University of Padova

Department of Department of Mathematics

Master Thesis in Master degree in Data Science

Identification of infesting plants in monoculture fields through CNNs from UAV imagery

Supervisor
Lamberto Ballan
University of Padova

Co-supervisor
Stefano Bazzolo
Dromt S.r.l.

Master Candidate
Francesco Vo

Academic Year

2023-2024

For my family, that always supported me.

Abstract

The identification of infesting plants in monoculture fields is a critical task for enhancing crop yields and promoting sustainable agriculture. This thesis investigates the use of Convolutional Neural Networks (CNNs) for the automated detection and classification of infesting plants in agricultural settings using high-resolution imagery obtained from Unmanned Aerial Vehicles (UAVs). Various CNN architectures, including ResNet50, VGG16, InceptionV3, Inception-ResNetV2, and DenseNet201, were evaluated to determine their effectiveness in distinguishing between crops and infesting plants.
The study demonstrates that DenseNet201 outperforms other models, achieving the highest accuracy due to its dense connectivity pattern, which ensures efficient gradient flow and feature reuse. This architecture's ability to learn complex and abstract features makes it particularly adept at handling the variability and complexity present in UAV imagery.
The thesis addresses the challenges of dataset preparation, including data augmentation and class imbalance, and highlights the potential of integrating UAVs and CNNs for real-time agricultural monitoring. The impact of false positives (FP) and false negatives (FN) is also carefully examined, given their significant implications in crop monitoring, by introducing methodologies to control the values of precision and recall.
The results indicate that this approach can significantly reduce labor costs and enhance crop management efficiency. Future research directions include improving model generalization through diverse datasets, exploring advanced CNN architectures, developing real-time processing methods, and integrating automated intervention systems. By addressing these areas, the research aims to contribute to more effective and sustainable agricultural practices.

Contents

Listing of figures

Listing of tables

Listing of acronyms

NDVI	Normalized Difference Vegetation Index
VARI	Visible Atmospherically Resistant Index
SAVI	Soil Adjusted Vegetation Index
UAV	Unmanned Aerial Vehicle
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
ML	Machine Learning
DL	Deep Learning
SVM	Support Vector Machine
GLM	Generalized Linear Model
TP	True Positive
FP	False Positive
FN	False Negative
VGGNet	Visual Geometry Group Network
YOLO	You Only Look Once
CPU	Central Processing Unit
GPU	Graphics Processing Unit

1 Introduction

1.1 Introduction

Monoculture farming, the practice of growing a single crop species over a large area, has become a prevalent agricultural method due to its efficiency and economic benefits. However, this practice also presents significant challenges, particularly the infestation of fields by unwanted plant species.
These infesting plants can severely impact crop yields by competing for resources such as water, nutrients, and light. Effective identification and management of these infesting plants are crucial for maintaining high productivity in monoculture fields.
Traditional methods of infesting plant identification and control are labor-intensive and time-consuming because they rely heavily on visual inspection by experts.
The advent of advanced technologies in the field of computer vision and machine learning, particularly Convolutional Neural Networks (CNNs), offers a promising alternative for addressing this issue. CNNs, a class of deep learning models, have demonstrated remarkable performance in various image recognition tasks, making them suitable for identifying different plant species in agricultural settings [1].
Additionally, the integration of drone technology and Unmanned Aerial Vehicles (UAVs) has further enhanced the capabilities of these models. UAVs equipped with high-resolution cameras can capture detailed images of large agricultural fields, providing extensive and up-to-date

data for analysis. This synergy between CNNs and UAVs facilitates precise and efficient identification of infesting plants, enabling timely interventions and more effective infesting plant management strategies [2].

1.1.1 Background and motivation

The need for precise and efficient identification of infesting plants in monoculture fields has led to significant research interest in the application of CNNs. Unlike traditional image processing techniques, CNNs can automatically learn and extract features from images, enabling them to differentiate between crop and infesting plants with high accuracy. This capability is particularly beneficial in complex field environments where plant species can exhibit significant visual similarities and variations due to factors such as lighting conditions and plant growth stages.
The integration of UAVs into this framework has further revolutionized agricultural monitoring and management. UAVs equipped with high-resolution cameras and advanced sensors can capture comprehensive images of agricultural fields from various altitudes and angles. This aerial perspective allows for large-scale data collection in a relatively short period, significantly enhancing the efficiency and coverage of field monitoring.
UAVs provide several advantages over traditional ground-based methods. They can access difficult-to-reach areas, fly over fields without disturbing crops, and capture images under varying light conditions, ensuring consistent data quality. The high-resolution images obtained from UAVs can then be processed using CNNs to identify and classify infesting plants with high precision.
Recent studies have demonstrated the effectiveness of UAVs in agricultural applications. For example, Lottes et al. (2018) [3] utilized UAV-based imagery combined with CNNs for crop and infesting plant classification, achieving significant improvements in detection accuracy and operational efficiency. Similarly, Zhang and Kovacs (2012) [2] reviewed the application of small UAVs in precision agriculture, highlighting their potential to transform traditional farming practices by providing detailed and timely information on crop health and infesting plants.
This combination of UAV technology and CNNs enables the development of robust systems for real-time infesting plant detection and management.

1.1.2 Objectives

The primary objective of this thesis is to develop and evaluate a CNN-based approach for the identification of infesting plants in monoculture fields. This involves several specific aims:

1. Dataset creation: develop a comprehensive dataset comprising images of crops and various infesting plant species in a monoculture field. This dataset will include annotated examples for training and validation of the CNN model.

2. Model selection: design and train a CNN model capable of accurately classifying and differentiating between crops and infesting plants. The model should be robust enough to handle variations in image conditions such as lighting and plant growth stages.

3. Transfer learning: utilize transfer learning techniques to enhance the model's performance. By leveraging pre-trained models on large-scale image datasets, the CNN can achieve better generalization and faster convergence.

4. Performance evaluation: assess the performance of the CNN model in real-world field conditions. This will involve deploying the model on UAV-captured images and comparing its predictions against ground truth data.

5. False positives and false negatives study: conduct a thorough analysis of the model's false positive and false negative rates. Understanding the instances where the model incorrectly identifies infesting plants (false positives) or fails to detect them (false negatives) is crucial for improving its accuracy and reliability. This study will help in identifying the limitations of the current approach and in developing strategies to mitigate such errors.

6. Real-time application potential: explore the feasibility of integrating the CNN model with UAV technology for real-time monitoring and management of infesting plants. This includes evaluating the computational efficiency of the model and its ability to provide timely and actionable insights for farmers.

1.1.3 Contributions

By opting for a classification approach and utilizing transfer learning (as explained in 3), this thesis tries to navigate these challenges effectively, by finding a viable path for developing efficient and scalable solutions for anomaly detection in agricultural settings.
One of the most significant contributions of this thesis is the creation of a novel dataset specifically tailored for our research objectives. The development of this dataset was not merely a preparatory step; it constituted a core part of the research, providing a foundation upon which all subsequent analyses and conclusions were built.
Utilizing drone technology presented several advantages, however, it also introduced significant challenges, particularly regarding the altitude from which the photos were taken. UAV

imaging is critical for capturing comprehensive and broad-view data, but it also complicates the tasks of image processing and feature extraction.
This research contributes to the field by offering insights into the practical application of pre-trained models for aerial image classification, providing a foundation for future studies aiming to enhance the automation and accuracy of plant disease and infestation detection.

1.1.4 Structure of the thesis

The thesis is organized as follows: Chapter 2 is a theoretic overview on machine and deep learning. Chapter 3 introduces the problem and the state of the art, and then describes the methodology, including data collection, model architecture, and training procedures. Chapter 4 presents the experimental results and performance evaluation of the proposed model. Chapter 5 discusses the findings, implications, and potential limitations of the study. Finally 6 concludes the thesis and outlines directions for future research.

1.2 Hosting company: Dromt S.r.l.

Dromt is an innovative startup founded in 2021, aiming to revolutionize the commercial drone sector by transforming drones into autonomous, intelligent, and highly functional tools. The company's mission is to simplify and automate the use of drones for a wide range of applications, making them accessible even to those without specific piloting skills.
The heart of Dromt's innovation is its all-in-one platform, designed to be compatible with most commercial drones available on the market. This platform integrates both flight management and image analysis, providing an intuitive user interface and fully automated flight management. From flight planning to takeoff, from flight to image collection, and from landing to automatic data download, every aspect is handled effortlessly through the platform.
One of the most innovative aspects of the Dromt system is the use of GPS combined with advanced computer vision-based navigation technologies. This allows drones to operate in areas where the GPS signal is weak or absent, ensuring precision and reliability. This technology is particularly useful in complex scenarios, such as densely populated urban areas or geographically challenging regions, where precise navigation is crucial.
In the agricultural sector, Dromt has found one of its first and most effective fields of application. The platform enables farmers to use drones to monitor vast agricultural fields, identifying areas of water stress, plant nutrition issues, or infestations. The acquired images are processed to produce vegetation indices, such as NDVI (Normalized Difference Vegetation Index) and VARI (Visible Atmospherically Resistant Index), and the generated prescription maps guide agricultural machinery to distribute resources like water and fertilizers optimally, thus improving crop yields and reducing waste and environmental impacts.
Simultaneously, Dromt is developing applications in other sectors. In the photovoltaic market, the platform is used to identify hotspots and other anomalies in solar panels, which can cause a reduction in energy efficiency. In the construction sector, drones equipped with Dromt technology can monitor the progress of work, identify potential structural issues, and enhance work safety. In the field of emergency management, drones can be employed to survey areas affected by natural disasters or accidents, providing crucial images for rescue and recovery operations.
Dromt does not limit itself to technology; it is also committed to continuous research and development to improve its solutions and expand its applications. With a constant commitment

to innovation and sustainability, Dromt is charting a new course for the use of drones in the commercial and industrial world, promoting not only operational efficiency but also environmental responsibility.

2
Theorical overview

2.1 Artificial Neural Networks

2.1.1 Introduction to ANNs

An artificial neural network (ANN) is a computational model inspired by the way biological neural networks in the human brain process information. The essence of an ANN is to abstract the functioning of the brain's neural network and use this abstraction to process information, learn, and make decisions.
The key components of a ANN are:

- Input units: units that receive signals from the external environment.
- Hidden units: these neurons are located between input and output units and are not directly observable from outside the network. They play a critical role in the internal processing of information.
- Output units: these neurons produce the final output of the network.

Neurons are connected through synapses, each connection having a specific weight. The weight is akin to the memory of the neural network and determines the strength and influence of one neuron on another.

Adjusting these weights during the learning process allows the network to make better predictions or classifications.
Each neuron processes the input it receives using an activation function, which determines whether it should be activated (fire) or not. This function introduces non-linearity into the network, enabling it to solve complex problems. [4]

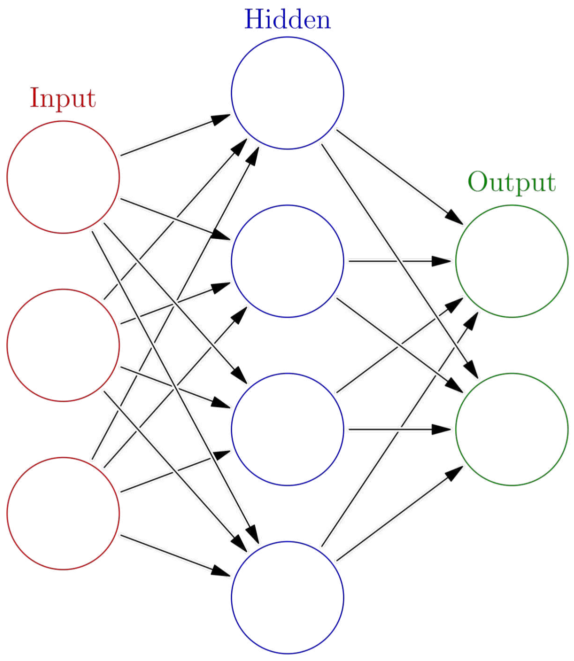


Figure 2.1: Architecture of a ANN.

2.1.2 History of ANNs

The research on Neural Network started when the M-P Model (1943), proposed by McCulloch and Pitts, marked the beginning of neural network theory. Hebb's Rule (1949) suggested that synaptic strength increases when neurons fire together, forming the basis for learning and memory in neural networks.
Rosenblatt introduced the Perceptron (1957), demonstrating the ability to classify input vectors. However, Minsky and Papert, highlighted limitations of simple perceptrons, particularly their inability to solve non-linear problems like XOR.
After that publication the research on artificial neural networks suffered a heavy blow.
Things started to get better when Hopfield introduced the Hopfield Network (1982), using

energy functions to prove network stability. The backpropagation algorithm was developed in the same period (1986), enabling multi-layer networks to adjust weights effectively.
Today, research in neural networks is flourishing, due to the development of deep learning (2006), proposed by Hinton. Deep learning is to construct a machine learning architecture model with multiple hidden layers, and a large amount of more representative characteristic information is obtained through large-scale data training. [4]

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of Artificial Neural Network (ANN) specifically designed for image pattern recognition tasks. CNNs have gained prominence due to their impressive performance in these tasks.
The main advantage of CNNs over traditional ANNs is that CNNs reduce the number of parameters, preventing overfitting. The architecture of a Convolutional Neural Network consists of three main layers:

Convolutional layers Convolution is a fundamental operation in signal processing and machine learning, particularly within convolutional neural networks (CNNs). In image processing, the discrete convolution operation on a two dimensional array I, like an image, with a kernel matrix K, can be written as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.1}$$

The kernel or filter is a smaller $2D$ matrix of learnable parameters. During the training process, the values of K are adjusted to capture essential features such as edges, textures, or more complex patterns. The kernel K slides over the input image I from the top-left to the bottom-right. At each position (i, j), the element-wise multiplication of the kernel values and the corresponding input values is computed and summed up to produce the output value $S(i, j)$.
The result of the convolution operation is a new 2D array S, often referred to as the feature map or activation map. Each value $S(i, j)$ in this map indicates the presence of a particular feature detected by the kernel at position (i, j) [5].

Pooling layers Pooling layers perform downsampling operations. The primary function of pooling layers is to reduce the spatial dimensions of the feature maps, thereby decreasing the

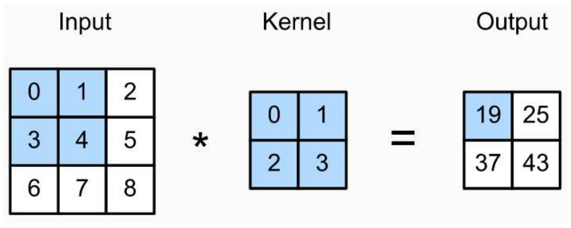


Figure 2.2: 2D convolution operation example.

computational load and the number of parameters in the network. This reduction is achieved without significantly losing the essential features captured by the convolutional layers. Common pooling operations include max pooling, which selects the maximum value from a set of neighboring pixels, and average pooling, which computes the average value. These operations help in making the representation more manageable and robust to variations in the input data, such as translations and distortions.

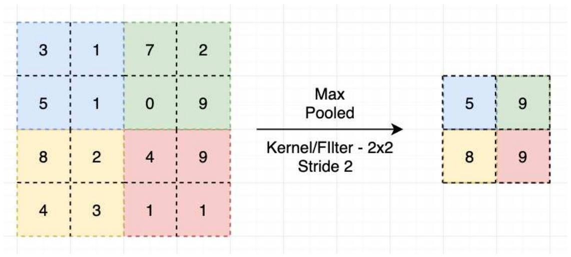


Figure 2.3: Pooling layer. Example of max pooling.

Fully connected layers Fully-connected layers, also known as dense layers, are analogous to the layers found in traditional artificial neural networks (ANNs) 2.1. In a CNN, fully-connected layers are typically placed towards the end of the architecture. They serve the critical function of combining the features extracted by the convolutional and pooling layers to produce the final output. Each neuron in a fully-connected layer is connected to every neuron in the previous layer, allowing for a comprehensive integration of the learned features. This connectivity enables the network to perform tasks such as classification or regression by mapping the high-level features to the desired output classes or continuous values.

Non-linearity A convolutional layer usually applies convolution followed by a Rectified Linear Unit (ReLU), which is an activation function used to introduce non-linearity into the model. The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x)$$

This activation function sets all negative values to zero, allowing the model to capture non-linearities while maintaining computational efficiency.
After the convolutional and ReLU layers, the final layer of a convolutional neural network often uses the softmax function to convert the output logits into probabilities for classification tasks. The softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

where z_i represents the input logits, and the denominator is the sum of exponentials of all logits z_j. The softmax function ensures that the output probabilities are in the range $[0, 1]$ and sum to 1, making them suitable for multi-class classification problems.

