitive:** offers excellent support for the editor, with add-ons everywhere and less debugging time.
- **Simplicity:** It was designed to be simple to use and learn, so that you can spend less time reading the documentation.
- **Robust:** it provides production-ready code with documentation automatic interactive.

This software infrastructure is designed to optimize the development experience so that we can write simple code to build APIs ready to be implemented production.

4.2.2 Uvicorn

Uvicorn is an ASGI (Async Server Gateway interface) compatible web server [3]. It is the binding element that manages web connections from the browser or API client and then allows FastAPI to serve the actual request.

4.2.3 JavaScript

JavaScript is a dynamic programming language widely used for web development, web applications, game development, and more. It allows for the implementation of dynamic features on web pages that cannot be achieved with just HTML and CSS. [4]

This library has the following features:

- **Light-Weight Scripting Language:** JavaScript is lightweight, designed for data handling in web applications.
- **Dynamic Typing:** Supports dynamic typing where variable types are defined based on stored values.
- **Object-Oriented Programing Support:** Provides support for object-oriented programming principles.
- **Functional Style:** Utilizes a functional approach where functions can be used as objects and passed to other functions.

- **Independent Platform:** JavaScript is platform-independent, allowing scripts to run anywhere without affecting output.
- **Interpreted Language:** JavaScript is interpreted, processed line by line by a built-in interpreter in web browsers.
- **Single Threaded:** By default, JavaScript is single-threaded but supports async processing and web workers for parallel execution.
- **Async Processing:** Supports Promises and Async functions for asynchronous requests and processing.

JavaScript's versatility and powerful applications make it a fundamental tool for developers working on web-based projects, offering interactivity and dynamic content creation on websites.

4.1 Development

In the following section, we will delve into the most critical aspects of constructing a web application. These components are fundamental and are typically indispensable. They share some similarities with the development of mobile software, games, or other types of software. This project primarily focuses on the essential technical aspects of building a single-page web application. It's important to note that there are additional crucial aspects to consider. For instance, application security is a vast topic that cannot be extensively covered in this work but warrants mention. It's something developers should consider while developing both backend and frontend functionalities. Additionally, it's a vital component of the application's infrastructure and monitoring.

The development of our web application for diagnosing plant diseases is divided into two parts:

4.1.1 Server side (Backend)

The backend refers to the server side of the website. It primarily stores our previously developed deep learning model, along with the class names and the pre-processing operations performed on images, ensuring that everything on the client side of the website functions correctly. It's the part of the website that remains unseen and uninteractable by users, constituting the software segment not in direct contact with users. The features and components developed in the backend are indirectly accessible to users through a frontend application. Activities such as API development, library creation, and working with non-user-interface system components or even scientific programming systems are also part of the backend responsibilities.

For the development of our application's backend, we have opted for the FastAPI library in Python, as previously mentioned. This process involves several steps as follows:

Install the necessary prerequisites

Before initiating the coding phase, it's essential to install FastAPI along with other required libraries. For this purpose, we use a virtual environment, where all libraries are managed, simplifying the development and deployment process.

- **Installing FastAPI**
  FastAPI installation follows the standard procedure for any Python module. However, it lacks an integrated development server. Therefore, we will employ Uvicorn, an ASynchronous Server Gateway Interface (ASGI) server, to host FastAPI.
  The installation of both modules is achieved using the following command:

```
pip install fastapi uvicorn
```

Next, we create a directory to house all the necessary files for our web application's server side. Subsequently, we craft the main file, "main.py," which oversees all functionalities of our web application.

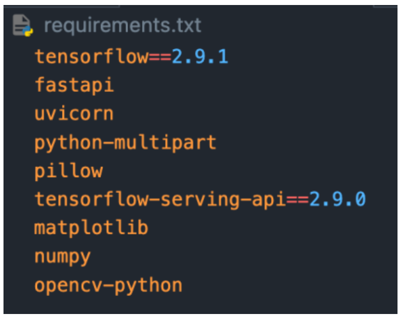
- **Testing our API**: The "main.py" file defines all path operations. To run this file, open the terminal in our directory and execute the following command:

```
uvicorn main:app—reload
```

- **Import the different libraries**
  In order for our program to work correctly, we must first import all the libraries used during the development of our CNN model, to do this, we generate a file called “requirments.txt” using the command:

```
pip install pipreqs
pipreqs Web_application/api
```

After executing the two previous commands we will have our “requirmenets.txt” file as follows:



```
requirements.txt
tensorflow==2.9.1
fastapi
uvicorn
python-multipart
pillow
tensorflow-serving-api==2.9.0
matplotlib
numpy
opencv-python
```

