Development of a machine learning plant growth estimator for hydroponics

SA Ngorima

orcid.org/0000-0002-0775-3529

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor: Prof ASJ Helberg

Graduation: May 2022
Student number: 36450057

Declaration

I, Simbarashe Aldrin Ngorima, hereby declare that the dissertation entitled "Development of a machine learning plant growth estimator for hydroponics" is my original work and has not already been submitted to any other university or institution for examination.

Simbarashe Aldrin Ngorima

Student Number: 36450057

Signed on the 9th day of December 2021

Acknowledgements

Firstly, I would like to thank God for the strength and wisdom to complete this study.

My supervisor, Prof Albert Helberg, thank you for your guidance, patience, and dedication towards your students. You taught me much more than writing an excellent dissertation.

I also would like to thank Dr Michel Nhambura for his mentorship and motivation. Your input masked my weaknesses, challenged my frontiers, and had a significant impact, which I genuinely appreciate.

Thank you for all the prayers, love and motivation, mum. With your eternal inspiration, I made the most out of this opportunity.

Thank you for the years of camaraderie support to my friend Justice Mallen. The academic journey wouldn't have been easy without your company.

Lastly, I would like to acknowledge the National Research Foundation (NRF) and the North-West University for assisting me financially throughout the research study.

Abstract

Growth stage estimation in indoor farming is crucial for precision agriculture. It reduces the wastage of resources such as nutrients and water in plant cultivation. Overly, yield and quality of the agricultural produce are improved while minimising running costs. However, plants at different growth stages pose similar morphological shapes, making growth stage estimation complex.

Therefore, an algorithm that can investigate other features of plant leaf besides shape is needed to distinguish plant growth stages accurately. In this work, we suggested three approaches: first, a machine learning approach that combines morphological operators to generate a contour mask dataset that highlights the morphological leaf structures, Gabor filters for feature extraction, and traditional machine learning algorithms for classification, second, a deep learning approach through transfer learning, and third, an approach that uses a feature extractor based on deep learning techniques and a classifier based on traditional machine learning techniques. Side-by-side experiments are run to evaluate the performance of the three proposed approaches In addition, two datasets (canola and radish) were created from a publicly available dataset as "bccr-segset" to develop and test these approaches.

We found that the third approach of using a feature extractor based on deep learning techniques and a classifier based on traditional machine learning techniques can classify four plant growth stages with the highest accuracy of 98.4% and a faster average classification time per image of 0.07 seconds. On the other hand, the first approach, the machine learning method, achieved slightly lower classification accuracy than the deep learning models but had the shortest classification time per image of 0.04 seconds.

Finally, we compared our results with similar research studies and found out that our proposed approaches compete well with these studies.

Keywords: Growth stage estimation, Precision Agriculture, Machine learning, Deep learning, Gabor filter, Morphological operations, Support Vector Machines.