CNN ARCHITECTURE A typical CNN architecture involves stacking convolutional layers interspersed with ReLU activations and pooling layers, followed by fully-connected layers to generate the final classification output.
CNNs reduce the number of parameters needed compared to traditional ANNs, making them less prone to overfitting and more efficient for image data. They can be resource-heavy, especially for large images, requiring careful design choices like resizing input images or adjusting filter sizes and strides. [6]

2.3 Image processing

Recent advancements in digital image processing have increasingly leveraged neural networks due to their robust capabilities in handling complex data and performing various tasks without the need for prior assumptions about input distributions.
Egmont-Petersen, de Ridder, and Handels (2002) [7] provide a comprehensive review of over 200 applications of neural networks in this domain, highlighting the roles of feed-forward neural networks. They propose a novel two-dimensional taxonomy that categorizes image processing algorithms by the type of task and the abstraction level of the input data. This taxonomy encompasses six major tasks: preprocessing, data reduction/feature extraction, segmentation, object recognition, image understanding, and optimization. The different steps in image processing are:

1. Preprocessing: set of operations that give as a result a modified image with the same dimensions as the original image (e.g., noise reduction).

2. Feature extraction: any operation that extracts significant components from an image. The number of extracted features is generally smaller than the number of pixels in the input window.

3. Segmentation: any operation that partitions an image into regions that are coherent with respect to some criterion.

4. Object detection and recognition. Determining the position and, possibly, also the orientation and scale of specific objects in an image, and classifying these objects.

5. Image understanding: obtaining high level (semantic) knowledge of what an image shows.

2.3.1 Image pre-processing

Pre-processing consists of any operation of which the input consists of sensor data, and of which the output is a full image. Pre-processing operations generally fall into one of three categories: image reconstruction (to reconstruct an image from a number of sensor measurements), image restoration (to remove any aberrations introduced by the sensor, including noise) and image enhancement (accentuation of certain desired features, which may facilitate later processing steps such as segmentation or object recognition). [7]
The techniques for color image processing can be divided in classical and non-classical approaches. Classical methods involve filtering techniques, that comprise methods such as vector directional filters (VDF) and multichannel edge-enhancing filters (MEEF) which are used for noise removal and edge enhancement, and segmentation like graph-theoretic approaches, mean shift analysis (MS), and Markov random field (MRF) models. Statistical mixture models like Gaussian and Dirichlet mixtures are also utilized for modeling image data distributions.
Non-classical approaches encopass Neural Network based approaches, techniques such as CNN multilayer structures, competitive learning (CL), and self-organizing maps (SOM), and Wavelet based approaches: multi-resolution analysis (MRA) using wavelets aids in signal representation, noise removal, and feature extraction. [8]

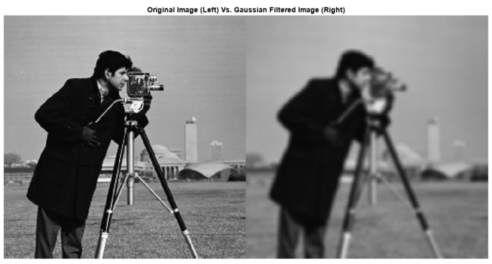


Figure 2.4: On the right: original image. On the left: image processed using the Gaussian filter.

2.3.2 Feature extraction

Feature extraction can be seen as a special kind of data reduction of which the goal is to find a subset of informative variables based on image data. Since image data are by nature very high dimensional, feature extraction is often a necessary step for segmentation or object recognition to be successful.
Besides lowering the computational cost, feature extraction is also a means for controlling the curse of dimensionality. When used as input for a subsequent classification algorithm, one wants to extract those features that preserve the class separability well. [7]
Features are divided into two main categories: local features, that include geometric attributes such as concave/convex parts, number of endpoints, branches, and joints, and global features, topological (e.g., connectivity, projection profiles, number of holes) and statistical properties (e.g., invariant moments). [9]

2.3.3 Image classification

One of the main problems in computer vision is the image classification problem, which is concerned with determining the presence of visual structures in an input image. Image classification is a task in machine learning where the goal is to categorize an entire image into one of several predefined classes. [10] The process usually involves:

1. Input: the input to the system is an image, which can be represented as a matrix of pixel values.

2. Feature Extraction: using techniques like Convolutional Neural Networks, features are extracted from the image. CNNs apply convolutional filters to the image to detect patterns such as edges, textures, and more complex structures in deeper layers.

3. Classification: the extracted features are fed into a classifier (typically a fully connected layer followed by a softmax function) to assign the image to a specific class label.

Image classification can be categorized into three main types based on the nature of the labels:

- Binary classification, which involves categorizing images into one of two possible classes.
- Multiclass classification deals with categorizing images into one of several classes. Unlike binary classification, multiclass classification involves more than two categories.
- Multilabel classification allows for each image to be assigned multiple labels. This type is essential in scenarios where an image can belong to more than one category simultaneously.

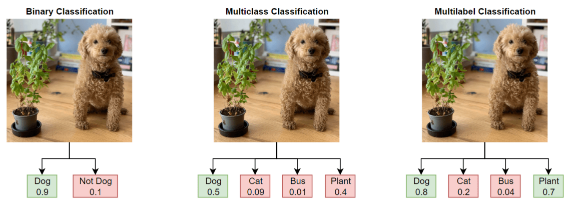


Figure 2.5: Example of different types of image classification. From left to right: binary classification, multiclass classification, multilabel classification.

2.4 Fine-tuning

2.4.1 Introduction

While training deep neural networks from scratch can lead to state-of-the-art results, it often requires large datasets and extensive computational resources. Fine-tuning offers an efficient alternative, enabling the adaptation of pre-trained models to new tasks with less data and reduced computational overhead.

Fine-tuning leverages pre-trained models that have already learned a wide array of features from large datasets. This approach is motivated by the idea that early layers of neural networks capture generic features (such as edges in images) that are applicable across various tasks, while later layers capture task-specific features. By reusing the early layers and only training the later ones, fine-tuning can significantly accelerate the training process and improve performance on target tasks with limited data.

Pre-trained models serve as the backbone for fine-tuning. Transfer learning, the broader framework encompassing fine-tuning, involves transferring knowledge from one domain (source) to another (target). The success of transfer learning hinges on the similarity between the source and target tasks.
Research by Yosinski (2014) [11] highlights that the effectiveness of transfer learning depends on the depth at which transfer occurs, with deeper layers being more specific to the source task. They also found that transferring features from earlier layers is more beneficial for dissimilar target tasks.

2.4.2 Fine-tuning techniques

The techniques used in fine tuning are: feature extraction, full model fine-tuning and layer wise fine-tuning.
Feature extraction involves freezing the weights of the pre-trained model and only training the final classifier. This approach is effective when the new dataset is small or similar to the original dataset. Donahue (2014) [12] demonstrated the power of feature extraction in computer vision tasks, where a pre-trained CNN was used to extract features for various image classification tasks, yielding significant performance improvements with minimal training.
Full model fine-tuning involves unfreezing all or most of the layers of the pre-trained model and retraining them on the new dataset. This method is beneficial when the new dataset is large enough to support extensive retraining or when the target task is substantially different from the original task. Howard and Ruder (2018) [13] popularized this approach with the Universal Language Model Fine-tuning method, which achieved state-of-the-art results in text classification by fine-tuning all layers of a pre-trained language model on target tasks.
Layer-wise fine-tuning involves gradually unfreezing layers of the pre-trained model and training them incrementally. This technique helps stabilize training and prevent catastrophic forgetting, where the model loses knowledge of the pre-trained features. This approach was extensively analyzed by Felbo (2017) [14] in their work on sentiment analysis and emotion detection in text, demonstrating its effectiveness in retaining pre-trained knowledge while adapting to new tasks.
In computer vision, fine-tuning has been widely adopted for tasks such as object detection, image segmentation, and face recognition. For instance, He (2016) [15] demonstrated the use of fine-tuning in the ResNet architecture, achieving remarkable improvements in image classification and object detection tasks.

2.5 Anomaly detection

Anomaly detection, also known as outlier detection, is a critical task in data science and machine learning that involves identifying rare items, events, or observations that differ significantly from the majority of the data. These anomalies can indicate significant but rare events, such as fraud detection, network security breaches, equipment failures, or medical condition deviations. The primary goal is to discover patterns in data that do not conform to expected behavior.

Anomaly detection is formally defined as the identification of data points that deviate from a well-defined notion of normal behavior. According to Chandola et al. (2009) [16], an anomaly is an observation that is inconsistent with the rest of the data. This inconsistency can arise due to various reasons, including inherent variability in the data, rare but significant events, or even errors in data collection.

There are several techniques for anomaly detection, broadly classified into three categories: supervised, unsupervised, and semi-supervised methods.

1. Supervised anomaly detection: this approach requires labeled data, where both normal and anomalous instances are identified. The method involves training a classifier to distinguish between normal and anomalous classes. Techniques include decision trees, support vector machines, and neural networks.

2. Unsupervised anomaly detection: in the absence of labeled data, unsupervised methods are employed. These techniques assume that normal instances are far more frequent than anomalies and use statistical measures to identify outliers. Common methods include clustering algorithms (e.g., k-means, DBSCAN), principal component analysis (PCA), and isolation forests.

3. Semi-supervised anomaly detection: these methods use a large amount of unlabeled data and a small amount of labeled data to build models. This is particularly useful when obtaining labeled data is expensive or time-consuming. Blázquez-García et al. (2021) [17] highlight several semi-supervised techniques that leverage autoencoders and generative adversarial networks (GANs) for anomaly detection in complex datasets.

2.6 False positives and false negatives

In the realm of deep learning, the concepts of false negatives and false positives are critical to understanding and evaluating model performance, especially in classification tasks. These terms

originate from the confusion matrix, which is a fundamental tool for assessing the performance of a classifier.
A false negative (FN) occurs when a model incorrectly predicts the negative class for an instance that actually belongs to the positive class. Conversely, a false positive (FP) arises when the model incorrectly predicts the positive class for an instance that actually belongs to the negative class. These errors have distinct implications depending on the application:

- False negatives (FN): a false negative occurs when the model fails to detect an object that is present in the image. This type of error is particularly problematic in critical applications such as disaster management and agricultural monitoring, where missing an object can lead to severe consequences. For instance, in precision agriculture, failing to detect a pest infestation can result in substantial crop damage.

- False positives (FP): a false positive occurs when the model incorrectly identifies an object that is not present in the image. This type of error can lead to unnecessary actions, such as wrong reports and warnings.

To evaluate and compare the performance of object detection models, metrics such as precision, recall, and the F1 score are commonly used:

Precision: measures the proportion of true positive detections among all positive detections. A higher precision indicates fewer false positives.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.2}$$

Recall: the ratio of true positives to the sum of true positives and false negatives. It measures the ability of the model to identify all positive instances.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.3}$$

F1-Score: the harmonic mean of precision and recall, providing a balanced measure of a model's performance.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Confusion matrix. A confusion matrix is a tool used in machine learning and statistics to evaluate the performance of a classification algorithm. It is a specific table layout that allows visualization of the performance of an algorithm, particularly in terms of its accuracy and error rates.
The confusion matrix summarizes the results of a classification by comparing the actual labels (or classes) with those predicted by the model. The matrix itself is typically a square matrix with dimensions equal to the number of classes being considered. Table 2.1 shows the typical structure of a confusion matrix for a binary classification problem.
The confusion matrix is not limited to binary classification and can be extended to multi-class classification problems, where each class will have its own row and column, leading to an $n \times n$ matrix for n classes. Each element (i, j) of the matrix indicates the number of instances where the actual class is i and the predicted class is j.

	Predicted positive	**Predicted negative**
Actual positive	True positive (TP)	False negative (FN)
Actual negative	False positive (FP)	True negative (TN)

Table 2.1: Confusion matrix.

Precision-recall curve The precision-recall curve is a plot that illustrates the trade-off between precision and recall for different threshold settings of a binary classifier.
As we vary the threshold that determines whether a given observation is classified as positive or negative, both precision and recall will change. The curve is generated by plotting recall on the x-axis and precision on the y-axis for these different threshold values.
When the threshold is very low, the model classifies many observations as positive, including most of the actual positives but also many negatives, leading to high recall but low precision. As the threshold increases, fewer observations are classified as positive, which reduces the number of false positives and increases precision, but also increases the number of false negatives, reducing recall [18].

Conclusion Using these metrics, researchers can better understand the trade-offs between false negatives and false positives and make informed decisions to optimize model performance. These errors are not just statistical inconveniences; they have real-world impacts that must be carefully managed. The choice to reduce false negatives over false positive or vice-versa depends on the goals of the users based on their own needs and the type of project [19].

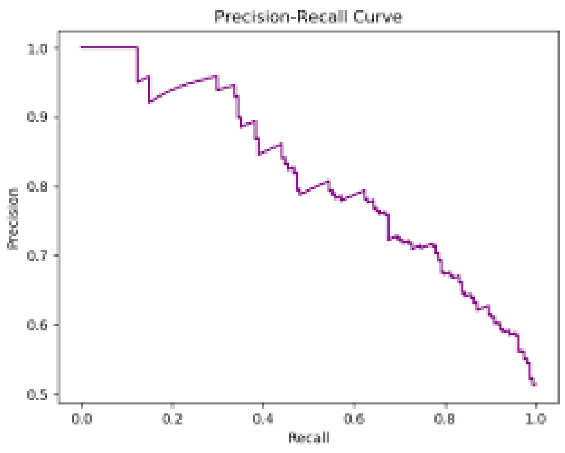


Figure 2.6: Example of PR curve.

3 Task and methodologies

3.1 Description of the project

The goal of this project is to recognize infesting plants in a monoculture field from aerial photographs taken from a drone.
We can consider the described problem as an anomaly detection problem because in this case we are not interested in recognizing the specific infesting plant, but we are only interested in detecting the presence or not of them in the field. Also, we are not concerned about extremely accurate localization of the infesting plants. Moreover, from what can be inferred from the image 3.1, the infesting plants can be very similar to monoculture plants, making the detection more difficult. Finally, there wasn't an already made dataset to work with at the beginning of the project, so it had to be created from zero.
For these reasons, in this thesis it was decided to treat this problem as a classification problem, more precisely a multiclass classification problem, where we are going to detect and classify 3 main classes: infesting plants, non-infesting plants and terrain (this will be further explained in section 3.3). The task does not involve detection or segmentation due to the fact that classification is easier to implement and, as shown in 3.3, the dataset for classification can be created using few images. Detection and segmentation tasks usually require careful labelling of images to create the desired dataset and the latter should have at least thousands of images in order to satisfying results.

Creating a dataset from scratch is not an easy task and therefore methods to reduce the workload should be considered for these kind of projects. Finally, fine-tuning was used on pre-trained models for classification, also in this case to reduce the number of data for training.
This thesis will also study the impact of false positives and false negatives during classification. By assessing the cost of each misclassification we can adapt the model to minimize the overall impact of errors on agriculture. This research will be conducted on finding the best range of parameters to tune the elasticity of the model on false positive/negative detection.

Assumptions and limitations. In this thesis, we assume that when conducting crop analysis on two consecutive days, the probability that the model fails to detect a pest on the first day and subsequently detects it on the second day or vice versa are independent events. Specifically, the probability of detection on the second day is not conditioned by the outcome of the first day's analysis. This assumption simplifies our model by treating each day's detection probability as an isolated event, thereby ignoring potential temporal dependencies between successive days. While this assumption facilitates the development and application of our model, it represents a limitation of this thesis, as it may not accurately reflect the dynamics of pest emergence and detection in real-world scenarios.

Introduction to the state of the art. Before delving into the methodology employed in this thesis, this section will present a selective review of recent studies on machine learning and deep learning algorithms for detecting plant infestations.

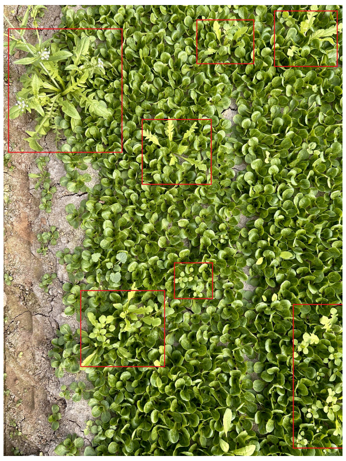


Figure 3.1: Example of an image with highlighted infesting plants.

3.2 State of the art

Traditional methods of plant disease detection rely heavily on visual inspections by experts, which are often time-consuming, subjective, and limited by the availability of trained professionals. With the advent of advanced machine learning (ML) and deep learning (DL) techniques, the detection of plant diseases has seen significant improvements in accuracy, efficiency, and scalability. This chapter explores the state of the art in plant disease detection, focusing on the latest advancements in ML and DL methodologies.

3.2.1 Examples of studies

Leon et al. (2021) The study of Leon et al. (2021) [20] addresses the critical issue of identifying diseases and physiological disorders in potato crops, specifically focusing on late blight (caused by *Phytophthora infestans*) and vascular wilt (caused by *Verticillium spp.*) using multispectral imagery from drones.
Potato crops were monitored in designated plots, with diseases and disorders visually characterized and multispectral images were captured using a MicaSense RedEdge camera attached to a hexacopter drone. Vegetation indices (Normalized difference vegetation index (NDVI) and soil-adjusted vegetation index (SAVI) to cite some) were calculated from the spectral data and used to discriminate between healthy and diseased plants. Machine learning models, particularly a generalized linear model (GLM) and a supervised random forest classifier, were employed to evaluate the discrimination capacity of the indices.
The random forest classifier showed accuracy rates between 73.5% and 82.5%, with kappa values ranging from 0.56 to 0.71, and ROC-AUC values from 0.88 to 0.98. Vegetation indices proved effective in distinguishing between healthy plants and those affected by diseases.
Even though the random forest model reaches a good level of accuracy, the results aren't high enough compared to the results of CNNs shown in the next papers.