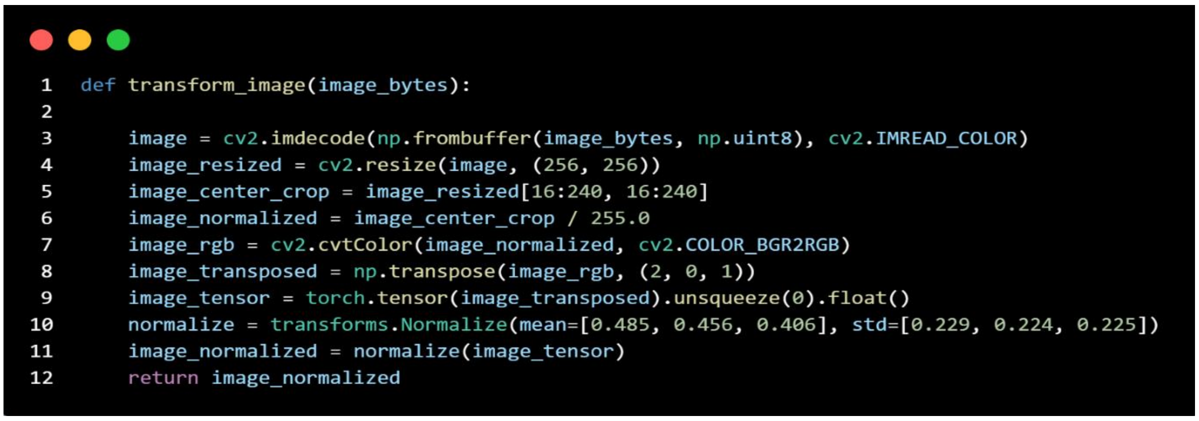
We install the libraries using the command:

```
pip install-r requirements. txt
```

Now that our development environment is ready, we approach the part server-side programming.

Image pre-processing

Before running the disease detection task, and after loading an image in the application, pre-processing was applied to these images so that our model can treat them correctly. First we use the “numpy.array()” function which takes an image as an argument and converts it to a Numpy array. Then, using the “numpy.expend_dims” function, we add a dimension of one unit to our image which is represented by an array of values in the interval [0, 255], finally, we resize these values in the interval [0, 1] to have images with the same features that our CNN was trained with as shown the code below:



```python
def transform_image(image_bytes):

    image = cv2.imdecode(np.frombuffer(image_bytes, np.uint8), cv2.IMREAD_COLOR)
    image_resized = cv2.resize(image, (256, 256))
    image_center_crop = image_resized[16:240, 16:240]
    image_normalized = image_center_crop / 255.0
    image_rgb = cv2.cvtColor(image_normalized, cv2.COLOR_BGR2RGB)
    image_transposed = np.transpose(image_rgb, (2, 0, 1))
    image_tensor = torch.tensor(image_transposed).unsqueeze(0).float()
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    image_normalized = normalize(image_tensor)
    return image_normalized
```

Diagnostic

This is where the most important step of our backend comes, this is the code part who is responsible for diagnosing different diseases.

When the user loads an image of a plant leaf, the web application must detect the type of plant in question as well as the disease associated with that type. For this, we we need the template file that we saved earlier in the same project file. To load our model file, we use the function load_model as follows:

```python
model_path = 'model.pth'
```

Now we have another element "@app.route ('/predict')", this one matches the function "predict()" with the URL /predict, the latter, like its name indicates it, takes the image given by the user, performs all the pre-processing, and passes through the different layers of the proposed CNN model, finally giving a confidence score on the belonging of this image to one of the predefined classes in using the softmax() function. Finally, we obtain the type of food as well as the associated disease. We will use it as an index to search in the "class_names" table which contains the different classes of foods that we have.
The interface on our server side (Backend):