TABLE OF CONTENTS

LIST OF TABLES

Tables

LIST OF FIGURES

Figures

List of Acronyms

ANN Artificial neural network

CM colour moments

CNN Convolutional neural network

EASA Environmental adaptive segmentation algorithm

ExG Excess Green Index

ExGR Excess Green minus Excess Red Index

ExR Excess Red Index

FN False Negative

FP False Positive

GLCM Gray-Level Co-Occurrence

GPU Graphics processing unit

GST Generalized Search Tree

HIS Hue-Saturation-Intensity

HSV Hue-Saturation-Value

IDE integrated development environment

IoT Internet of Things

LBP Local Binary Pattern

MSBPNN Mean-shift algorithm with Back Propagation Neural Network

NDVI Normalised Difference vegetative Index

PCA Principal Component Analysis

PNN Probabilistic Neural Networks

PSOMM Particle Swarm Optimisation clustering and Morphology Modelling

RBF radial basis function

RBPNN Radial Based Probabilistic Neural Network

ReLU rectified linear unit

RESNET residual neural network

RGB Red Green Blue

SCED Simple Canny Edge Detection

SGG stochastic gradient descent

SIFT Scale Invariant Feature Transform

SOM Self-organising maps

SVM support vector machines

TN True Negative

TP True Positive

VGG Visual Geometry Group

XGBoost eXtreme Gradient Boosting

Chapter 1

Introduction

This chapter summarises the study. The first part, 1.1, introduces Machine learning, deep learning, convolutional neural networks (CNN), and transfer learning. The second part, 1.2, is the motivation to justify why the study is essential. Section 1.3 outlines the research questions, the main aim, and objectives. Next, we outline the method to address the research questions in Section 1.4. Finally, the last section, 1.5, summarises the whole dissertation.

1.1 Background

Hydroponic systems are known for their efficiency in producing high yields with less use of resources and space [1]. Hydroponic systems placed indoors make data collection possible, which is not as feasible in traditional farming. Researchers have developed several approaches to achieve precision agriculture in hydroponics. One approach includes employing the internet of things (IoT) concept in hydroponic farms for remote controlling and monitoring [2]. Machine learning approaches have been used to guide

and coordinate the control system of these farms [3], [4]. Deploying a camera to monitor plants visually is more beneficial as it returns the plant growth and health status, which is not possible with just sensors. However, visual monitoring is not a simple task, and there is a lack of a straightforward computer vision method for real-time plant monitoring from image acquisition to image processing [5]. The existing computer vision monitoring approaches have shortcomings that cannot be ignored. For example, one method used the percentage green pixel method to measure the plant weight [5]. This method suffers where there is inconsistent lighting [6]. However, we can apply computer vision in hydroponics by combining feature extraction techniques, morphological operators, and machine learning classification techniques to build a robust plant growth classification and monitoring model. By doing so, plant monitoring through vision is made possible regardless of the camera hardware being used, the light nature, or the light level the camera is receiving.

This work will develop and compare three approaches to designing a plant growth estimator for hydroponics. The first one is the machine learning approach, where the feature extraction method is manually constructed using Gabor filters, Sobel filters, morphological operators. Then, machine learning algorithms such as support vector machines and XGBoost build the classifier. The second approach involves using pre-trained deep learning models such as VGG-16, VGG-19, and ResNet-50, through the transfer learning technique [7]. The third approach combines a pre-trained deep learning model with machine learning classification algorithms. Finally, the performances of the developed models are evaluated by carrying out side-by-side comparison experiments and examining performance metrics such as classification accuracy, confusion matrices, F1-score, precision and recall.

1.2 Problem statement

Growth estimation is key in achieving precision agriculture [8]. The plant growth stage shows the plant development stage [9]. Knowing the plant growth stage is crucial in deciding the most appropriate amount and type of nutrients that should apply to plants. As a result, resources such as water and nutrients are saved and running costs are reduced. Knowing the plant growth stage is also important in predicting yield [10]. Therefore, to achieve precision farming, growth stage assessment must be frequent. The most cultivated plants of indoor agriculture are broadleaf crops. These plants are very sensitive; they require the correct type of nutrients and proper environmental conditions to be supplied to achieve the best quality.

At present, growth stage assessment is done manually using eye-based scales such as the BBCH-scale (leafy vegetables forming heads) [11]. However, manual assessment of the growth stage is laborious and time-consuming, especially when a large farm needs to be attended to. Therefore, it is worth considering automating growth stage classification in broadleaf plant farming to reduce misclassification.

Computer vision as a way of monitoring plants can automate the growth stage estimation process from plant images captured in real-time using any ordinary camera feed hardware positioned above the hydroponics plants. However, broadleaf plants at different growth stages still pose similar morphological features, making it difficult to distinguish the growth stages. Therefore, our goal is to design a novel classification method that investigates other aspects of plant leaf than the shape to identify and distinguish growth stages.

1.3 Project scope

From the above problem statement, we limit this project's scope to the growth stage classification of broadleaf plants.