Pujari et al. (2016) There has also been research on SVM. For example Pujari et al. (2016) [21] used Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs) for detecting and classifying plant diseases. The approach involves using color and texture features from images of plant diseases, which are then processed with the described models. The study focuses on six classes of plant diseases and uses a dataset of 900 sample images.
The paper concludes that the SVM classifier, with its reduced feature set, provides a reliable

and efficient method for the classification of plant diseases (the F1-score reached for SVM is 94.50%).
SVMs are particularly known for their high accuracy and precision in binary and multi-class classification problems. They are effective in high-dimensional spaces and are particularly useful in cases where the number of dimensions exceeds the number of samples. However, the use of SVM is training large image dataset presents some problems:

- SVMs can struggle with scalability, particularly when dealing with large datasets typical in image classification tasks. Training time increases significantly with the size of the dataset, making SVMs impractical for very large-scale image datasets.
- Images often contain highly complex and non-linear patterns. While SVMs can use kernel functions to handle non-linear data, they may not capture the intricacies as effectively as deep learning models, which are specifically designed to learn hierarchical representations.
- SVMs typically require manual feature extraction and selection (done by Pujari in this paper), which can be labor-intensive and may not always result in the most relevant features. Deep learning models, on the other hand, can automatically learn features from raw image data, leading to potentially better performance.

Mohanty et al. (2016) CNNs have become the cornerstone of deep learning-based plant disease detection due to their superior performance in image recognition tasks. Research by Mohanty et al. (2016) [22] used CNNs to identify 14 crop species and 26 diseases using a public dataset of 54306 images. The models used during training are AlexNet [23] and GoogLeNet [24].
During preprocessing every image in the dataset is resized to 256 x 256 pixels. Then, the images are used in 3 different configurations: color, original color images, grey-scale and segmented, where the background is removed. To evaluate model performance and mitigate overfitting, various train-test splits are used: 80-20, 60-40, 50-50, 40-60, and 20-80. The dataset includes multiple images of the same leaf from different orientations, and these are handled to ensure that all images of a particular leaf are either in the training set or the testing set, not both.
The highest overall accuracy was 99.34%, achieved by GoogLeNet using transfer learning on color images with an 80-20 train-test split. GoogLeNet consistently outperformed AlexNet across all configurations and models trained on color images performed the best. The best

results were observed with an 80-20 train-test split but even with more challenging splits like 20-80, the models performed well, demonstrating robustness against overfitting. For instance, GoogLeNet with transfer learning on color images achieved an overall accuracy of 98.21% with a 20-80 split.

An important thing to note is that transfer learning (see 2.4) also consistently outperformed training from scratch for both models, as we can see in 3.2

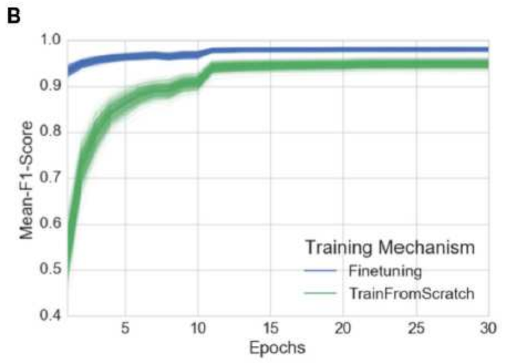


Figure 3.2: Transfer learning compared to learning from scratch. Transfer is a more efficient method for training a model.

Ferentinos (2018) In the study conducted by Ferentinos (2018) [25], transfer learning techniques were employed to leverage pre-trained deep learning models, such as VGG16, ResNet, and Inception, for the purpose of plant disease detection and diagnosis. The primary focus was on utilizing convolutional neural networks (CNNs) to accurately identify plant diseases from images of healthy and diseased leaves.

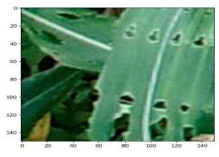
Ferentinos utilized an extensive open database comprising 87,848 images that covered 25 different plant species and 58 distinct classes, which included various plant diseases as well as healthy plants. The images in the dataset were captured in both controlled laboratory conditions and real cultivation environments. The database's diversity ensured that the trained models could generalize well to real-world scenarios.

To train the CNN models, the database was divided into training and testing sets, with an 80/20 split. Pre-processing of the images involved resizing and cropping them to 256x256 pixels. Unlike some prior approaches, grayscale conversion and leaf segmentation were not performed, as deep learning models have the capability to effectively learn relevant features directly from raw color images.

The best model was VGG16, which achieved a remarkable accuracy of 99.53% in identifying

the correct plant disease from leaf images. This high success rate underscores the potential of transfer learning and CNNs in developing effective automated plant disease detection systems. The inclusion of images captured under real cultivation conditions in the dataset contributed to the robustness and practical applicability of the trained models.
There are some minor problems found in the study, for example out of the 82 misclassified images, some were identified as "faulty" because they did not contain any plant leaves. For instance, certain images were incorrectly registered in the class for *tomato with early blight* but were classified by the model as *healthy corn* due to similar soil textures and the slim appearance of the corn leaves.
In conclusion, Ferentinos (2018) demonstrates the efficacy of transfer learning and deep learning methodologies in overcoming the challenges of plant disease detection. The study sets a foundation for future research and development in integrating such systems into practical agricultural workflows.

Ishengoma et al. (2022) Ishengoma et al. (2022) [26] proposed a novel hybrid CNN model aimed at enhancing the speed and accuracy of detecting maize plants infested by fall armyworms (FAWs) using UAV-based imagery.
The proposed hybrid model combines VGG16 and InceptionV3 in a parallel setup: the input image is simultaneously applied to both models and the outputs of both models are passed to the full-connected layers for classification. This design leverages the lower-layer feature extraction capabilities of both models while only retraining the top layers to minimize training time and maximize performance.
The study was conducted on maize farms in Morogoro, Tanzania, where UAVs (Unmanned Aerial Vehicles) were used to capture high-resolution images of maize leaves. The dataset, which contains a total of 500 images, included both original and augmented images to balance the number of healthy and infected samples. After training, we notice that the hybrid approach reduces the training time by 16% to 44% compared to individual models and achieves an accuracy of 96.98%.
This paper shows the possibility of achieving high accuracy in this context with a medium-sized dataset. The hybrid model approach can be leveraged for improving performances.
Even if only the images of the infected maize are shown, we can infer that the different classes are different from each other, due to the fact that we can see noticeable holes in the leaves of the infested plants; this makes the classification easier and allows the researchers to reach high performances.


(a)

(b)

Figure 3.3: Infested maize leaves shown in Ishengoma paper.

YADAV ET AL. (2023) There are also studies that developed detection algorithms. For example Yadav et al. (2023) [27] presented an approach for detecting volunteer cotton plants in maize fields using UAV imagery and the YOLOv3 deep learning algorithm. The method aims to improve the identification and management of volunteer cotton (VC), which poses significant challenges in cotton-producing areas due to its potential to harbor pests and diseases.
The YOLOv3 algorithm is employed due to its superior performance in real-time object detection. The architecture is built upon Darknet-53, a backbone network with 106 layers in total, including residual blocks to handle vanishing gradients effectively. The model divides the input images into grids and predicts bounding boxes for objects in these grids. It uses predefined anchor boxes and logistic regression for classifying the detected objects.
In this paper mean average precision (mAP) is used as the performance metric. mAP is a commonly used metric in object detection tasks to measure the accuracy of a model. It represents the mean of the average precision (AP) scores for each class in the dataset. The AP for a single class is calculated as the area under the precision-recall curve, which plots precision against recall at various threshold levels. mAP provides a single value that summarizes the performance of the model across all classes, making it easier to compare and understand the model's effectiveness. For YOLOv3, which is a multi-class object detector, mAP gives a comprehensive measure of accuracy by combining precision and recall.
The use of mAP30 in this study, where the Intersection over Union (IoU) threshold is set to 30%, indicates a lower strictness in matching predicted bounding boxes with ground truth. This is practical for the application because volunteer cotton plants often grow in groups, and a lower IoU threshold reduces the risk of missing these plants while ensuring that nearby plants (even if partially overlapping) are detected. By employing these metrics, the researchers ensured that the YOLOv3 model could effectively detect VC plants in aerial images, achieving an

overall accuracy (mAP) higher than 80%. The YOLOv3 model demonstrated high accuracy in detecting volunteer cotton plants in various conditions. The use of residual blocks and logistic regression significantly improved the model's performance, especially in differentiating between maize and VC.

AP calculation

$$AP = \sum_{k=0}^{n-1} [R(k) - R(k+1)] \cdot P(k)$$

where R is recall and P is precision at different thresholds.

mAP calculation

$$mAP = \frac{1}{n'} \sum_{k=1}^{n'} AP_k$$

where n' is the number of classes.
Even though YOLO seems to be a promising architecture, the problem resides in the creation of the dataset. In this paper, the dataset required for a detection model needs to have hundreds of data points. Manual labelling for creating a dataset from scratch can be considered only if one has the available resources and most importantly, if the goal of one project is to reach high level of detection, which is not the objective of this thesis.

3.2.2 Challenges

Despite the significant advancements, several challenges remain in the field of plant disease detection using ML and DL methods:

1. Data quality and quantity: high-quality labeled datasets are essential for training robust models. For training a model we need large, well-labeled datasets; this task can be resource-intensive due to manual data labeling.
2. Generalization: models trained on specific datasets may not generalize well to different environments, crop varieties, or disease manifestations. Addressing this requires the development of more generalized models.

3.3 Dataset

This segment discusses the dataset, explaining how it was created and the pre-processing steps used to prepare the dataset for training. This section provides an overview of the data, including how it was generated and its general characteristics. It details the data structure requirements of our models and discusses the data processing steps, taking into account the limitations of the available training resources. This explanation provides a thorough understanding of the data's transformation from collection to its final form.

3.3.1 Dataset description

The dataset for this thesis was gathered in a greenhouse from a monoculture field of valerianella (*valerianella locusta*). The images were taken during the morning in a single day. They were captured using two methods:

1. An iPhone 14 camera aimed at the ground from a height of 0.5 meters. In total 60 images were collected with this method.
2. A drone-mounted camera directed at the ground from a height of 2 meters. To take these photos, the drone flew over two crop strips at a time and it required two passes to cover the entire field (figure 3.4 shows an example of an image taken with this method). The drone model used is a Mavic3M from DJI. With this method, instead, we collected 30 images.

The models will be tested on images captured by the drone. Images taken from a low height serve two purposes: they increase the number of data points and provide higher resolution images, as these images were taken with an iPhone. This approach is beneficial for testing the model under easier conditions and to check during the first phases of the project its potential success in the task. Additionally, having numerous data points can enhance the model's generalization capability during training.

Each image is divided in a grid of patches, each one of 112x112 pixels. During the initial phase of the project the size of the patches was chosen to be of 224x224 pixels, which is the default input size for many CNNs, including ResNet 3.4.1 and DenseNet 3.4.5. Adopting this patch dimension led to some problems. The main issue was that the patch often captured more than one class per image; for example there could have been a patch where there were an infesting plant and crops in the same image. Also, the final segmentation resulted inaccurate due to the patch being too large and finally, having bigger patches reduced the number of data

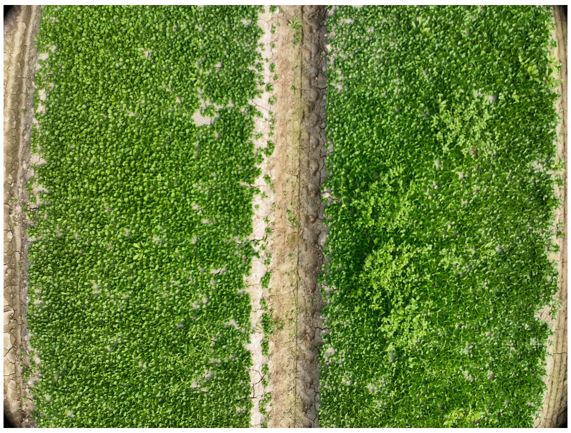


Figure 3.4: Example of image taken from the drone.

points.
Conversely, using a smaller patch dimension, such as 56x56, presented the opposite problem: an inability to capture meaningful classes, although it provided a greater number of data points. Ultimately, a 112x112 patch dimension was chosen as a middle ground between these two options.
During preprocessing, a given image is divided into smaller, uniformly-sized sections. After being read, the image is then systematically cropped into smaller segments based on a predefined dimension. These cropped segments are subsequently collected and stored. The patches are not overlapping. Each patch is cropped from the image in a grid-like manner, with each patch being directly adjacent to the next. During the process the positions are calculated for each patch based on fixed dimensions, ensuring that each segment is distinct and non-overlapping. In case there is the need to increase the number of *infesting* labelled images in the dataset it is possible to manually crop the original image only on the region of interest (in this case, where the infesting plants are present) and then create the patches from the cropped images. Subsequently, a label is assigned to each patch:

1. *Infesting*: there is the presence of an infesting plant.
2. *Valeriana*: there is not the presence of an infesting plant.
3. *Terrain*: there is mainly terrain and not enough culture.

For the iPhone images we can extract about 1000 patches per image, instead for the drone images we are able to extract approximately 1500 patches. The presence of the label *terrain* is

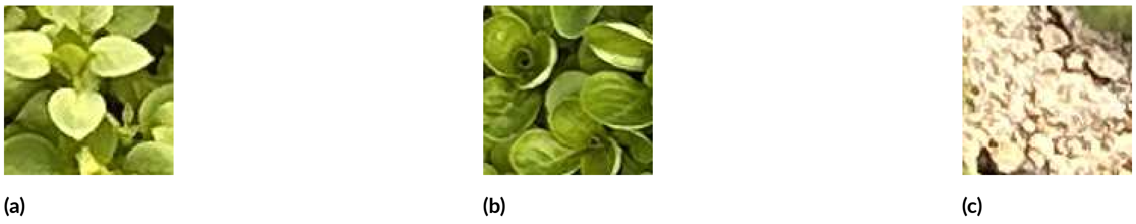


Figure 3.5: Example of patches from the first dataset. a) Label *infesting*. b) Label *valeriana*. c) Label *terrain*.

helpful during the classification because it was evident during testing that having only 2 classes increased the percentage of misclassification; for example the patches where there was mainly terrain were usually classified as *infesting*.
After this process 2 dataset were created based on the images present, which are going to be called *dataset-v1*, that contains only patches from iPhone images and *dataset-v2*, that contains both the images from the iPhone and the drone camera.
The project started by evaluating only the iPhone images mainly because the classes were easier to distinguish from the resulting patches. After testing the model trained on the first dataset, it was evident that the model did not classify well enough the images from the drone so it was necessary to expand the dataset further to incorporate also the images captured from the drone. So in this thesis, the first dataset is used to test the models under simplified conditions and to check whether of not the models were able to distinguish the different classes. The second dataset is the main dataset and the model trained on that will be used to evaluate the entire image.
The first dataset contains 6804 entries: 1364 *infesting*, 916 *terrain*, 4524 *valeriana*. The second dataset contains 10708 entries: 2355 *infesting*, 1557 *terrain*, 6796 *valeriana*.
Not all the images collected in the greenhouse were used for the creation of the datasets. For the first one, we used 10 images for the creation of the patches and for the second dataset 10 more images from the drone were included. It is important to note that after the creation and labelling of the patches, not all the *valeriana* patches were incorporated in the datasets; this choice was made to avoid unbalancing the datasets too much.
As we can see in 3.6 the patches are more blurred and have less quality compared to the patches in 3.5; this comes natural due to the height from which the images were taken and therefore makes the classification harder.

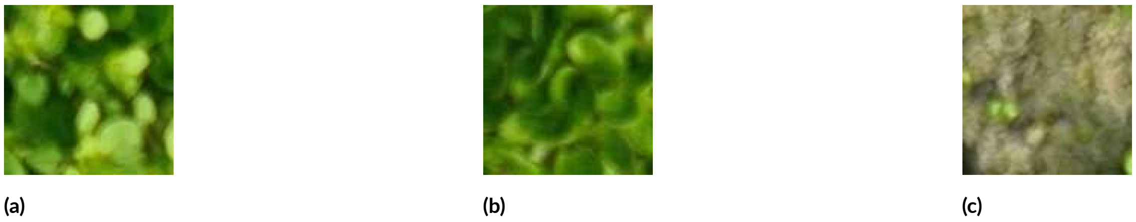

(a) (b) (c)

Figure 3.6: Example of patches from the second dataset. a) Label *infesting*. b) Label *valeriana*. c) Label *terrain*.