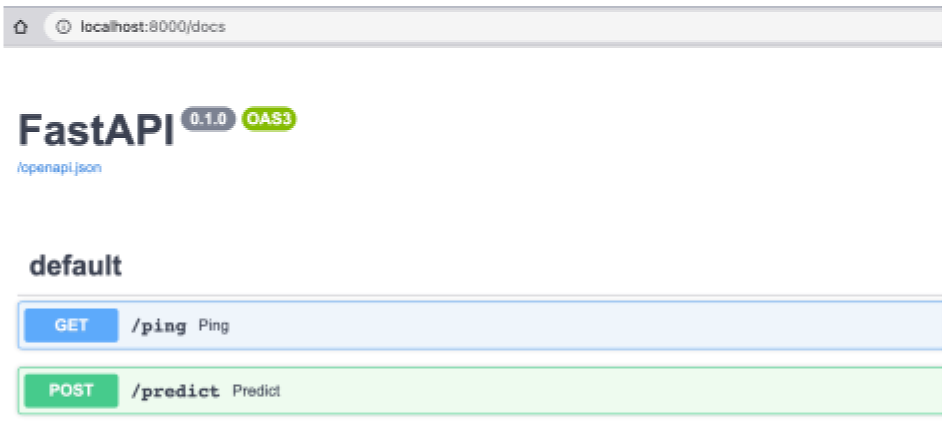
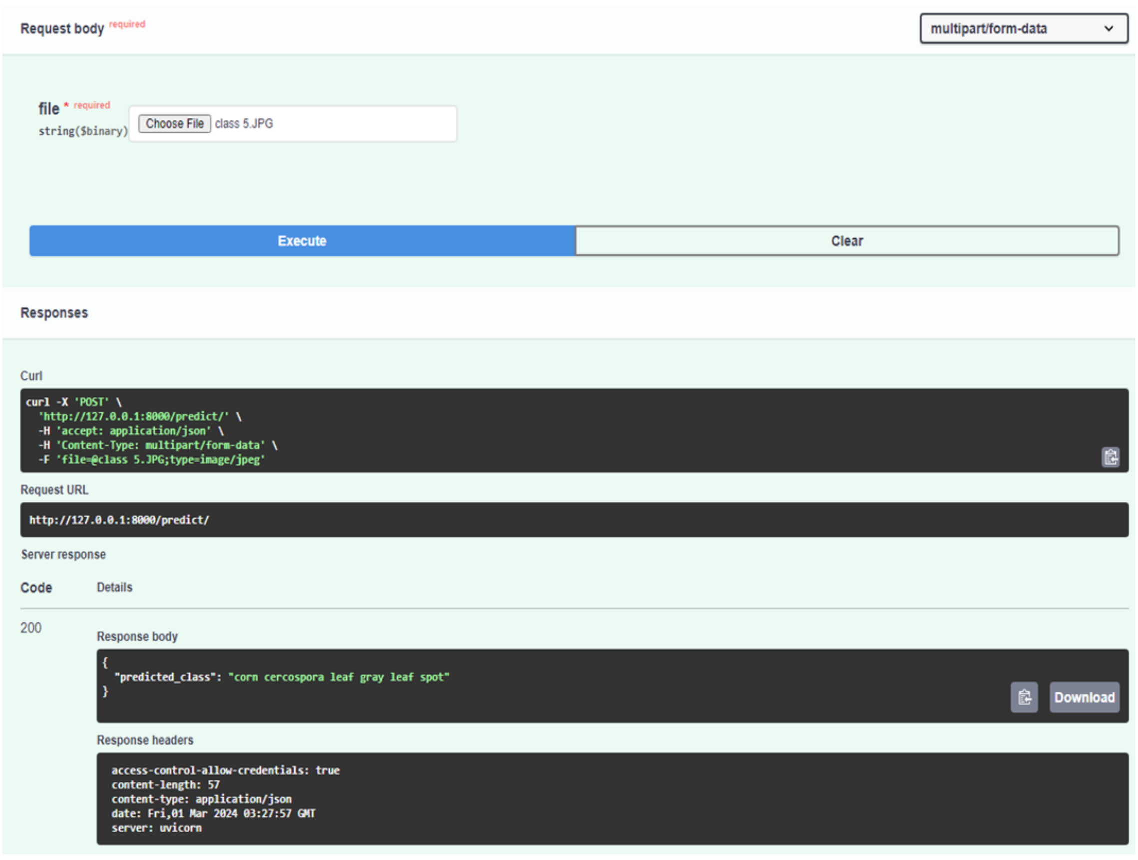


Fig. 4.1: Backend interface



Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict/' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@class 5.JPG;type=image/jpeg'
```

Request URL

```
http://127.0.0.1:8000/predict/
```

Server response

Code Details

200

Response body

```
{
  "predicted_class": "corn cercospora leaf gray leaf spot"
}
```

Response headers

```
access-control-allow-credentials: true
content-length: 57
content-type: application/json
date: Fri,01 Mar 2024 03:27:57 GMT
server: uvicorn
```

Fig. 4.2: Backend interface after execution of the predict() function

4.1.1 User side (Frontend)

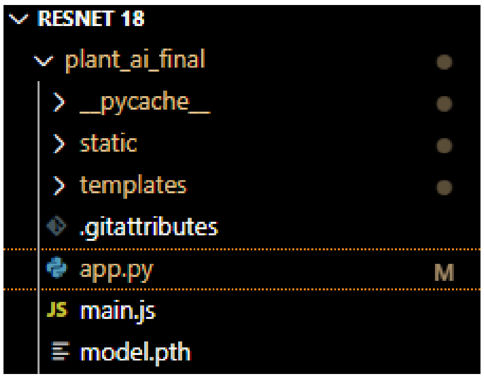
Part of a website that the user directly interacts with is called frontend. It is also called the "client side" of the application. it understands everything users see in the browser: colors and styles of text, images, graphics and tables, buttons, colors and navigation menu. HTML, CSS and JavaScript are the languages used for frontend development. The structure, design, behavior and content of everything we see on browser screens when websites, web applications or mobile applications are opened, are put into effect work by frontend developers. Responsiveness and performance are two objectives main ones of the frontend.

We need to ensure that the site is responsive, i.e. a human interface machine must be ergonomic as well as efficient. In addition, these interfaces must be easy to use and understandable by users.

As mentioned previously we will use the Js, HTML and CSS library for the design of our web interface

Directories

We create the required folders as shown in fig 4.3