- **Dataset**: A publicly available dataset as "bccr-segset" [12] contains 30 000 images of three different plant species, is used to train and test the developed method.

- **Feature extraction technique**: Gabor filters are considered to design a feature extraction method manually.

- **Classification algorithm**: We will investigate the best machine learning classification algorithm between Support vector machines and XGBoost are considered the classification algorithm.

- **Deep learning models**: We focus on three pre-trained deep learning models, VGG-16, VGG-19, and ResNet-50, that have attained state-of-the-art results in many image classification competitions.

Our purpose is to investigate the best approach to develop a plant growth estimator between traditional machine learning, deep learning and a third approach that uses a deep neural network model as a feature extractor and then combines it with a machine learning classification algorithm. It is our goal to design a state-of-the-art plant growth stage model.

Note: Throughout this thesis, although deep learning is considered a subset of machine learning, we followed a convention that distinguishes between deep learning techniques and other traditional machine learning techniques when referring to “deep learning” and “machine learning”.

1.4 Research Questions

This study focuses on answering the following research questions.

1. How can a computer vision plant growth estimator model be developed using the traditional machine learning method?
2. To what extent is the algorithm of combining Gabor filters, morphological operators, and RBF kernel SVM classifiers effective in classifying growth stages?
3. How can one design a plant growth estimator by combining deep learning and machine learning approaches?
4. Do the results obtained from this study compete with those obtained in similar studies?

1.5 Research Objectives

To answer the research questions mentioned above, we address the following objectives:

- Do a literature survey of the existing feature extraction methods.
- To develop a model to estimate four plant growth stages (stage 1, stage 2, stage 3, and stage 4) using the traditional machine learning approach of handcrafting the feature extraction part and training the classification algorithms using the extracted features.
- To develop the growth stage estimation model using the deep learning approach.

- To develop a growth estimation model with a feature extractor based on deep learning techniques and a classifier based on traditional machine learning algorithms.
- Compare and analyse the three approaches, machine learning, deep learning, and the fused model (deep learning as a feature extractor and machine learning as a classifier) based on performance metrics, which are; the overall classification accuracy, confusion matrices, F1-score, recall, precision, and execution time.
- Evaluate and discuss the results obtained from the two approaches, machine learning and the deep learning approach.

1.6 Research methodology

This study is a quantitative study that is based on empirical investigations. Therefore, practical experiments form a crucial part of this work. We did the comparison analysis of these experiments on one IDE: Spyder IDE on a Windows system. We took the following steps to fulfil the primary aim of the research:

- Literature study:

  We conducted an extensive literature study to understand computer vision techniques, feature extractions methods, image processing methods, machine learning algorithms, and classification ability. In addition, we reviewed recent research studies to benchmark the method we are developing. A careful evaluation of the feature extraction techniques such as LBP [13], PCA, t-SNE and Gabor filters [14] was done to select the best feature extraction method which solves our problem.
- Experimental development:

  - Prepare data by grouping the images into folders representing their respective growth stages.
  - Split the images into training and testing sets (90% for training and 10% for testing).
  - Set up the Spyder environments.
  - Test the Gabor feature extraction filter on a few images to select the best parameters to set up the Gabor function.
  - Fine-tune the deep learning models.
  - Search for the optimal XGBoost and SVM kernel hyperparameters.
- Evaluate and discuss the experimental results.
  - We compared the results of each experiment and discussed the developed model's performance compared to the deep learning models in terms of classification accuracy, confusion matrices, F-1 score, precision, recall, training and testing time.

1.7 Dissertation overview

This dissertation aims to design an efficient and economic plant growth estimator based on machine learning. A significant part of this dissertation comprises empirical experiments and investigations that result in designing a high-performing plant growth estimator. We structure the dissertation as follows:

- Chapter 2 presents background information from the reviewed relevant literature regarding machine learning and computer vision in agriculture for plant classification. We also discussed deep learning approaches in agriculture for computer vision.
- Chapter 3 discusses the dataset and the preprocessing done to it.