3.3.2 Dataset labelling

At the beginning of the project every patch was manually labelled. The criterion for assigning the patches was the following: a patch was labelled *terrain* if the terrain covered almost all of the image, *infesting* if there was a noticeable presence of infesting plants and all the rest of patches were assigned to *valeriana*. Sometimes it was hard to decide whether an image should be labelled *infesting* or *valeriana*: in this case if there was even a small presence of infesting plant (e.g., the infesting plant was on the upper corner of the patch), the image was labelled *infesting*. This was an arbitrary choice based on the preferred output classification of the model; an opposite choice could also be made.

This method was very time-consuming and later during the project development another labelling method was developed to solve this problem. The proposed method involves manual segmentation and automatic creation and labelling of the patches. The process involves segmenting the image by tracing the infesting plants. Then, the segmented image and the original image are subsequently divided into patches following the established method and an algorithm determines if a patch contains part of the infesting plant. A patch is labeled as *infesting* if the equivalent segmented patch contains at least 20% of infesting plants in the overall image. To label the *valeriana* and *terrain* classes we start from the original image which is divided in patches. From each patch we extract the green pixels and if the if the percent of the green pixels on the overall image is less than 5% the patch is classified as *terrain*, otherwise it is classified as *valeriana*.

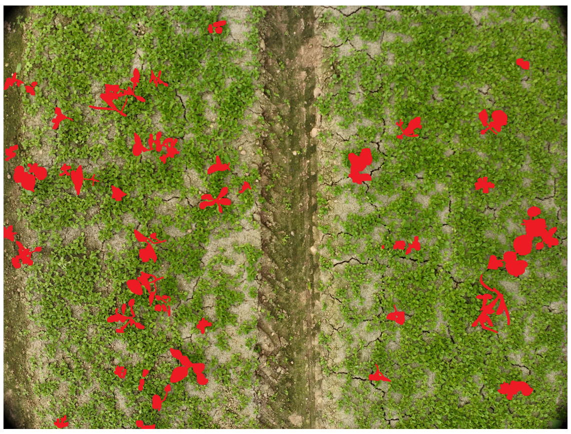


Figure 3.7: Example of segmented image.

3.3.3 Dataset pre-processing

Due to the nature of the dataset the class *infestanti* is less present compared to the class *valeriana*; this makes the dataset unbalanced.
When we have an unbalanced dataset, it means that some classes have significantly more samples than others. For example, in the first dataset we have 70% of the images labeled as *valeriana* and only 20% labeled as *infesting*. Dataset imbalance can be problematic for many reasons:

- The model will naturally try to minimize the overall error during training. if most of the images are of *valeriana*, the easiest way for the model to achieve high accuracy is to predict *valeriana* for every image. This is because it will be correct 70% of the time, thus achieving high accuracy. Accuracy, in this case, becomes a misleading metric. A high accuracy might indicate that the model is performing well, but in reality, it's simply ignoring the minority class.

- By focusing on the majority class (*valeriana*), the model doesn't learn the distinguishing features of the minority class (*infesting*). As a result, when it encounters a new image of an infesting plant, it's likely to missclassify it as a crop, because it hasn't learned enough about what makes infesting plants different from crops.

To solve this issue we can drop randomly a percentage of images with class *valeriana* in order to create a more balanced dataset. The drop percentage is set to 70% for the first dataset. The second dataset is not reduced and therefore used in its entirety in order to use all the available images. This was done in order to visualize the problems listed before. Also empirical analysis as shown in Chapter 4 proved that even though some models classified all patches as *valeriana*, for deeper models this issue did not occur.
Finally, some the metrics explained in 2.6 (precision and recall) will be used alongside accuracy to get a better evaluation.

Given that the dataset was collected on the same moment of the day, different kind of transformations to the dataset were applied in order to increase the number of data points and to enhance the performances of the model under different conditions (e.g., picture taken at different hours, diverse resolutions of the camera) without collecting other data. The data augmentation is applied using ImageDataGenerator from keras on the train dataset. The transformations applied are:

- Zoom: involves either zooming in or out on an image. This helps the model to be invariant to different sizes of objects within the image. The zoom range is set to 0.2; this means zooming can be between 80% and 120%.

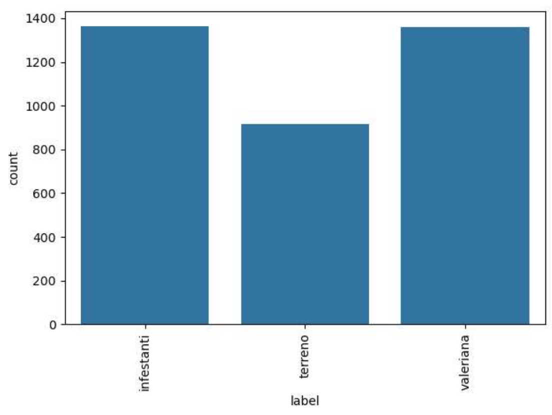


Figure 3.8: First dataset balanced.

- Flip: flipping an image can be done horizontally or vertically. This is particularly useful in cases where the orientation of the object in the image does not affect its classification. In this case the flip is applied both horizontally and vertically.
- Brightness intensity: brightness change involves adjusting the brightness levels of an image, making it either lighter or darker. This can help the model be more robust to different lighting conditions.

Finally, every patch is rescaled by a factor of 1/255. Images typically have pixel values in the range of 0 to 255 for each channel. By rescaling by 1/255, these values are normalized to the range of 0 to 1. This is done to ensure to have parameters that are in a smaller and consistent range, which is helpful for stability and faster convergence during training.

3.3.4 Dataset-v3

During the last phases of the project, we collected another sample of photos during a field mission in the greenhouses. The photos were gathered during the day 8/7/2024 from 11:30 to 16:30 (from late morning to early afternoon). In total, for the valerianella field, 30 new images were collected with a drone using the same method described at the beginning of the section 3.3 (model DJI Mavic3M, 2 meters height). Finally, the photos were taken in another greenhouse, therefore we don't have an overlapping dataset.
All the new images were manually segmented and processed using the second method explained in 3.3.2 to create labelled patches. These images were added to the *dataset-v2* and the resulting dataset is going to be called *dataset-v3* which is the result of 50 segmented images. This dataset

contains a total of 62808 entries: 8101 *infesting*, 4518 *terrain*, 50189 *valeriana*. During preprocessing the drop percentage for the valeriana class was set to 50% in order to preserve the 1:3 ratio between *infesting* and *valeriana* classes. The resulting dataset, which is going to be called *dataset-v3-reduced-50*, contains a total of 37713 entries: 8101 *infesting*, 4518 *terrain*, 25095 *valeriana*. Finally, the same augmentations were applied.

3.4 Models

This section outlines the technical implementation of neural network models developed for analyzing the infesting plants dataset. Each model is trained on the dataset and evaluated based on different kind of parameters. The models used in this project are ResNet50, VGG16, InceptionV3, InceptionResNet and DenseNet201.

3.4.1 ResNet

Residual Networks (ResNet) stand out as one of the most influential and widely adopted models. ResNet, introduced by He (2016) [15], addresses the problem of training very deep neural networks by introducing a novel residual learning framework.
The fundamental breakthrough of ResNet lies in its ability to mitigate the vanishing gradient problem, which typically hampers the training of deep networks. Traditional CNNs often struggle to propagate gradients effectively when the network depth increases. This leads to degraded performance and makes it challenging to optimize the network.
ResNet overcomes this issue by introducing residual blocks. Each residual block consists of a series of convolutional layers, batch normalization, and ReLU activation functions. What sets these blocks apart is the identity shortcut connection, which bypasses one or more layers by performing identity mapping. The output of the residual block is the sum of the input and the output of the convolutional layers within the block.

$$y = \mathcal{F}(x, \{W_i\}) + x \tag{3.1}$$

In the function 3.1, x represents the input to the residual block, $\mathcal{F}(x, \{W_i\})$ denotes the residual function, and W_i are the weights of the convolutional layers.
When the input data x is passed into the residual block, it is initially processed by one or more convolutional layers, batch normalization, and ReLU activation functions. These operations transform the input in some way, which is described by the function $\mathcal{F}(x, \{W_i\})$. This resid-

ual function represents the output of the convolutional layers in the residual block. It's called the "residual" function because it tries to learn the residual mapping. The idea is that it is easier for the layers to learn the residual (the difference) between the input and the desired output rather than the entire transformation directly.
The identity shortcut connection is a direct connection that skips the convolutional layers and directly connects the input to the output. The final output y of the residual block is obtained by adding the input x to the output of the residual function $\mathcal{F}(x, \{W_i\})$. This addition helps to propagate the input directly to the output, which can alleviate the vanishing gradient problem and make it easier to train deep networks.
In their paper, He et al. demonstrated that ResNet with 152 layers significantly outperformed

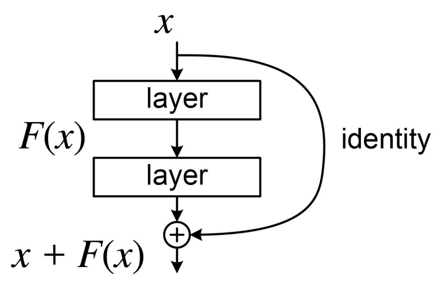


Figure 3.9: ResNet block.

state-of-the-art models on the ImageNet dataset, while still being easier to optimize. This was a remarkable achievement, as previous architectures struggled to go beyond 20-30 layers without encountering performance degradation.
Since its inception, several variants of ResNet have been proposed to further enhance its performance and applicability. Some notable variants include ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152. These versions differ in the number of layers, providing a trade-off between model complexity and computational efficiency. ResNet-50 and ResNet-101 are particularly popular in practice due to their balance of accuracy and computational cost [He2016] [15].
ResNet has been successfully applied to a wide range of computer vision tasks, including image classification; ResNet models consistently achieve top performance on benchmarks like ImageNet and CIFAR-10.

3.4.2 VGGNet

VGGNet, introduced by the Visual Geometry Group (VGG) at the University of Oxford, has significantly influenced the field of computer vision. Presented by Simonyan and Zisserman in 2014, VGGNet achieved remarkable performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014 [28].
VGGNet's architecture is notable for its simplicity and uniformity. Unlike its predecessors, which often employed complex topologies, VGGNet relies on very small (3x3) convolution filters throughout the network, a design choice that contributes to its clarity and effectiveness.
The key idea behind VGGNet is the use of multiple stacked convolutional layers with small receptive fields. The depth of the network is increased by adding more convolutional layers, with the number of filters doubling after every few layers. For instance, VGG-16 and VGG-19, two popular configurations, have 16 and 19 weight layers, respectively.
VGGNet offerts several advantages, primarily its uniform architecture where the use of small, uniform convolutional filters (3x3) simplifies the network design and makes it easier to understand and implement and its increased depth which allows it to capture more complex and hierarchical features, which contributes to its high performance on image recognition tasks. VGGNet has also become a popular choice for transfer learning.
VGGNet comes in several configurations, primarily differing in the depth of the network; the deeper variants, such as VGG-16 and VGG-19, generally achieve better performance at the cost of increased computational resources.
VGGNet has been successfully applied in numerous computer vision applications, including:

- Image classification: VGGNet's pre-trained models are extensively used for image classification tasks, achieving high accuracy on benchmark datasets such as ImageNet.

- Object detection: VGGNet serves as a backbone for object detection frameworks like Fast R-CNN and Faster R-CNN, providing strong feature extraction capabilities [Girshick2015] [29], [Ren2015] [30].

- Semantic segmentation: VGGNet-based models, such as Fully Convolutional Networks, have been employed for pixel-wise segmentation tasks, significantly improving the accuracy of semantic segmentation [Long2015] [31].

3.4.3 InceptionNet

InceptionNet, also known as GoogLeNet, introduced a novel architecture that significantly advanced the field of deep learning for computer vision. It was developed by Szegedy et al. (2015) [24] at Google.

The key innovation of InceptionNet is the inception module, which allows the network to capture multi-scale information more effectively. Traditional convolutional networks typically use layers with fixed filter sizes, which may not be optimal for capturing features at different scales. The inception module, on the other hand, applies multiple filters of different sizes to the same input and concatenates their outputs. An inception module typically includes: 1x1 convolutions, used to reduce dimensionality and computational cost, 3x3 convolutions and 5x5 convolutions, respectively used to capture medium-sized and large features and 3x3 max Pooling which helps to capture spatial hierarchies and provides additional robustness.

Each of these convolutional and pooling operations is applied in parallel, and their outputs are concatenated to form the input for the next layer. The architecture of InceptionNet is deep, consisting of 22 layers, but thanks to the inception modules, it remains computationally efficient.

The inception module is designed to approximate a sparse network with a series of dense layers. This approach helps to mitigate the issue of computational expense that typically accompanies deep networks. Szegedy et al. (2015) introduced the initial InceptionNet architecture, often referred to as GoogLeNet (Inception v1) [Szegedy2015]. An example of an inception module looks as follows:

$$\text{Concatenate}\left(\text{Conv1x1}(x), \text{Conv3x3}(x), \text{Conv5x5}(x), \text{MaxPool3x3}(x)\right)$$

Where ConvNxN denotes a convolution with an $N \times N$ filter.

The advantages of InceptionNet are its ability to perform multi-scale feature extraction, thanks

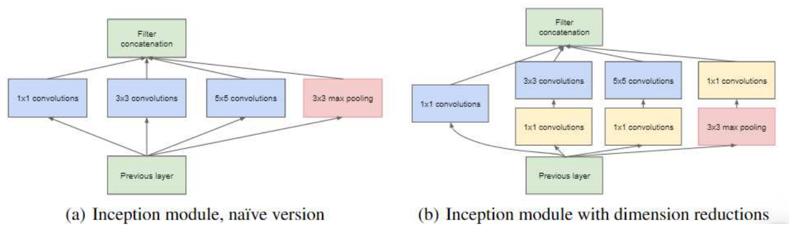


Figure 3.10: Inception modules.

to the inception module which allows the network to capture features at multiple scales, improving its ability to recognize objects of varying sizes and dimensionality reduction, by using 1x1 convolutions, InceptionNet effectively reduces the number of parameters and computational complexity without sacrificing performance.
The variants of InceptionNet are: Inception v2 and v3 which introduced factorized convolutions and improved training methods [Szegedy2016] [32] and Inception v4 and Inception-ResNet which combined inception modules with residual connections for better performance [Szegedy2017] [33].

3.4.4 Inception-ResNet

Inception-ResNet is a hybrid neural network architecture that combines the strengths of Inception modules and Residual connections, aiming to improve both the depth and efficiency of deep learning models. This architecture was introduced by Szegedy et al. (2017) [33] and has proven to enhance the performance and training speed of deep networks.
Inception-ResNet integrates the multi-scale feature extraction capability of Inception modules with the residual learning framework popularized by ResNet. The key idea is to combine the inception modules, which capture features at multiple scales, with residual connections, which help in training deeper networks by mitigating the vanishing gradient problem.
By combining inception modules and residual connections, Inception-ResNet can effectively capture multi-scale features while maintaining the benefits of deep networks. Also residual connections facilitate the training of very deep networks by allowing gradients to propagate more easily through the network.
There are versions of Inception-ResNet are: Inception-ResNet v1, the initial version and Inception-ResNet v2, an improved version with deeper architecture and modified inception modules.

3.4.5 DenseNet

DenseNet, short for Densely Connected Convolutional Networks, is a neural network architecture introduced by Huang et al. (2017) [34]. DenseNet's design addresses some of the limitations of traditional deep networks by ensuring maximum information flow between layers.
DenseNet's architecture is characterized by dense connectivity between layers. In contrast to traditional convolutional networks, where each layer has its own set of weights and is only connected to the previous layer, DenseNet connects each layer to every other layer in a feed-forward fashion. Specifically, each layer receives the feature maps of all preceding layers as input, lead-

ing to improved feature reuse and learning efficiency.
Formally, the output of the l^{th} layer is defined as:

$$\mathbf{x}_l = H_l([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{l-1}])$$

where $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{l-1}]$ refers to the concatenation of the feature maps from layers 0 to $l-1$, and H_l denotes the transformation (composed of Batch Normalization, ReLU, and Convolution) applied at the l^{th} layer.
The DenseNet architecture offers many advantages; dense connectivity ensures that gradients flow more easily through the network, facilitating training of very deep networks and DenseNet is parameter-efficient because it does not require learning redundant feature maps, as features are reused throughout the network. Also the direct connections between layers help to alleviate the vanishing gradient problem, which is common in deep networks.
DenseNet comes in several variants, primarily differing in the number of layers and the connectivity pattern: DenseNet-121, DenseNet-169 and DenseNet-201.
DenseNet has been successfully applied in various computer vision tasks, demonstrating its versatility and robustness but it shines in medical image analysis; DenseNet's ability to capture fine-grained features makes it suitable for medical image analysis, such as tumor detection and segmentation [Shen2017] [35].