```
RESNET 18
  plant_ai_final
    > __pycache__
    > static
    > templates
      .gitattributes
      app.py            M
      main.js
      model.pth
```

Fig. 4.3: Directory of the Frontend part

- **Directory static:** In this folder are placed all the images used on the site, as well as the css and webfonts formatting files.
- **Directory Templates:** Here are placed all the html files index.html and the rest of the site sections.
  - **index.html:** It is the main interface file on the site that contains images of the plants that the site supports.
  - **ai.html:** It is the page through which pictures are uploaded and plant diseases are learned.
  - **plans.html:** It is the page through which one can subscribe to the site and obtain additional benefits.
  - **project.html:** After discovering the disease, go to this page to learn about the proposed solutions for treatment and the definition of the disease.
  - **main.js:** Although we have developed the script for each section internally, we have attached some changes to this file that will facilitate the site's work and its response to the user.
  - **App.py:** This file is used to prepare the fast API and prepare the form to receive requests. This file is considered the basis of the site, without it the form will not work.

4.2.4 Testing

Now that your plant disease detection app is ready When in use, we run tests to make sure it works properly. To do this, We have to implement our backend's app.py file as well as a User interface with opening the site file ai.html from vs code Live Server After executing the above command, our UI window opens automatically as shown in Figure 4.4:

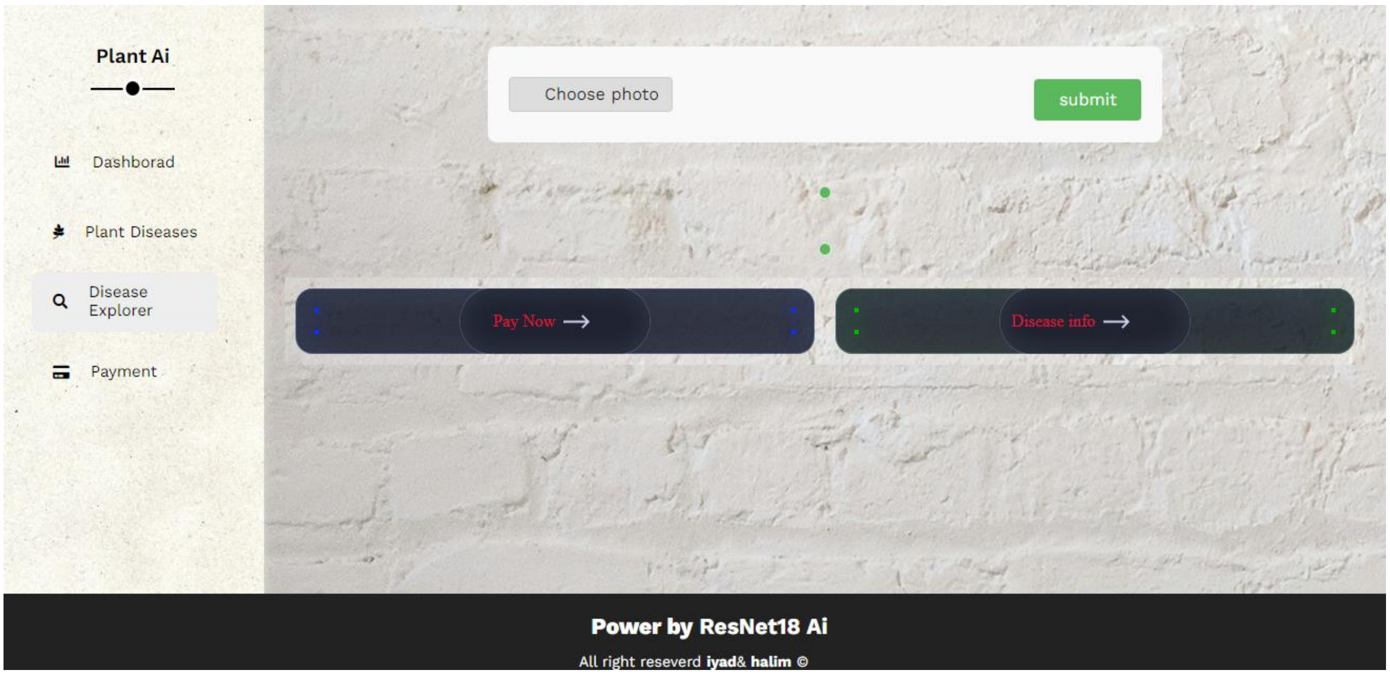


Fig. 4.4: Page part of our web interface

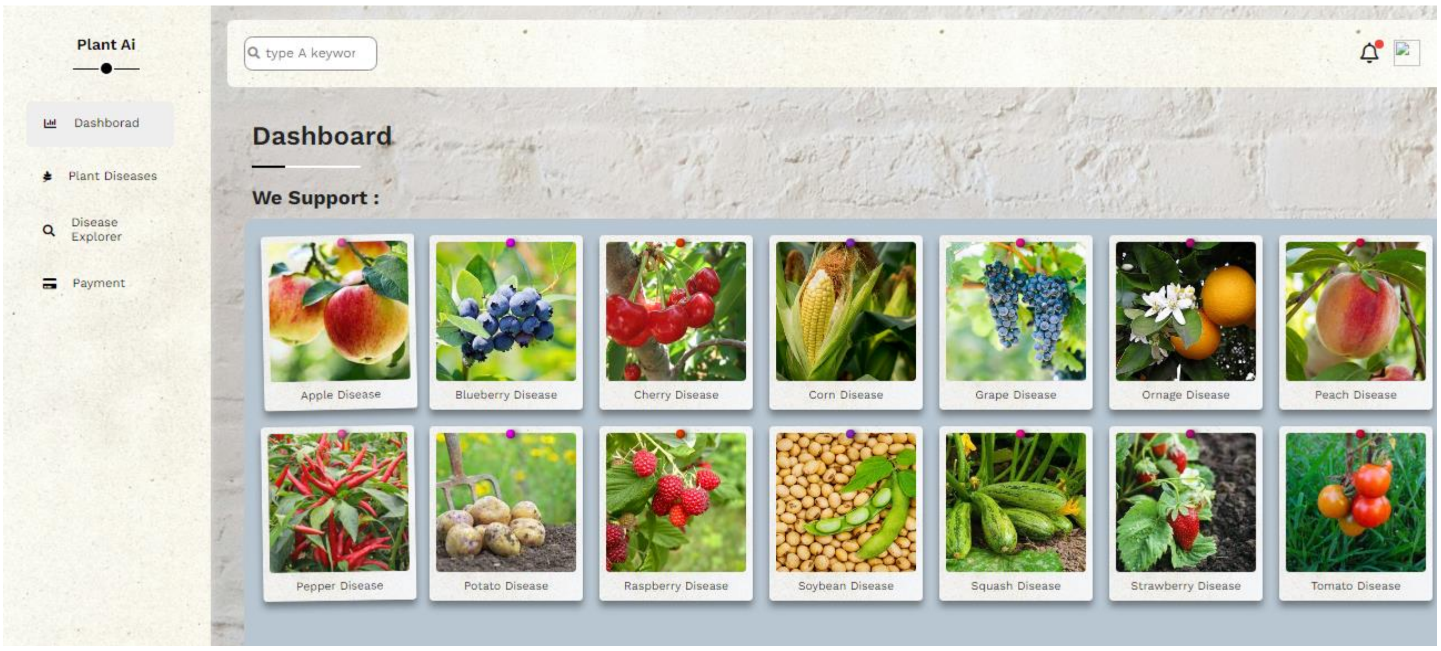


Fig. 4.5: Main page of our web interface

After downloading the image, it shows us the type of disease detected

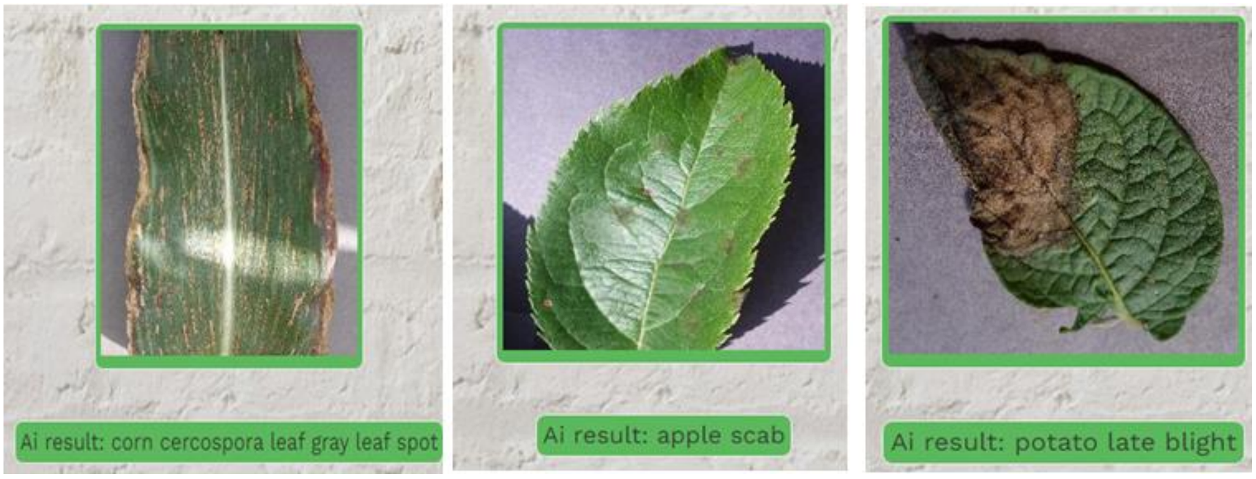

Ai result: corn cercospora leaf gray leaf spot
Ai result: apple scab
Ai result: potato late blight

Fig. 4.6: Application response after execution

4.3 Conclusion

In this chapter we presented the basic concepts that are different from each other. It follows the development of our web interface. D'autre part, on an explanation the use of infrastructure software uses Js and FastAPI to ensure this great user experience, in the but to perform the diagnostic of plant diseases (Tomato, apple, potato, orange, etc.), which translates into the implementation of our model that developed in the 03 chapiter, using the convolution neuronal network.

Conclusion and perspectives

Conclusion and perspectives

This project objective is to create a website that helps people diagnose and classify plant diseases using computer programs called convolutional neural networks. CNNs are systems that take pictures of plant leaves and decide if the plant is healthy or not. Second, we put the model that we proposed in a web interface so that it would be an effective system that users could use.

A subset of three crop types from the PlantVillage database were tested using the models AlexNet, VGG16, ResNet152, and a model we proposed. We conclude from the experiments that:

- For the detection problem, ResNet152v2 and VGG16 offer the best almost optimal performances (greater than 98%), however, these models are greedy in terms of computing capacity and hardware resources as well as in time of execution.
- The classification of several types of pathologies that can affect the same type of crop was completed with accuracies of around 96% at the level of our model, which confirms the robustness of our system for identifying the different plant diseases.
- Effective use of a CNN is an experimental design problem, where multiple tests are required to find the right configuration for the application to consider.
- It is important to adjust the input data size to the depth of the network in order to obtain discriminative features at the output of the convolution block. A network containing a small convolution block cannot train a very large image. However, if the network is very deep, the characteristics of a small image will be lost.
- The implementation of deep learning models in web applications requires a lot of space to host it and make it accessible to the public, because we need to load all the libraries used like tenserflow and keras without counting the size of the model which represents an important criterion in the choice of the last.

Finally, the results obtained can be further improved with adequate computing resources to carry out a more efficient design of deep CNN models. This research can be extended to other types of crops offered by the Plant Village base or other public bases, in order to verify the robustness of the developed systems. It would be interesting to test other CNN models, such as Exception. As possible research perspectives for the work that concerns deployment, it is very interesting that this system is implemented in a mobile application to facilitate the processing task for users thanks to the camera present in the smartphone, and without having to resort to a connection Internet.

Bibliography

Chapter 1

1. STRANGE, Richard N; SCOTT, Peter R. Plant disease: a threat to global foodsecurity. Annual review of phytopathology. 2005, vol. 43, n°1, p. 83-116.
2. EBRAHIMI, MA; KHOSHTAGHAZA, Mohammad Hadi; MINAEI, Saeid; JAM-SHIDI, Bahareh. Vision-based pest detection based on SVM classification method.*Computers and Electronics in Agriculture*. 2017, vol. 137, p. 52-58.
3. CHENG, Xi; ZHANG, Youhua; CHEN, Yiqiong; Wu, Yunzhi; YUE, Yi. Pest identification via deep residual learning in complex background. Computers and Electro-*nics in Agriculture*. 2017, vol. 141, p. 351-356.
4. Li, L., Zhang, S., & Wang, B. (2021). Plant disease detection and classification by deep learning—a review. *IEEE Access*, *9*, 56683-56698.
6. Fang, Y., & Ramasamy, R. P. (2015). Current and prospective methods for plant disease detection. *Biosensors*, *5*(3), 537-561.
7. Sankaran, S., Mishra, A., Ehsani, R., & Davis, C. (2010). A review of advanced techniques for detecting plant diseases. *Computers and electronics in agriculture*, *72*(1), 1-13.
8. Rathod, A. N., Tanawal, B., & Shah, V. (2013). Image processing techniques for detection of leaf disease. *International Journal of Advanced Research in Computer Science and Software Engineering*, *3*(11).