- Chapter 4 illustrates the research method followed to achieve the Machine learning models.
- Chapter 5 presents the Design specification and the experimental setup used to develop the deep learning models. In this chapter, a third method of using a deep learning model as a feature extractor and a machine learning model for feature extraction is proposed.
- Chapter 6 presents the implementation of all the methods.
- Chapter 7 presents the experiments' results, and performance analysis of all the models is done using metrics, such as classification accuracy, F-1 score, confusion matrices, and precision. We also evaluated the training and testing time of these models.
- Chapter 8 concludes the dissertation with a brief discussion of the key findings, implications, and recommendations for future work.

Chapter 2

Background and Literature survey

This chapter presents the background of the study with information obtained from relevant literature. It also introduces concepts that are key to the study. Similar studies were investigated, and we identified areas that require further improvement.

2.1 Introduction

A detailed description of the application domain covered in this dissertation, together with the motivation for the need for developing technology to enhance crop management, particularly plant growth stage estimation, is presented in this chapter. We motivate the use of computer vision and image processing to enhance plant monitoring. We explain key concepts essential to the study. We also review the state-of-the-art works related to our problem in this section. These include the convolutional neural networks classification models, feature extraction methods, image segmentation, and classification algorithms.

The anatomy and taxonomy of plant leaves are also described in this chapter since they are the foundations for growth stage identification in this dissertation. Leaves have been

used for plant classification by several researchers. Shape, colour, and vein structure are among the most used plant leaf features. Many techniques, including sophisticated processes and calculations, have been used to extract these feature characteristics. The features are input into a classification algorithm such as the XGBoost and SVMs. One work is by Kulkarni et al. in [15]. In this work, they proposed an approach that uses texture, venation, shape, and colour to identify and recognise plants. The radial-basis-probabilistic-neural network (RBPNN) was used as the classification algorithm. The methods achieved a classification accuracy of 93.82% on a publicly available dataset, the Flavia dataset [149]. Kadir et al. [16] proposed a similar method, which employs texture, colour, shape, and venation of plant leaves to detect and classify plants. RBPNN was also used as the classifier, and classification accuracy of 93.75% was achieved. Therefore, excellent plant classification can be achieved by using plant leaf features.

2.2 Leaf anatomy

The fundamental components of plant leaves are the blade, petiole, and stipules. The epidermis, mesophyll, and vascular tissues are the three major tissues present in leaves. A tissue is made up of layers of cells. The cuticle is a thin waxy coating covering the leaf's outer surface. The primary purpose of the cuticle is to prevent water loss from leaves. Underneath the cuticle layer, there is the epidermis layer. The phloem and the xylem nerve tissues are situated inside the leaf veins. Veins extend all over the plant, from the roots to the leaves. Sheet cells make up the external layer of the vein, hovering through the xylem and phloem. Nutrients in plants move around along the phloem tissue, while the xylem moves water [16]. A visual presentation of the leaf anatomy and the fundamental components present in a leaf is given in Figure 2.0 and Figure 2.1 below.