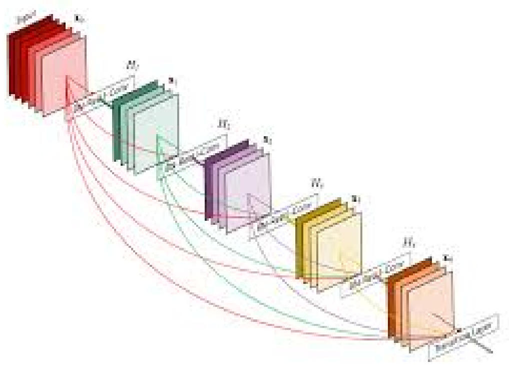


Figure 3.11: DenseNet architecture.

4 Experimental results

In this chapter we will provide the performance analysis of the models. The models evaluated are: ResNet50, VGG16, InceptionV3, InceptionResNetV2, DenseNet201. These models were fine-tuned using the ImageNet weights and trained on both versions of the dataset 3.3. In this case, all the layers of the pre-trained model were freezed and we only trained the last layers, which were added on top of the pre-trained models (see 2.4.2 for reference).
The datasets were augmented as previously described and were divided in training, validation and test sets. The datasets were initially divided between training and validation with a 80% and 20% split respectively. Then, the training set was further split to create the test set (80% and 20% respectively).
Training was conducted over 60 epochs with a learning rate of 1e-5 and a batch size of 32. Patience was set to 3 and started from epoch 20. The models were trained on a GEFORCE GTX 1050 GPU.
Model performance was evaluated on the test set of the dataset using accuracy, precision, and recall as metrics. For each model, training and validation loss, as well as training and validation accuracy, are reported (validation is referred to validation set).
The second part of this chapter revolves around the methodologies to tune the number of false positives and false negatives during and after classification. For this task, a different test was used, which did not require the previously mentioned *dataset-v1* and *dataset-v2*. The process and the different test will be explained in detail later in the section 4.5.

4.1 Model parameters

Learning rate The learning rate controls the size of the steps the optimization algorithm takes during training. It determines how quickly or slowly a model converges to a minimum of the loss function. The learning rate is a hyperparameter that scales the gradient updates during training. If the learning rate is too high, the model might converge too quickly to a sub-optimal solution or oscillate around a solution without ever settling down, leading to poor generalization. Conversely, if the learning rate is too low, the training process will be excessively slow, and the model might get stuck in local minima, requiring more time to converge to an optimal solution.
When the learning rate is too high the optimization algorithm makes large updates to the model parameters. This can result in the following issues:

- Overfitting: high learning rates can cause the model to memorize the training data rather than learning the underlying patterns. This is because large steps can skip over or miss the general trends in the data, leading to high variance in predictions.
- Instability: large updates can cause the loss function to fluctuate widely or even diverge, preventing the model from converging to a stable solution.
- Poor generalization: due to the rapid changes in parameter values, the model may not generalize well to unseen data, exhibiting poor performance on validation and test datasets.

When training the models with a high learning (e.g., 1e-4) all of the issues presented are evident. On the other hand, very low learning rates make the optimization process painstakingly slow. Training with a very low learning rate requires a significantly higher number of epochs to converge, increasing computational costs and time.
An intermediate learning rate, such as 1e-5 offers the best solution for convergence, stable training and generalization.

Batch size The batch size, which refers to the number of training examples used in one iteration to update the model parameters, can significantly influence the training dynamics and overall performance of the model. In the training process, the batch size of 32 is identified as optimal. This choice balances the trade-offs between computational efficiency, convergence stability, and model generalization.
Using a large batch size in training machine learning models can lead to several challenges,

including higher memory requirements that may exceed hardware capacity, causing out-of-memory errors. Additionally, large batch sizes result in more deterministic and smoother gradient updates, which might reduce the model's ability to generalize to unseen data due to the lack of gradient noise. While computation per iteration might be faster with large batches, the time per epoch could be longer since there are fewer iterations per epoch.
Conversely, small batch sizes introduce noisy updates with higher variance, making the training process potentially less stable and more prolonged. Smaller batches may lead to inefficient computation as they might not fully utilize the parallel processing capabilities of modern hardware, extending the overall training time and requiring more epochs for the model to converge.
The selection of a batch size of 32 represents a well-considered balance between computational efficiency, convergence stability, and model generalization. It avoids the pitfalls of both very large and very small batch sizes, ensuring that the training process is both efficient and effective.

Epochs An epoch represents a full cycle through the entire training dataset. During each epoch, the model processes all training samples, updating its parameters through backpropagation and gradient descent. The purpose of multiple epochs is to allow the model to refine its parameters progressively, improving its performance with each pass through the data.
If the number of epochs is too low, the model might not have sufficient opportunity to learn the underlying patterns in the data. This can result in underfitting, where the model fails to capture the complexities of the training data, leading to poor performance on both the training and validation sets. Underfitting occurs because the model has not had enough iterations to adjust its weights adequately, resulting in a model that lacks the necessary capacity to make accurate predictions.
On the other hand, training for too many epochs can lead to overfitting. Overfitting occurs when the model learns not only the underlying patterns but also the noise and specific details of the training data. While this might result in excellent performance on the training set, the model's ability to generalize to new, unseen data diminishes, leading to poor performance on validation and test sets. Additionally, training for too many epochs is computationally inefficient, wasting time and resources.
Usually, the value of epochs is usually set between 50 and 100. This is an indicative number because if we use *patience* as a hyperparameter we model will terminate training well before reaching the maximum number of epochs.

Early stopping Early stopping is a regularization technique used during training of machine learning models to prevent overfitting. It involves monitoring the model's performance on a validation set and stopping the training process when the performance ceases to improve. The parameter *patience* determines how many epochs to wait for an improvement before halting the training.
Early stopping is a safeguard against overfitting by terminating the training process once the model's performance on a validation set stops improving. The *patience* parameter specifies the number of epochs to wait after the last improvement before stopping. If the model does not show improvement in the validation performance within the specified patience period, training is halted. This approach helps in finding the sweet spot where the model has learned sufficiently but not excessively.
The patience parameter helps avoid premature termination of training. Without patience, training might stop as soon as the model experiences a minor fluctuation or temporary plateau in performance, which might not accurately represent its overall learning trend. By allowing a few additional epochs, patience provides the model with an opportunity to overcome short-term stagnations and potentially reach a better performance.
Choosing to start monitoring for early stopping from epoch 20 is a strategic decision. Early epochs are typically characterized by significant improvements as the model quickly learns basic patterns in the data. Monitoring for early stopping from the very beginning might result in premature termination. By waiting until epoch 20, we allow the model to settle and make substantial initial progress. This setup ensures that early stopping is applied only after the model has had enough time to stabilize its learning curve.

4.2 Dataset v1

Table 4.1: Model performance comparison on first dataset. The values are calculated on the test set.

Model	Accuracy	Precision	Recall	Time
ResNet50	67.3%	73.7%	51.2%	3min 42s
VGG16	85.3%	86.3%	84.8%	6min 7s
InceptionV3	86.4%	86.4%	85.4%	3min 52s
InceptionResNetV2	87.5%	88.3%	80.1%	4min 9s
DenseNet201	90.1%	90.9%	89.3%	4min 52s

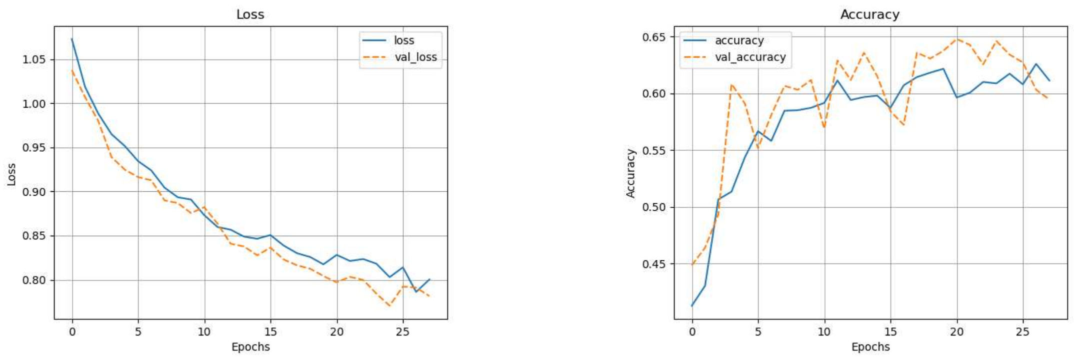


Figure 4.1: Loss and accuracy plots for ResNet50.

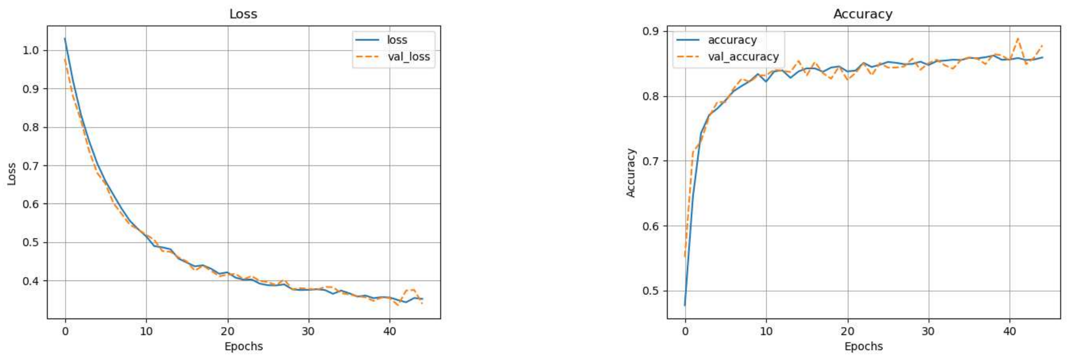


Figure 4.2: Loss and accuracy plots for VGG16.

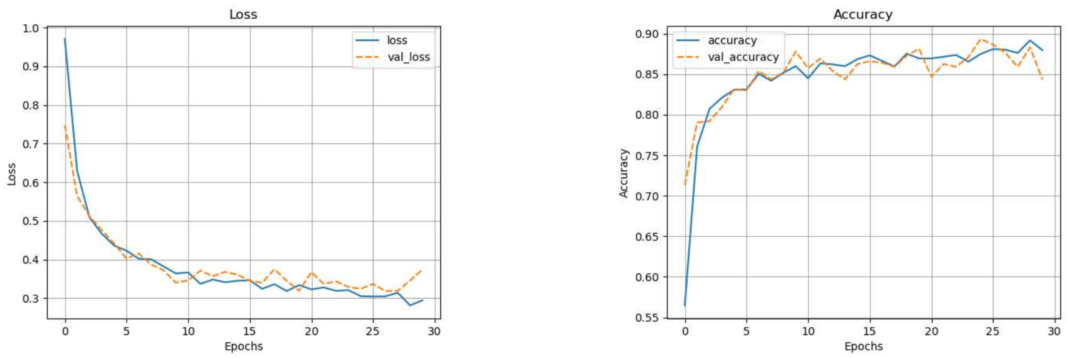


Figure 4.3: Loss and accuracy plots for InceptionV3.

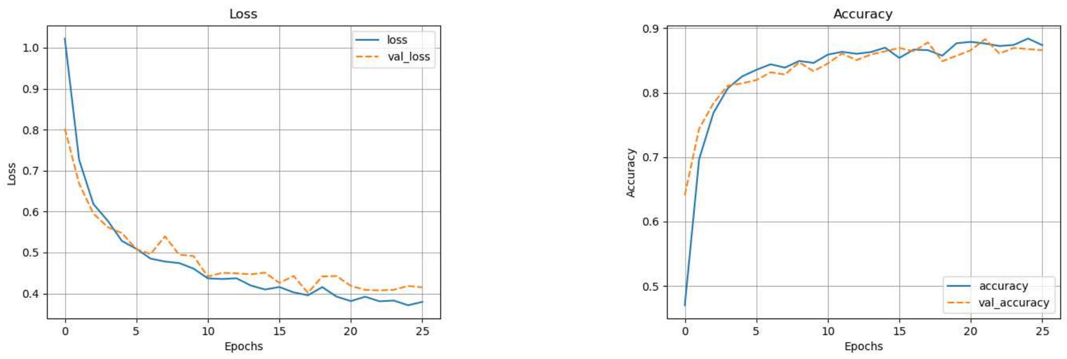


Figure 4.4: Loss and accuracy plots for InceptionResNetV2.

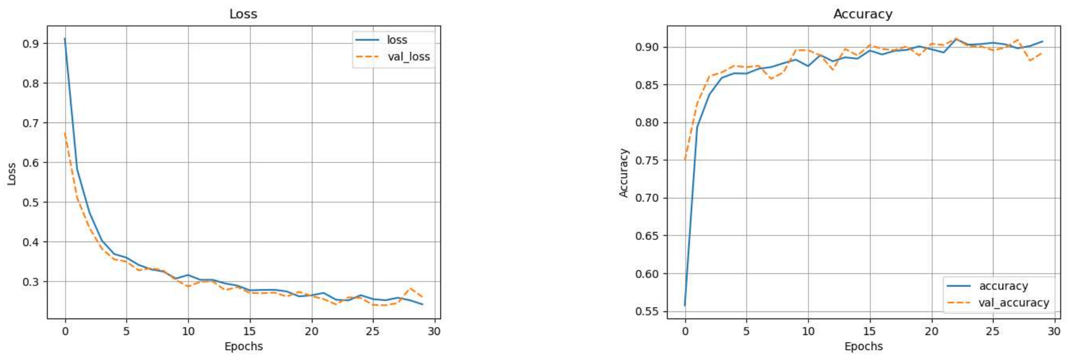


Figure 4.5: Loss and accuracy plots for DenseNet201.

The plots on the left represent the loss on the training set and validation set over the epochs. Having the two losses aligned means that the model is not overfitting and instead it is generalizing well the images on the dataset. Having a training loss that decreases rapidly but a validation loss that is increasing instead of decreasing means that the model is overfitting.
The plots on the right, instead, visualize the accuracy both on the training and validation sets over the epochs.
Plot analysis:

- ResNet50: the loss and accuracy plots for ResNet50 indicate moderate performance, with some fluctuation in the validation accuracy and loss over epochs. The model achieves an accuracy of 67.3%, indicating room for improvement, particularly in recall where it scores 51.2%.

- VGG16: the model shows better performance with an accuracy of 85.3%. The training and validation accuracy curves are closely aligned, suggesting good generalization. The loss and accuracy plots show consistent improvement over epochs. Also, the precision and recall are both high, indicating balanced performance across different classes.

- InceptionV3 and InceptionResNetV2: these models perform similarly close to VGG16 with a slightly better accuracy at 86.4% and 87.5% respectively. The precision-recall balance is well-maintained, and the models converge smoothly as seen in the loss and accuracy plots.

- DenseNet201: the best-performing model with a remarkable accuracy of 90.1%. Precision and recall are also very high.

The best performing model is DenseNet201 with an accuracy of 90.1% on the test set. We can see on the confusion matrix 4.6 the results for each class. For this project we associate value 0 as the *infesting* class, value 1 as the *terrain* class and value 2 as the *valeriana* class. The model easily distinguish the class *terreno* from the other classes; without too many surprises the errors occur mostly between the classes *infestanti* and *valeriana*: these classes are very similar to each other and the error in the classification can be explained on the similarity of the images on the dataset.
The average training time for the models is 4 minutes per models. VGG16 has the highest training time of 6min 7s followed by DenseNet201 with 4min 52s. As we can see, training time is not an issue for this dataset if we can parallelize the training on images using a GPU.

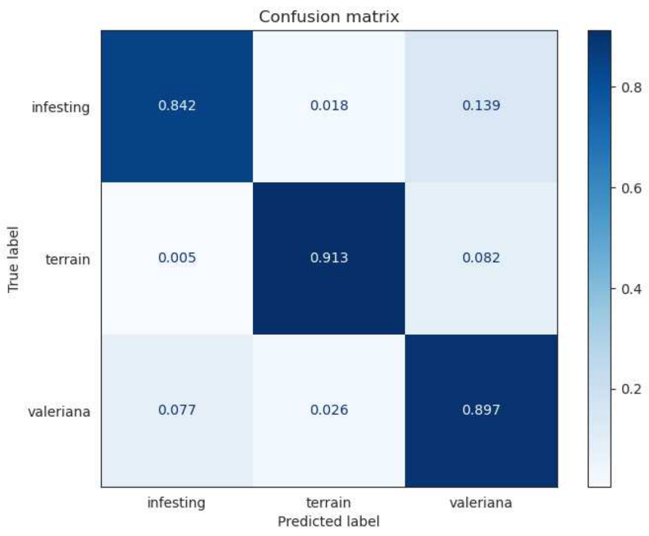


Figure 4.6: Confusion matrix of DenseNet201 on first dataset test set. Values in the confusion matrix are normalized.