9. Garrett, K. A., Nita, M., De Wolf, E. D., Esker, P. D., Gomez-Montano, L., & Sparks, A. H. (2021). Plant pathogens as indicators of climate change. In *Climate change* (pp. 499-513). Elsevier.
10. Lucas, G. B., Campbell, C. L., & Lucas, L. T. (1992). *Introduction to plant diseases: identification and management*. Springer Science & Business Media.
11. Sonka, M., Hlavac, V., Boyle, R., Sonka, M., Hlavac, V., & Boyle, R. (1993). Image pre-processing. *Image processing, analysis and machine vision*, 56-111.
12. Maini, R., & Aggarwal, H. (2009). Study and comparison of various image edge detection techniques. *International journal of image processing (IJIP)*, *3*(1), 1-11.
13. Vincent, G. (2004). *Détection efficace de contours d'images* (Doctoral dissertation, Université du Québec en Outaouais).

14. Singh, V., & Misra, A. K. (2017). Detection of plant leaf diseases using image segmentation and soft computing techniques. *Information processing in Agriculture, 4*(1), 41-49.

15. Kumar, G., & Bhatia, P. K. (2014, February). A detailed review of feature extraction in image processing systems. In *2014 Fourth international conference on advanced computing & communication technologies* (pp. 5-12). IEEE.

16. Jagtap, S. B., & Hambarde, M. S. M. (2014). Agricultural plant leaf disease detection and diagnosis using image processing based on morphological feature extraction. *IOSR J. VLSI Signal Process*, *4*(5), 24-30.

17. Dayang, P., & Meli, A. S. K. (2021). Evaluation of image segmentation algorithms for plant disease detection. *Int. J. Image Graph. Signal Process*, *13*, 14-26.

18. Mohameth, F., Bingcai, C., & Sada, K. A. (2020). Plant disease detection with deep learning and feature extraction using plant village. *Journal of Computer and Communications*, *8*(6), 10-22.

19. Hossain, M. A., & Sajib, M. S. A. (2019). Classification of image using convolutional neural network (CNN). *Global Journal of Computer Science and Technology*, *19*(2).

20. Bhosle, K., & Musande, V. (2019). Evaluation of deep learning CNN model for land use land cover classification and crop identification using hyperspectral remote sensing images. *Journal of the Indian Society of Remote Sensing*, *47*(11), 1949-1958.

21. Zhang, Y. (2012). Support vector machine classification algorithm and its application. In *Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16, 2012. Proceedings, Part II 3* (pp. 179-186). Springer Berlin Heidelberg.

22. Murty, M. N., & Devi, V. S. (2011). *Pattern recognition: An algorithmic approach*. Springer Science & Business Media.

23. Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5-32.

24. Walczak, S. (2019). Artificial neural networks. In *Advanced methodologies and technologies in artificial intelligence, computer simulation, and human-computer interaction* (pp. 40-53). IGI global.

25. Asefpour Vakilian, K., & Massah, J. (2013). An artificial neural network approach to identify fungal diseases of cucumber (Cucumis sativus L.) plants using digital image processing. *Archives Of Phytopathology And Plant Protection*, *46*(13), 1580-1588.

26. Hughes, D., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. *arXiv preprint arXiv:1511.08060.*