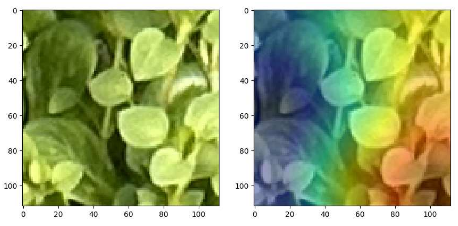


Figure 4.7: Heatmap visualization on a single patch using DenseNet201. The model clearly is able to distinguish the prominent features of the image.

After training, the model is used to classify the entire image. The process follows a similar approach to the dataset creation as described in section 3.3. Specifically, the images are segmented into patches, each with dimensions of 112 x 112 pixels. Subsequently, each individual patch is classified using the trained model (see figure 4.8 as an example). Currently, the results are only qualitative because we lack a test to evaluate the performance of the full image classification. Section 4.5 will attempt to solve this.

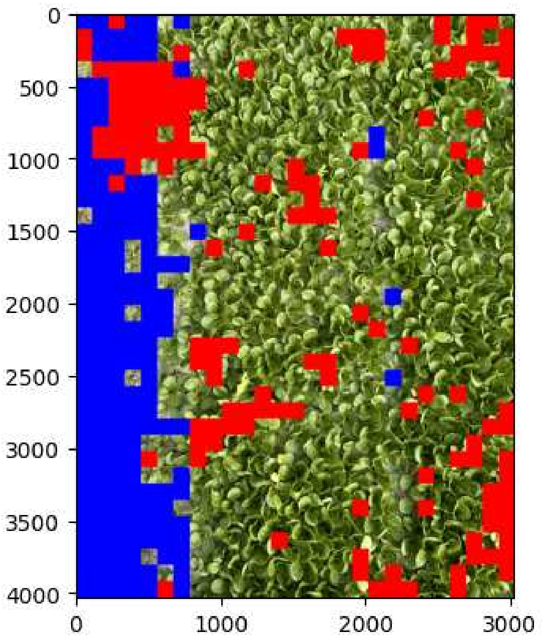


Figure 4.8: Total image classification using DenseNet201. The red patches represent the *infesting* class, the blue patches the *terrain* class.

4.3 Dataset v2

Table 4.2: Model performance comparison on second dataset. The values are calculated on the test set.

Model	Accuracy	Precision	Recall	Time
ResNet50	70.3%	72.1%	67.3%	10min 5s
VGG16	86.4%	87.8%	84.3%	11min 56s
InceptionV3	87.2%	87.8%	86.8%	11min 34s
InceptionResNetV2	81.0%	90.0%	71.5%	13min 38s
DenseNet201	89.6%	89.9%	89.3%	12min 44s

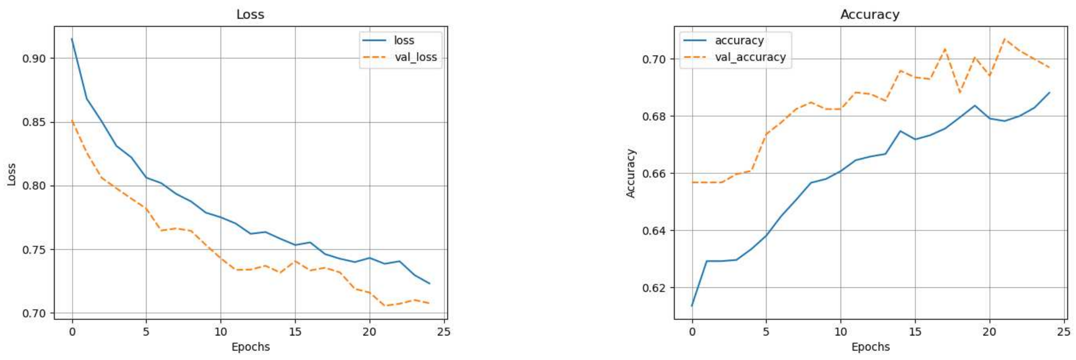


Figure 4.9: Loss and accuracy plots for ResNet50.

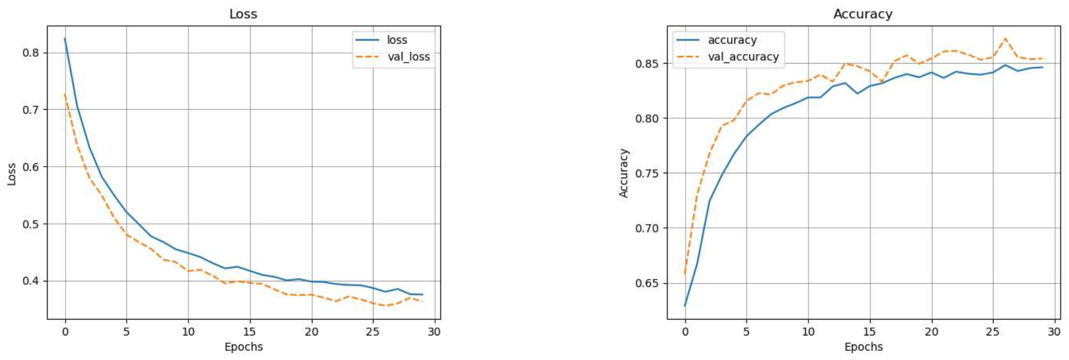


Figure 4.10: Loss and accuracy plots for VGG16.

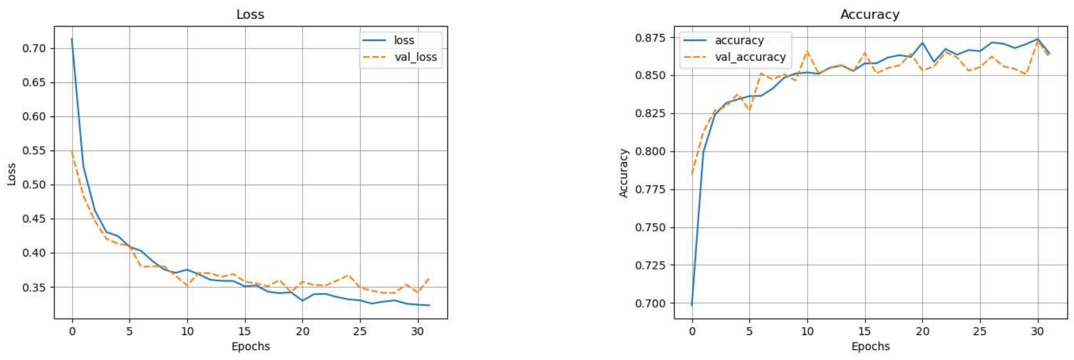


Figure 4.11: Loss and accuracy plots for InceptionV3.

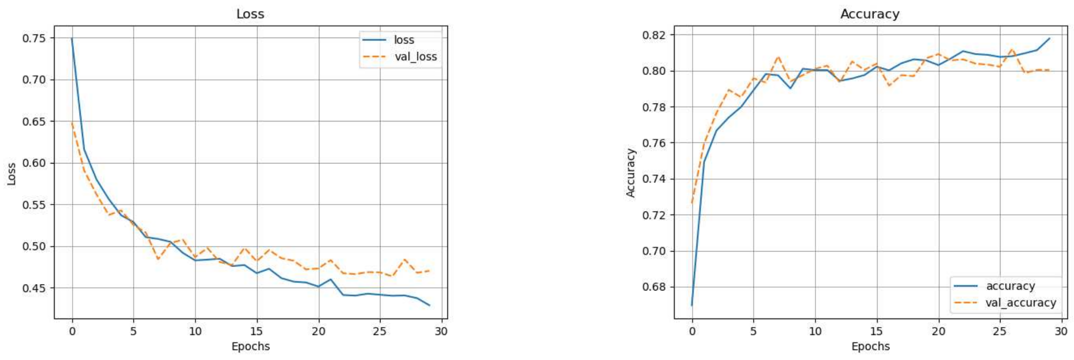


Figure 4.12: Loss and accuracy plots for InceptionResNetV2.

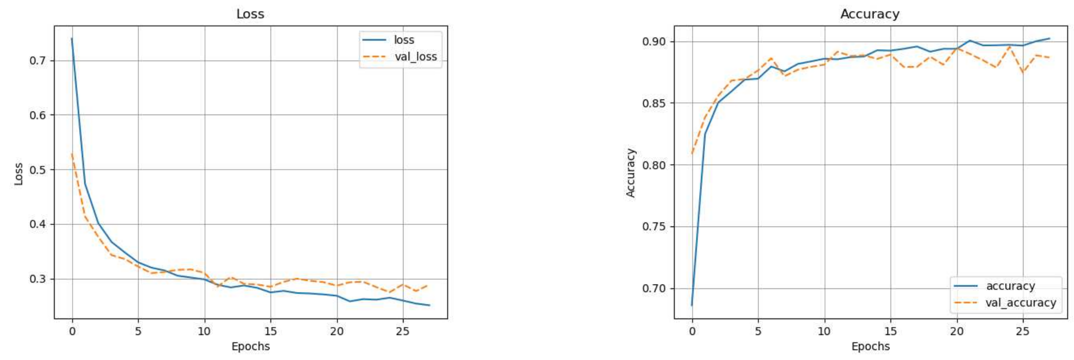


Figure 4.13: Loss and accuracy plots for DenseNet201.

The plot analysis reveals that DenseNet201 consistently outperforms the other models across both datasets, demonstrating superior accuracy, precision, and recall. The training and validation curves for DenseNet201 show smooth convergence, indicating robust learning and generalization. ResNet50, while improving slightly with *dataset-v2*, remains the least accurate, particularly struggling with recall. VGG16 and InceptionV3 perform comparably well, with close alignment of training and validation metrics, suggesting effective learning. Inception-ResNetV2 exhibits strong performance with balanced precision and recall, though slightly less accurate than DenseNet201.

If we inspect the confusion matrix of ResNet50 4.14, we see that the model tended to classify all the patches in one single class, in this case *valeriana*. This can be a problem in unbalanced datasets (like the one we are using now) because a model can get high accuracy by just classifying all images with one label. This problem can be solved by using deeper models, in this case DenseNet.

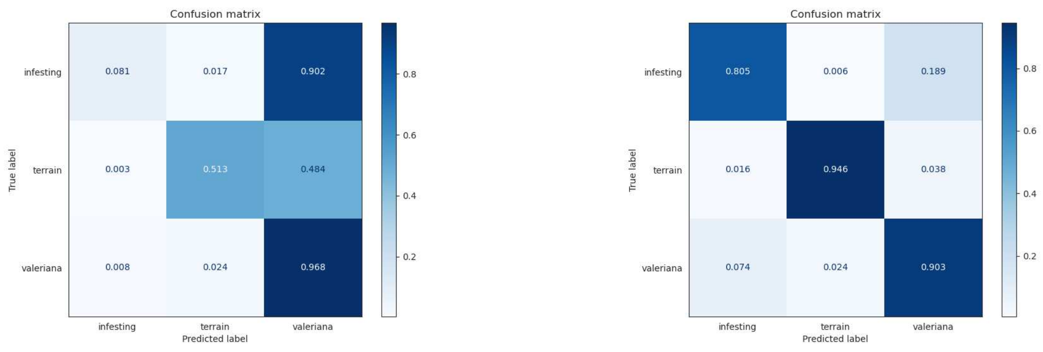


Figure 4.14: Comparison between the confusion matrices of ResNet50 and DenseNet201. On the left: the confusion matrix of ResNet50. On the right: the confusion matrix of DenseNet201. Both models were trained on the *dataset-v2*.

4.4 Dataset v3

Table 4.3: DenseNet201 performances on *dataset-v3* and *dataset-v3-reduced-50*. The values are calculated on the test set.

Dataset	Accuracy	Precision	Recall	Time
dataset-v3-reduced-50	90.3%	90.3%	90.1%	33min
dataset-v3	92.5%	92.5%	92.4%	-

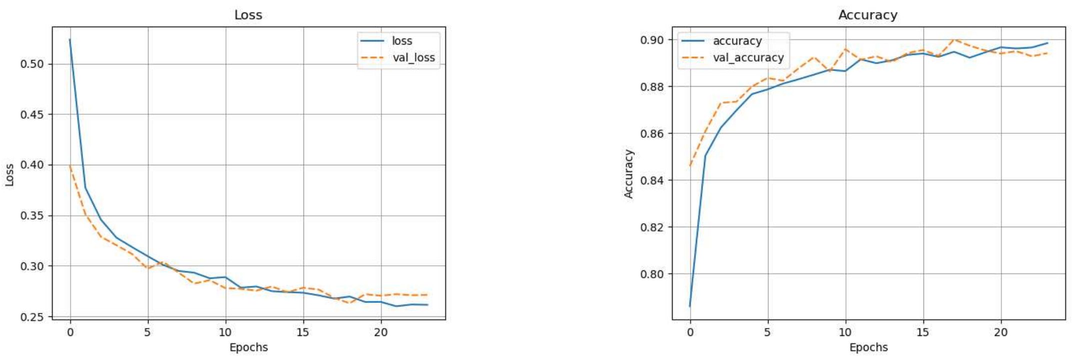


Figure 4.15: Loss and accuracy plots for DenseNet201 trained on *dataset-v3-reduced-50*.

For *dataset-v3-reduced-50*, only DenseNet201 was trained and tested, as it had been the best model for all the previous datasets. In this case, our goal is to verify if the results observed earlier remain consistent with a larger dataset.
We can see that the results for the model are very similar compared to the same performances on the previous datasets. For example, the aggregated accuracy for all the classes is 90.3% for DenseNet201 trained on *dataset-v3-reduced-50* and 89.6% for the same model trained on *dataset-v2*.
Considering only the *infesting* class, 4.16 the model achieved a precision of 92.5% and a recall of 73.1%, which combined give a F1-score of 81.7% (these values are calculated on the unnormalized confusion matrix, see 2.6 for reference).
For the full dataset *dataset-v3*, where data balancing was not applied, we have overall a higher accuracy of 92.5% due to the fact that we have many samples from the *valeriana* class, which increases the performances. In fact, if we inspect each class individually we see that for the *infesting* class we have lower performances: the precision is 90.2% and the recall is 65.7%. The F1-score is 76.0% which is lower of the value calculated in the reduced dataset. On the contrary,

for the *valeriana* class, the values of precision, recall and F1-score are respectively 93.1%, 97.8% and 95.4%.

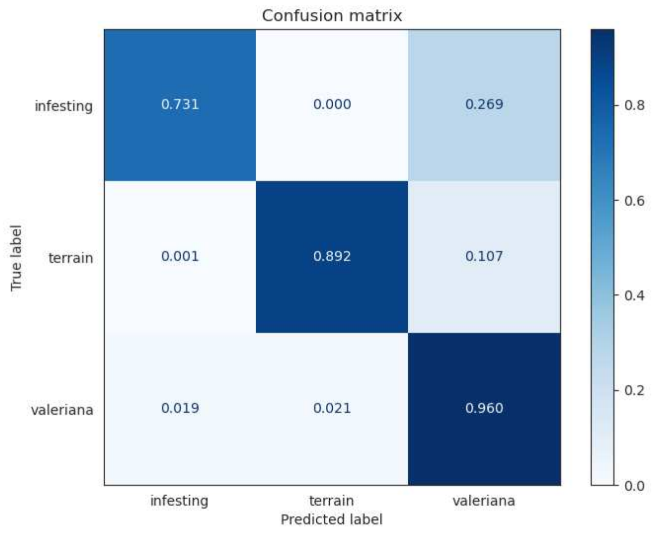


Figure 4.16: Confusion matrix of DenseNet201 on *dataset-v3-reduced-50* test set. Values in the confusion matrix are normalized.

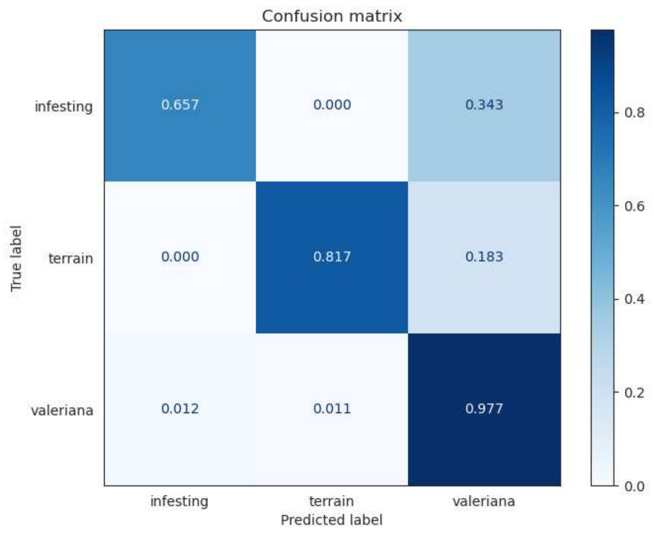


Figure 4.17: Confusion matrix of DenseNet201 on *dataset-v3* test set. Values in the confusion matrix are normalized.

4.5 Tuning methodologies

One of the goal of this project is to research and analyze the methodologies for reducing the number of false positives and false negatives after classification. In in our agricultural setting these classification errors can lead to many problems:

- A false positive occur when the true label of an image is not *infesting* but we classify if as the latter. This means that we wrongly identify a crop as an infesting plant. This error can lead to false alarms; if we alert the specialist to check for infesting plants but none are present, it results in a waste of time and resources. The metric used to visualize the rate of false positives is precision: a high precision indicates a low number of false positives, instead a low precision indicates the contrary.