27. Kawasaki, Y., Uga, H., Kagiwada, S., & Iyatomi, H. (2015). Basic study of automated diagnosis of viral plant diseases using convolutional neural networks. In *Advances in Visual Computing: 11th International Symposium, ISVC 2015, Las Vegas, NV, USA, December 14-16, 2015, Proceedings, Part II 11* (pp. 638-645). Springer International Publishing.

28. Fujita, E., Kawasaki, Y., Uga, H., Kagiwada, S., & Iyatomi, H. (2016, December). Basic investigation on a robust and practical plant diagnostic system. In *2016 15th IEEE international conference on machine learning and applications (ICMLA)* (pp. 989-992). IEEE.
29. Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., & Stefanovic, D. (2016). Deep neural networks based recognition of plant diseases by leaf image classification. *Computational intelligence and neuroscience, 2016*.
30. Amara, J., Bouaziz, B., & Algergawy, A. (2017). A deep learning-based approach for banana leaf diseases classification. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband*.

31. Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., & Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing, 234*, 11-26.
32. Zhang, X., Qiao, Y., Meng, F., Fan, C., & Zhang, M. (2018). Identification of maize leaf diseases using improved deep convolutional neural networks. *Ieee Access, 6*, 30370-30377.
33. Rangarajan, A. K., Purushothaman, R., & Ramesh, A. (2018). Tomato crop disease classification using pre-trained deep learning algorithm. *Procedia computer science, 133*, 1040-1047.

Chapter 2

1. Zouinar, M. (2020). Évolutions de l'Intelligence Artificielle: quels enjeux pour l'activité humaine et la relation Humain-Machine au travail?. *Activités*, (17-1).
2. Ciresan, D., Giusti, A., Gambardella, L., & Schmidhuber, J. (2012). Deep neural networks segment neuronal membranes in electron microscopy images. *Advances in neural information processing systems, 25*.
3. Liu, X., Deng, Z., & Yang, Y. (2019). Recent progress in semantic image segmentation. *Artificial Intelligence Review, 52*, 1089-1106.
4. Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review, 53*, 5455-5516.
5. Jha, G. K. (2007). Artificial neural networks and its applications. *IARI, New Delhi, girish_iasri@ rediffmail. com.*
6. Gupta, N. (2013). Artificial neural network. *Network and Complex Systems, 3*(1), 24-28.

8. Decaro, C., Montanari, G. B., Molinari, R., Gilberti, A., Bagnoli, D., Bianconi, M., & Bellanca, G. (2019). Machine learning approach for prediction of hematic parameters in hemodialysis patients. *IEEE journal of translational engineering in health and medicine, 7*, 1-8.

9. Dongare, A. D., Kharde, R. R., & Kachare, A. D. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, *2*(1), 189-194.
10. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533-536.
12. Hubel, D. H., & Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, *195*(1), 215-243.
13. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.
14. LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, *2*.
15. Zhang, W., Itoh, K., Tanida, J., & Ichioka, Y. (1990). Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied optics*, *29*(32), 4790-4797.
16. Pound, M. P., Atkinson, J. A., Townsend, A. J., Wilson, M. H., Griffiths, M., Jackson, A. S., ... & French, A. P. (2017). Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *Gigascience*, *6*(10), gix083.
17. Wang, W., Yang, Y., Wang, X., Wang, W., & Li, J. (2019). Development of convolutional neural network and its application in image classification: a survey. *Optical Engineering*, *58*(4), 040901-040901.
18. Shyam, R. (2021). Convolutional neural network and its architectures. *Journal of Computer Technology & Applications*, *12*(2), 6-14p.
19. Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017, August). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)* (pp. 1-6). Ieee.
20. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.
21. Shyam, R. (2021). Convolutional neural network and its architectures. *Journal of Computer Technology & Applications*, *12*(2), 6-14p.
23. Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310-316.
24. Job, M. S., Bhateja, P. H., Gupta, M., Bingi, K., & Prusty, B. R. (2022). Fractional Rectified Linear Unit Activation Function and Its Variants. *Mathematical Problems in Engineering*, *2022*.
25. Bjorck, N., Gomes, C. P., Selman, B., & Weinberger, K. Q. (2018). Understanding batch normalization. *Advances in neural information processing systems*, *31*.
26. Baldi, P., & Sadowski, P. J. (2013). Understanding dropout. *Advances in neural information processing systems*, *26*.
27. Brahimi, M., Boukhalfa, K., & Moussaoui, A. (2017). Deep learning for tomato diseases: classification and symptoms visualization. *Applied Artificial Intelligence*, *31*(4), 299-315.
28. Cruz, A. C., Luvisi, A., De Bellis, L., & Ampatzidis, Y. (2017). Vision-based plant disease detection system using transfer and deep learning. In *2017 asabe annual international meeting* (p. 1). American Society of Agricultural and Biological Engineers.

Webography

CHAPTER 1

5- Examples of leaves representing diseased plants – PlantVillage https://www.kaggle.com/datasets/emmarex/plantdisease

CHAPTER 2

7- Biological neuron models — https://en.wikipedia.org/wiki/Biological_neuron_model

11- https://medium.com/swlh/cyclical-learning-rates-the-ultimate-guide-for-setting-learning-rates-for-neural-networks-3104e906f0ae

22- CNN Settings — https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26

27- Model 01: AlexNet — https://www.geeksforgeeks.org/ml-getting-started-with-alexnet/?ref=header_search

28- Model 01: AlexNetArchitecture — https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951

29- Model 02: VGG16 — https://datascientest.com/quest-ce-que-le-modele-vgg

30- Model 02: ResNet152v2 — https://www.geeksforgeeks.org/introduction-to-residual-networks/?ref=lbp

31- Transfer Learning — https://www.geeksforgeeks.org/ml-introduction-to-transfer-learning/

CHAPTER 3

1- PlantVillage Dataset | Kaggle [https://bit.ly/3m2HYNV], https://bit.ly/3SKf6tW.

2- Python

What is Python ? Executive Summary | Python.org [https://bit.ly/3ze7tUw].

3- Torchvision [https://bit.ly/3IfsOAc]

4-

5- Colab : A Modern PaaS for Machine Learning – The New Stack [https://bit.ly/49tzaYw].

CHAPTER 4

1- Deploying ML Models as API using FastAPI – GeeksforGeeks [https://bit.ly/ 3yhJWRG]. [s. d.].

2- FastAPI [https://bit.ly/3nii6hS]. [s. d.].

3- Uvicorn [https://bit.ly/3xUeHL1]. [s. d.]

4- javascript definition [https://bit.ly/3V03oxT] [s. d.]