- A false negative occur when the true label of an image is *infesting* but we classify it as non infesting (we don't catch the infesting plant). This error can lead to undetected infestations spreading unchecked, potentially causing significant damage to crops and consequently increasing costs. The metric used to visualize the rate of false negatives is recall.

Precision and recall should be balanced, but it often occurs that when we try to increase one metric, the other decreases. To address this trade-off, we can use the F1-score, which is the harmonic mean of precision and recall. The F1-score provides a single metric that balances both precision and recall, giving us a more comprehensive evaluation of the model's performance. By optimizing for the F1-score, we ensure that our model maintains a good balance between precision and recall. After the classification 4 methodologies were applied to modify the level of classification on infesting plants:

1. Modify the dataset.

2. Modify the model during training.

3. Apply a different threshold during the classification of a single patch.

4. Apply post-processing on the full image after classification.

Total image test. The tuning methods were evaluated using a test image that was manually labeled. The labeling process (illustrated in 4.18) is similar to the process explained in subsection 3.3.2 (second method: manual segmentation and automatic labelling). After dividing the image in patches and labelling them, the values of the labels are saved on a matrix where each entry (determined by row and column indexes) represents a single patch. Each entry can have 2 values: 0 for *infesting* and 1 for *non infesting*. This matrix, which we are going to call *test label matrix* is used to evaluate the metrics for the total image classification. It is important to note that this test image was not used for the creation of *dataset-v1* or *dataset-v2*.
To evaluate the tuning methodologies, another images different from the test image previously described is divided in patches and each patch is classified using the previously trained model. Therefore, each patch can be labelled as *infesting*, *terrain* or *valeriana* and we create another label matrix. Using this label matrix and the test label matrix we count the number of true positives, false positives and false negatives only for the class *infesting*. From these values, we obtain precision, recall and F1-score. We calculate these values only on the *infesting* class because we are not focusing on the other classes for the following reason: the main goal of our analysis is to identify and manage the infesting plants. Therefore, our primary interest lies in the performance of the model with respect to detecting these infesting plants accurately and efficiently. Finally, for this test, we are not interested in reaching the same accuracy on the *dataset-v2* test set, because for the total image test we are only interested in detecting the infesting plants without being too accurate on predicting the complete bounding box. The figure 4.19 allows us to visualize all the bounding boxes of the infesting plants. As said before, these boxes are just indicative of the location of the infesting plants.

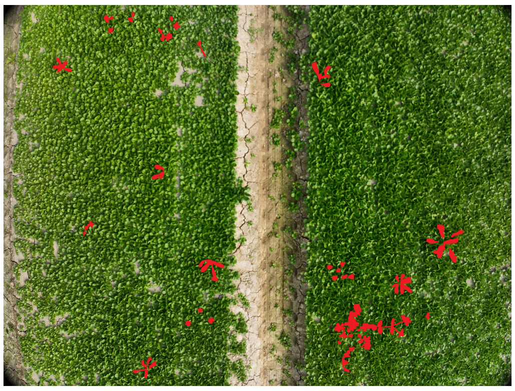


Figure 4.18: Manually segmented image.

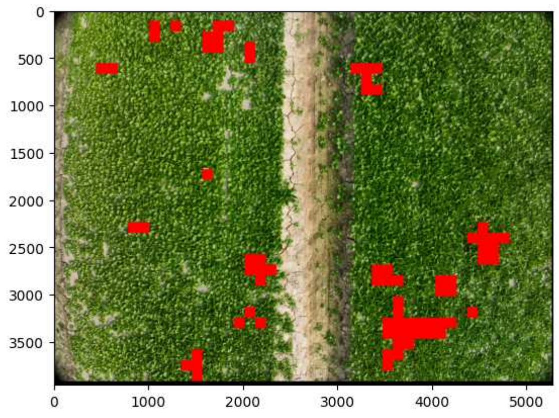


Figure 4.19: Segmented image using patches.

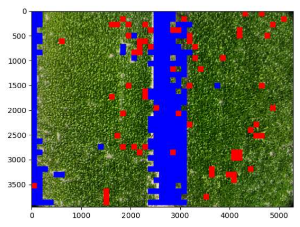
4.5.1 Tuning the dataset

As figure 4.14 shows, when the training data has a disproportionate number of instances in different classes, the model may become biased towards the majority class, leading to a higher number of false negatives or false positives in the minority class.
To solve this issue techniques such as oversampling the minority class, undersampling the majority class, and data augmentation can help balance the dataset and reduce bias [36].
In this test we tried undersampling the majority class and oversampling the minority classes. A test was conducted by comparing three models: a DenseNet201 trained on the *dataset-v2* where undersampling is applied to the *valeriana* class (the number of data points of the majority class was brought to the same number of the data points of the minority class), which from now we are going to refer as *undersampled-dataset-v2*, a DenseNet201 trained on the dataset-v2 where oversampling is applied to the minority classes (*infesting* and *terrain*) and a DenseNet201 trained on the usual *dataset-v2*. Oversampling is performed by sampling the minority classes with replacement: each time we draw a sample from the dataset, the original sample remains in the dataset. Thus, each data point can be drawn again in subsequent draws.
The performance metrics reveal notable differences. These results indicate that the first model, trained on the undersampled dataset, was more effective in correctly identifying positive instances, but it also had a higher number of false positives compared to the second model. The precision of the first model was 41.8%, which was lower than the second model's precision of 62.5%. However, the recall of the first model was significantly higher at 37.8% compared to the second model's recall of 20.3%. This suggests that while the model trained of the undersampled dataset might generate more false alarms, it is more reliable in identifying true positive cases.
The disadvantage of using undersampling is that we are deleting many samples from the *valeriana* class and therefore reducing the generalization capability of the model.
Using the oversampled dataset, we observe a different balance of performance metrics. The model trained on this dataset achieved a true positive count of 42, a significant improvement over the other models. However, this came with an increase in false positives, reaching 114, which greatly impacted its precision, resulting in a precision score of 26.9%. Despite this, the recall was the highest among the three models, at 56.8%, indicating that the oversampled model was the most effective at identifying true positive cases. The F1-score of 36.5% for the oversampled model falls between the scores of the other two models.
According to Buda et al. (2018) [36] while both oversampling and undersampling can be effective, oversampling generally performs better in the context of CNNs, especially when used

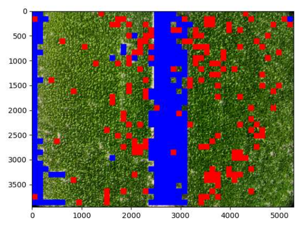
to completely eliminate class imbalance. Undersampling, though sometimes effective, has the drawback of discarding potentially useful data. While this can be true for many cases, oversampling in this case resulted in a very high number of false positives, leading to a classification where many data points are wrongly classified.
In the next sections, we will discuss other techniques that attempt to solve this task without interfering with the dataset.

Model	True positives	False negatives	False positives	Precision	Recall	F1-score
Undersampled dataset-v2	28	46	39	41.8%	37.8%	39.7%
Oversampled dataset-v2	42	32	114	26.9%	56.8%	36.5%
Original dataset-v2	15	59	9	62.5%	20.3%	30.6%

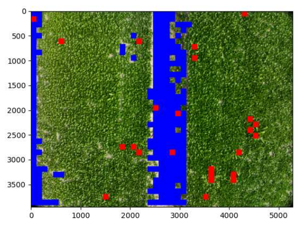
Table 4.4: Performance metrics on dataset tuning.



(a)



(b)



(c)

Figure 4.20: a) Segmentation done with model trained on undersampled dataset. b) Segmentation done with model trained on oversampled dataset. c) Segmentation done with model trained on original dataset.

4.5.2 Tuning the model

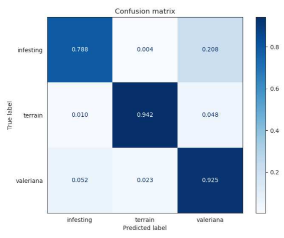
There are various ways to tune the level of false positive and false negative classifications by modifying the model parameters. One effective method is to adjust the class weights of the model. According to Johnson et al. (2009) [19] in imbalanced datasets, the model tends to be biased towards the majority class because it dominates the training process. To mitigate this, one can assign higher weights to the minority classes. By doing so, the learning algorithm penalizes misclassifications of minority class instances more heavily, thereby encouraging the model to improve its performance on these underrepresented classes. This approach ensures that the model pays more attention to the minority class during training, which can enhance recall but

often at the expense of precision. This is because the model becomes more aggressive in predicting the minority class, leading to more false positives.
In this test all the class weights were set to 1 and the DenseNet201 was trained on the full *dataset-v2*. Then, the weight for the class *infesting* was gradually increased and the model retrained on the same conditions. The results are shown in 4.6. We can see that increasing the weight of the minority class gradually increases the number of true and false positives and at the same time reduces the number of false negatives (recall improves at the cost of precision). This pattern continues until weight 3.5 and after that point there is not substantial improvement in the model; in fact, the F1-score slightly dropped at weight equal to 4.
The models were then tested on the complete image (total image test 4.5). Comparing the results from the two tables 4.6 and 4.8, we can observe how the model's performance on individual patches translates to performance on complete images. Increasing the weight of the minority class (*infesting*) improves recall in both scenarios, albeit at the cost of precision, which is consistent with our observations on individual patches.
For instance, in table 4.6, increasing the weight from 1.5 to 3.0 improves recall from 76.2% to 83.4%, but decreases precision from 84.1% to 74.4%. Similarly, in table 4.8, we see recall improve from 35.1% to 50.0% when the weight increases from 1.5 to 3.0, but precision drops from 48.2% to 34.3%. Additionally, the F1-score shows a peak at different weight values in both tests. For individual patches, the F1-score peaks at 1.5 with a value of 80.0%, whereas for complete images, it peaks at a weight of 3.0 with a value of 40.7%. This suggests that the optimal class weight might differ based on whether the model is evaluated on patches or complete images.
In conclusion, tuning the class weights enhances recall by increasing the number of true positives but also raises the number of false positives, thereby reducing precision, thus confirming the results found by Johnson et al. (2009) [19]. This behavior is evident both in the patch test and in the total image test.

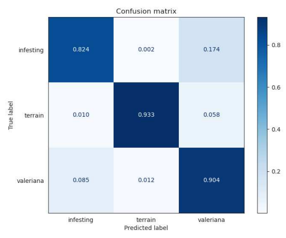
Table 4.5: Model performance comparison with different weights. The values are calculated on the test set.

Weight	True positives	False negatives	False positives	Precision	Recall	F1-Score
1.0	340	131	54	86.3%	72.2%	78.6%
1.5	359	112	68	84.1%	76.2%	80.0%
2.0	369	102	85	81.3%	78.3%	79.8%
2.5	388	83	118	76.7%	82.4%	79.4%
3.0	393	78	135	74.4%	83.4%	78.7%
3.5	396	75	135	74.6%	84.1%	79.0%
4.0	394	77	155	71.8%	83.7%	77.3%

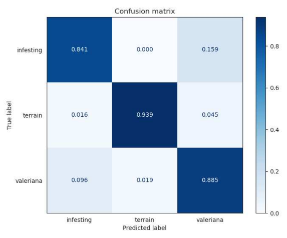
Table 4.6: Model performance metrics for different weights. The metrics were evaluated using the test set of *dataset-v2*.



(a)



(b)



(c)

Figure 4.21: Confusion matrix comparison between different tuned models. a) Class weight *infesting* set to 1.5. b) Class weight *infesting* set to 2.5. c) Class weight *infesting* set to 3.5. Models were trained on dataset-v2.

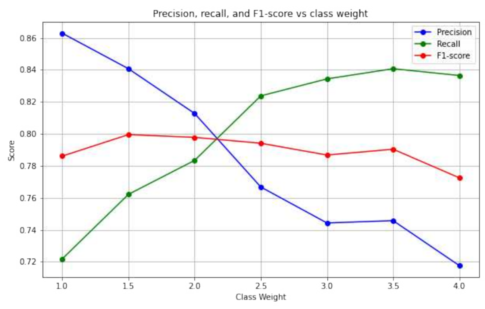


Figure 4.22: Combined plot with precision, recall, and F1-score against the class weight using the data from table 4.6.

Table 4.7: Model performance metrics for different weights. The metrics were evaluated using the total image test.

Weight	True positives	False negatives	False positives	Precision	Recall	F1-Score
1.5	26	48	28	48.2%	35.1%	40.6%
2.0	23	51	36	39.0%	31.1%	34.6%
2.5	30	44	51	37.0%	40.5%	38.7%
3.0	37	37	71	34.3%	50.0%	40.7%
3.5	31	43	88	26.1%	41.9%	32.1%
4.0	38	36	94	28.8%	51.4%	36.9%

Table 4.8: Model performance metrics for different weights. The metrics were evaluated using the total image test.

4.5.3 Tuning the results

To further adjust the level of classification one can directly apply a different threshold over the classification of a single patch. The softmax layer assigns to each image a percentage for each label and these percentages add up to 1; these values indicate how much the model is certain that the image belongs to a specific class.
To apply a different threshold on classification means that the model will not choose the class with highest percentage, but it will, first of all, check if the given class surpass the value to assign that label and, if not, it will choose the class with highest percentage from the remaining classes. Using this method, we can make the model more *aggressive* by applying a low threshold, allowing it to detect infesting plants with the slightest presence. Conversely, we can make the model more *conservative* by increasing the threshold, ensuring it only detects infesting plants when it is very certain.
For example, when the trained model classifies the image 4.23 the predictions are:

$$\begin{bmatrix} 0.46483302 \\ 0.00149037 \\ 0.5336766 \end{bmatrix}$$

Without setting a threshold, the model classifies the patch as *valeriana* since this class has the highest probability at 53.4%. However, if we lower the threshold for the *infesting* class to 0.4, the model will label the image as *infesting* because its probability of 46.5% exceeds the 40% threshold. Conversely, if we set the threshold to 0.5, the model will not classify the image as *infesting* and will instead select the class with the next highest probability from the other classes. The table 4.10 and the accompanying graph 4.24 summarize the performance metrics

Figure 4.23: Patch used in the example.

of a model tested on an image, where different thresholds were applied to the output probabilities. The metrics considered include precision, recall, and F1-score for thresholds ranging from 0.1 to 0.9.
As the threshold increases, precision improves from 5.9% to 100%, while recall decreases from 86.8% to 2.6%. The F1-score, which balances precision and recall, peaks at an intermediate

threshold (0.4) with a value of approximately 38.0%. The F1-score maintains a high value between 0.4 and 0.7. Beyond this interval, the F1-score drops rapidly.
Low thresholds (0.1 - 0.3) yield high recall but low precision, intermediate thresholds (0.4 - 0.6) provide a balance between precision and recall, and high thresholds (0.7 - 0.9) result in high precision but low recall.
Finally, the graph 4.25 represents the the precision-recall curve based on the data present in the table 4.10. The curve helps visualize how the model's performance varies with different classification thresholds, allowing us to draw the same conclusions as with the graph 4.24.

Table 4.9: Performance metrics at different thresholds. The metrics were evaluated using the total image test.

Threshold	True positives	False negatives	False positives	Precision	Recall	F1-score
0.1	33	5	525	5.9%	86.8%	11.1%
0.2	31	7	282	9.9%	81.6%	17.7%
0.3	30	8	159	15.9%	78.9%	26.4%
0.4	26	12	73	26.3%	68.4%	38.0%
0.5	18	20	34	34.6%	47.4%	40.0%
0.6	13	25	17	43.3%	34.2%	38.2%
0.7	11	27	7	61.1%	28.9%	39.3%
0.8	7	31	1	87.5%	18.4%	30.4%
0.9	1	37	0	100.0%	2.6%	5.1%

Table 4.10: Performance metrics at different thresholds. The metrics were evaluated using the total image test.

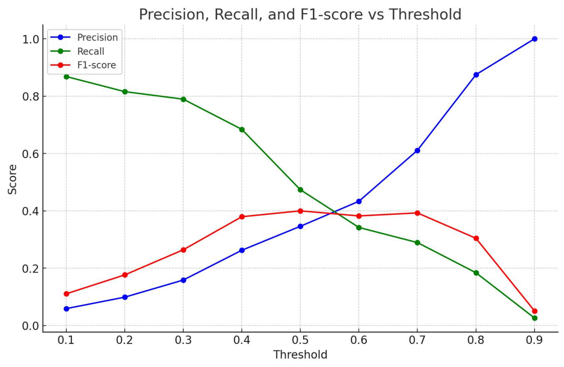


Figure 4.24: Combined plot with precision, recall, and F1-score against the threshold.

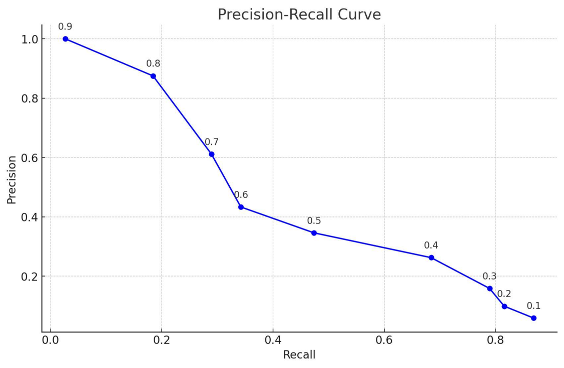


Figure 4.25: Precision-recall curve based on the data present on the table 4.10. Each point on the curve represents a different threshold value, and the annotations indicate the corresponding threshold.

4.5.4 Tuning through post-processing

Finally, we can work on the full image directly by doing post processing after the segmentation. There are several ways to implement this method but the simplest one is to map the patches as a matrix and, for each entry of the matrix, assign a value based on the label. After we have done this, we can implement simple processing techniques, for example:

1. Filter single red patches. Single points can be seen just as misclassification errors.

2. Filter red patches that are near the terrain. The model tends to overclassify the patches near the terrain as *infesting*. Patches near the terrain are less important to detect so we can apply this operation to remove them.

Using the first method, the precision improved significantly to 60.0%, but the recall dropped to 32.4%, yielding an F1-score of 42.1%. For the second method, the precision increased to 50.0% while maintaining the same recall of 37.8% resulting in an F1-score of 43.1%.
These variations highlight the trade-offs between precision and recall across different methods of handling the dataset, with the the second one achieving the best balance as indicated by its highest F1-score. However, it is evident that these operations can only reduce the number of false positives, thus improving only the precision.

Scenario	True positives	False negatives	False positives	Precision	Recall	F1-score
Test image	28	46	39	41.8%	37.8%	39.7%
Post-processing 1	24	50	16	60.0%	32.4%	42.1%
Post-processing 2	28	46	28	50.0%	37.8%	43.1%

Table 4.11: Performance metrics for different post-processing methods.

4.6 Comparison between CPU and GPU performances.

In this brief section we will explore the advantages of using a GPU versus a CPU on these tasks:

1. Training the model on dataset-v2.

2. Classifying a new image that was not used for the creation of the datasets.

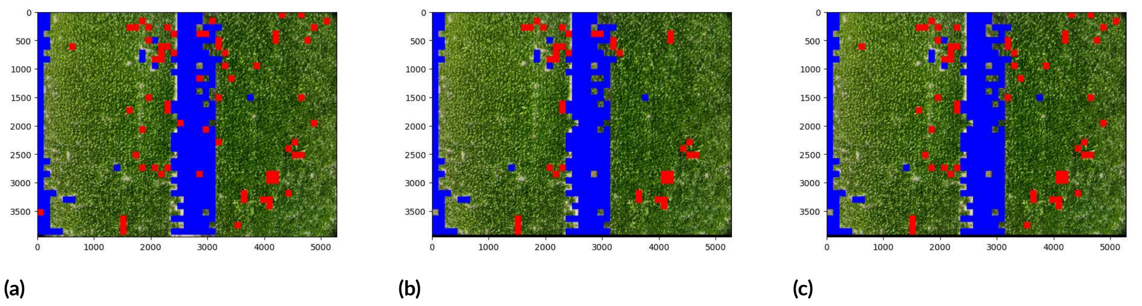


Figure 4.26: Example of post processing on the entire image. a) Segmentation done with model trained on balanced dataset. b) Filtering of single red patches. c) Filtering red patches near the terrain.

For these tasks the GEFORCE GTX 1050 GPU was used. For the first task, we see that if we train DenseNet201 without a GPU, the training time requires 34min 55s. By using a GPU, we reduce the time to 12min 44s, resulting in a 63.5% reduction of training time.
For the second task, we still use DenseNet201 to classify the entire image, by dividing it in patches and classifying every patch singularly using the model. To implement the GPU classification we employ batch processing, which is a method of executing multiple data processing tasks simultaneously as a single group, or batch, instead of handling each task one by one.
Without the GPU, the time required is 6min 20s and with the GPU the time is equal to 20s, reducing the classifying time by 94.7%. As we can see, this approach is particularly beneficial in deep learning, where GPUs can handle large batches of data efficiently, leading to significant time savings.
Batch processing reduces time with a GPU because it leverages the parallel processing capabilities of the GPU, which is designed to handle thousands of operations simultaneously. By processing a large group of data points (such as image patches) in a single batch, the GPU can efficiently distribute the computational workload across its many cores, significantly reducing the overhead associated with individually processing each data point.
In conclusion, the use of GPUs is very important during classification because during a field operation we require to classify a vast number of images collected from the drone, reducing the time to classify each image makes the task feasible for the company.

5 Discussion and future directions

5.1 Research findings

Best model. The research presented in this thesis demonstrates the potential of Convolutional Neural Networks (CNNs) for the identification of infesting plants in monoculture fields. By leveraging high-resolution imagery from Unmanned Aerial Vehicles (UAVs) and deep learning techniques, we developed a system capable of distinguishing between crops and various infesting plant species with high accuracy.

The analysis demonstrated that DenseNet201 outperformed other models like ResNet50, VGG16, InceptionV3, and InceptionResNetV2. DenseNet's architecture, characterized by its dense connectivity pattern, allows each layer to receive direct input from all preceding layers. This connectivity pattern ensures maximum information flow between layers, which helps in better gradient propagation and mitigates the vanishing gradient problem often encountered in deep networks.

This feature is particularly beneficial for our task of distinguishing between crops and infesting plants, which can be highly variable in appearance and often intermingled within the same field. The dense connections enable the model to learn more complex features and representations, making it more adept at differentiating between subtle differences in plant structures and textures. We can see the advantages of deeper models:

- Feature hierarchies: deeper models can learn more abstract and high-level features. In the context of plant identification, this means the model can better recognize intricate patterns and distinctions between different plant types, which is essential for accurately identifying infesting plants among crops.

- Improved learning: DenseNet's approach of connecting each layer to every other layer encourages feature reuse, which improves learning efficiency and model performance. This reduces the risk of overfitting, even with relatively smaller datasets, by ensuring that all layers contribute to learning robust features.

- Gradient flow: the direct connections between layers in DenseNet ensure that gradients flow more smoothly during backpropagation, facilitating the training of very deep networks. This is crucial for capturing the detailed and varied patterns present in UAV imagery of agricultural fields.

False positives and false negatives. The exploration of various tuning methodologies has highlighted the relationship between precision and recall in the classification of infesting plants in agriculture. By modifying the dataset, training parameters, classification thresholds, and applying post-processing techniques, the research aimed to balance the reduction of false positives and false negatives. The results showed that different approaches have distinct impacts on model performance.
Undersampling the majority class improved recall but lowered precision. It is advisable to use this technique only when necessary, that is to say when the ration between *infesting* and *valeriana* classes is higher than 1:3, because if we delete data points from the dataset we may reduce the generalization capabilities of the dataset.
Adjusting the model class weights also demonstrated a trade-off between precision and recall, with an optimal balance achieved at specific weight settings. Applying a threshold on the predictions allowed tuning of the model's aggressiveness in detecting infesting plants, demonstrating that lower thresholds increased recall at the cost of precision, and vice versa. Thresholding can be considered the best method for this task because it is the easiest to apply, as it does not require modifying the dataset or the model and because it allows us to achieve various levels of precision and recall with little intervention.
Finally, post-processing techniques significantly improved precision but did not enhance recall. Overall, the methodologies underscore the necessity of a tailored approach depending on the specific agricultural application and the relative importance of precision versus recall. The com-

prehensive evaluation using the F1-score provided a balanced metric, guiding us in optimizing the model for effective strategies for detecting infesting plants.

5.2 Challenges and limitations

One of the significant challenges faced during this study was the preparation and preprocessing of the dataset. The high variability in plant appearances and the similarities between the crops and the infesting plants required extensive data augmentation and careful labeling to ensure the model's effectiveness. Balancing the dataset to prevent class imbalance was also crucial in achieving reliable performance.
Despite the promising results, there are some limitations to our approach. The performance of the CNN models heavily depends on the quality and quantity of the training data. Inconsistent data quality due to varying weather conditions or UAV flight patterns can affect the model's accuracy. To solve this issue one has to label a large quantity of images that comprise the different stages of development of infesting plants and different lightning conditions; furthermore it would be helpful to test the model on different altitudes inside and outside the greenhouse.
Additionally, the computational resources required for training and deploying these models can be significant and may limit fast processing of the images. Having a GPU to train the model is mandatory to save time and to test different models on diverse datasets quickly. In addition to that, the classification on the full image also can't be done sequentially (patch classification) if the desired goal is cover large areas of the field using UAVs; for this reason the use of GPUs is necessary for this purpose.
It is also suggested not to further reduce the dimension of the patches. 112x112 pixel patches are a good compromise for both training and evaluation because they can capture the presence of infesting plants in a very precise ways; having larger patches can lead to images that capture too much information difficult to classify, on the contrary having smaller patches leads to the opposite problem: it is difficult to understand the class of a small patch. For this reason, the final segmentation can't be more precise.

5.3 Future work

Dataset. Future work should focus on improving the generalization capabilities of DenseNet by incorporating more diverse datasets that include different crop types, infesting plant species, and environmental conditions. Applying transfer learning techniques and domain adaptation strategies can also improve model robustness across various agricultural settings.

Models. Investigating more advanced CNN architectures, such as EfficientNet or vision transformers [37], could potentially yield better performance. These models have demonstrated state-of-the-art results in various image recognition tasks and may offer further improvements in detecting and distinguishing infesting plants.

Classification. A natural progression of this project should be classifying the different species of infesting plants instead of classifying all of them as *infesting*.
Another contribution can be on solving this problem directly performing detection or segmentation, without the need of patch extraction. Advanced detection models like YOLO [38] are able to recognize objects inside an image in a easy manner; YOLO is an object detection algorithm known for its speed and accuracy, and its integration with UAVs has led to significant advancements in various fields, including agriculture [39]. However, these models require a high amount of labelled data, and without the use of patches one would have to manually label thousand of images.

Temporal dependencies. One significant area for future work is to address the limitation related to the assumption of independence in daily pest detection probabilities. In this thesis, we assumed that the probability of detecting a pest on the second day is independent of whether it was detected on the first day. This simplification, while useful for initial model development, overlooks the potential temporal dependencies that could exist between successive days of crop analysis. Future research should focus on developing and integrating models that account for these temporal dependencies.

Real-time processing. Also developing methods for real-time image processing and analysis is essential for practical deployment. Implementing edge computing solutions where UAVs process images on-board and provide immediate feedback to farmers could significantly enhance the system's usability. Additionally, integrating this technology with existing precision agriculture platforms could facilitate comprehensive crop management and decision-making.

End-user feedbacks. Most importantly conducting extensive field trials and gathering feedback from end-users, such as farmers and agronomists, will provide valuable insights into the system’s effectiveness and usability. Iterative improvements based on real-world feedback can help refine the technology and make it more user-friendly.

6 Conclusion

This thesis investigates the application of Convolutional Neural Networks (CNNs) for identifying infesting plants in monoculture fields using high-resolution imagery from Unmanned Aerial Vehicles (UAVs). The primary goal was to enhance crop monitoring and infesting plant management practices through image recognition technologies.

The dataset creation was one of the critical components of this study. A comprehensive dataset comprising images of crops and various infesting plant species in monoculture fields was constructed. This dataset provided a robust foundation for training and evaluating the CNN models. The use of UAVs for image acquisition offered several advantages, including the ability to capture high-resolution images over large areas, which is essential for effective crop monitoring.

In terms of model development, several CNN architectures were explored. Among these, DenseNet showed promising results due to its dense connectivity patterns that help in mitigating the vanishing gradient problem and improving feature propagation. The transfer learning approach, which involved fine-tuning pre-trained models on our specific dataset, proved to be effective in achieving high classification accuracy.

Despite the promising results, several challenges were encountered during the research. One of the primary challenges was the variability in image quality due to different environmental conditions such as lighting, shadows and the different stages of growth of infesting plants. These factors can significantly affect the performance of the models. To address this, various image preprocessing techniques, such as augmentation, were applied to improve the robustness of the models.

Another challenge was the class imbalance in the dataset, where the number of images of certain infesting plant species was significantly lower than others. This issue was addressed using techniques such as undersampling, to balance the dataset and improve the model's ability to detect minority classes.
The research also highlighted the importance of considering false positives and false negatives in the evaluation process. By analyzing the trade-offs between these types of errors, the models were fine-tuned to achieve an optimal balance that minimizes both false positives and false negatives.
In conclusion, this thesis presents a step towards the use of deep learning techniques for improving agricultural practices. By addressing the challenges and exploring future directions, the research lays the groundwork for developing practical, efficient, and scalable solutions for crop monitoring management. The findings contribute to the broader goal of promoting sustainable agriculture.

References

[1] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and electronics in agriculture*, vol. 147, pp. 70–90, 2018.

[2] C. Zhang and J. M. Kovacs, "The application of small unmanned aerial systems for precision agriculture: a review," *Precision agriculture*, vol. 13, pp. 693–712, 2012.

[3] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss, "Uav-based crop and weed classification for smart farming," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3024–3031.

[4] Y.-c. Wu and J.-w. Feng, "Development and application of artificial neural network," *Wireless Personal Communications*, vol. 102, pp. 1645–1656, 2018.

[5] G. Ian, "Deep learning-ian goodfellow, yoshua bengio, aaron courville-google books," 2016.

[6] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[7] M. Egmont-Petersen, D. de Ridder, and H. Handels, "Image processing with neural networks—a review," *Pattern recognition*, vol. 35, no. 10, pp. 2279–2301, 2002.

[8] S. Bhattacharyya, "A brief survey of color image preprocessing and segmentation techniques," *Journal of Pattern Recognition Research*, vol. 1, no. 1, pp. 120–129, 2011.

[9] G. Kumar and P. K. Bhatia, "A detailed review of feature extraction in image processing systems," in *2014 Fourth international conference on advanced computing & communication technologies*. IEEE, 2014, pp. 5–12.

[10] R. Chauhan, K. K. Ghanshala, and R. Joshi, "Convolutional neural network (cnn) for image detection and recognition," in *2018 first international conference on secure cyber computing and communication (ICSCCC)*. IEEE, 2018, pp. 278–282.

[11] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in neural information processing systems*, vol. 27, 2014.

[12] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*. PMLR, 2014, pp. 647–655.

[13] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[14] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, and S. Lehmann, "Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm," *arXiv preprint arXiv:1708.00524*, 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[17] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[18] J. Cook and V. Ramadas, "When to consult precision-recall curves," *The Stata Journal*, vol. 20, no. 1, pp. 131–148, 2020.

[19] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.

[20] W. A. León-Rueda, C. León, S. G. Caro, and J. G. Ramírez-Gil, "Identification of diseases and physiological disorders in potato via multispectral drone imagery using machine learning tools," *Tropical Plant Pathology*, pp. 1–16, 2021.

[21] D. Pujari, R. Yakkundimath, and A. S. Byadgi, "Svm and ann based classification of plant diseases using feature reduction technique," *IJIMAI*, vol. 3, no. 7, pp. 6–14, 2016.

[22] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science*, vol. 7, p. 215232, 2016.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[25] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and electronics in agriculture*, vol. 145, pp. 311–318, 2018.

[26] F. S. Ishengoma, I. A. Rai, and S. R. Ngoga, "Hybrid convolution neural network model for a quicker detection of infested maize plants with fall armyworms using uav-based images," *Ecological Informatics*, vol. 67, p. 101502, 2022.

[27] P. K. Yadav, J. A. Thomasson, R. Hardin, S. W. Searcy, U. Braga-Neto, S. C. Popescu, D. E. Martin, R. Rodriguez, K. Meza, J. Enciso *et al.*, "Detecting volunteer cotton plants in a corn field with deep learning on uav remote-sensing imagery," *Computers and Electronics in Agriculture*, vol. 204, p. 107551, 2023.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[30] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[31] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[33] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[34] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[35] D. Shen, G. Wu, and H.-I. Suk, "Deep learning in medical image analysis," *Annual review of biomedical engineering*, vol. 19, pp. 221–248, 2017.

[36] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural networks*, vol. 106, pp. 249–259, 2018.

[37] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[38] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of yolo algorithm developments," *Procedia computer science*, vol. 199, pp. 1066–1073, 2022.

[39] C. Chen, Z. Zheng, T. Xu, S. Guo, S. Feng, W. Yao, and Y. Lan, "Yolo-based uav technology: A review of the research and its applications," *Drones*, vol. 7, no. 3, p. 190, 2023.

Acknowledgments

During my internship and subsequently for my thesis research I had the pleasure to work with Dromt. During this period where I worked closely together with the company I had the opportunity to learn from talented people and to familiarize with the startup's working environment. I would like to thank my colleagues in Dromt for helping me researching and providing me the appropriate technology, helping creating the dataset from images captured from their drones and finally for the assistance in writing this thesis.