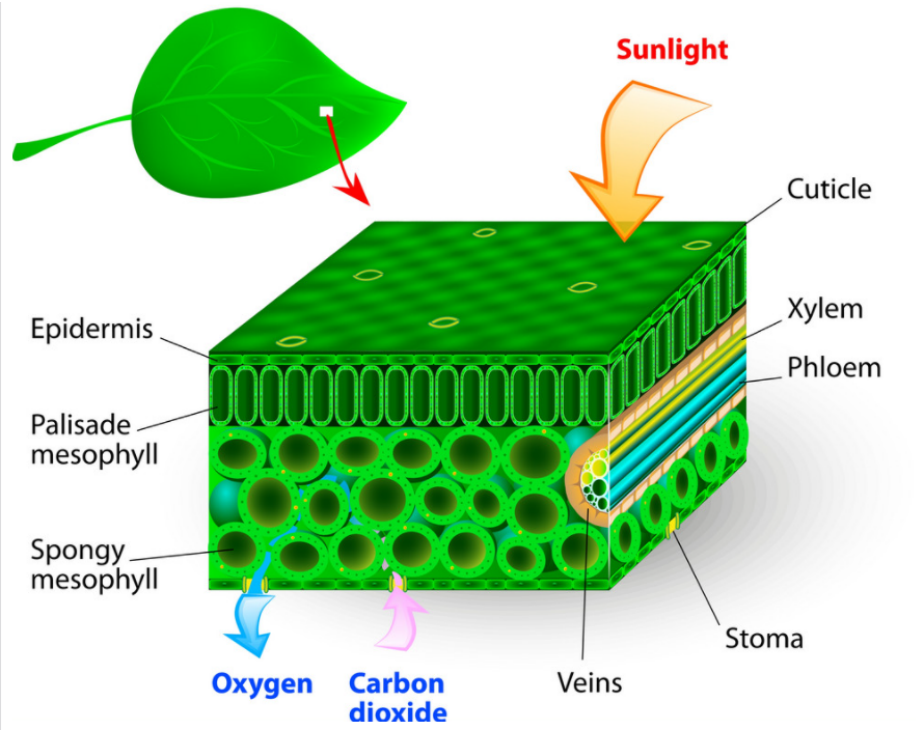

Figure 2.0 Leaf internal structure [17].

Another layer found inside the leaf structure is the mesophyll layer. This layer is divided into two layers: the palisade layer and the spongy layer. Palisade cells are segmented more and are found directly below the epidermis. In contrast, supple cells are more closely packed and are found in-between the lower epidermis and the palisade layers. Air gaps exist between the spongy cells that are important for gas exchange. Mesophyll cells have chloroplasts packed against them, where photosynthesis takes place. The epidermis layer lines the end portion of the leaf. The leaf also has stomata, which are tiny holes within the epidermis. The stomata are surrounded by guard cells shaped like two measuring hands. The stomata are open or close in response to changes in internal water weight. Figure 2.1 presents the plant's leaf external structure.

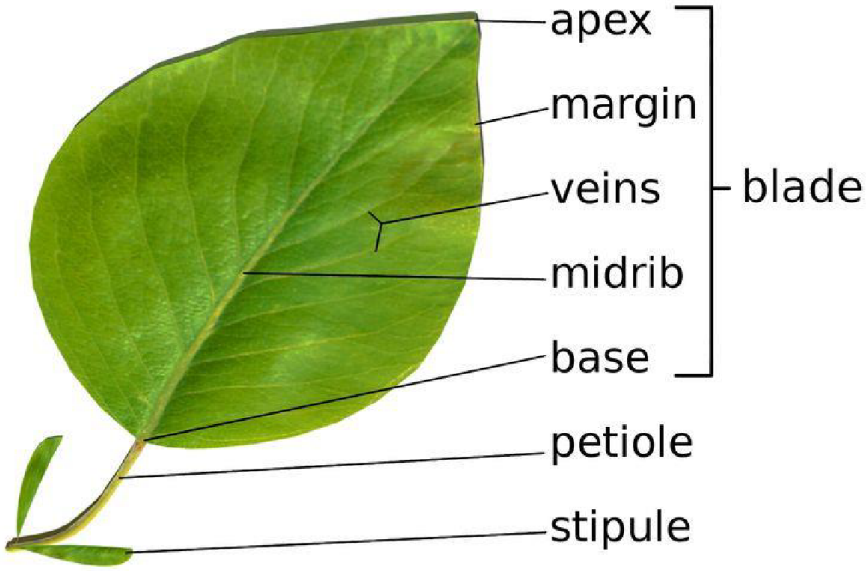

Figure 1.1. Leaf external structure [16].

2.3 Plant Features that can be used for growth estimation.

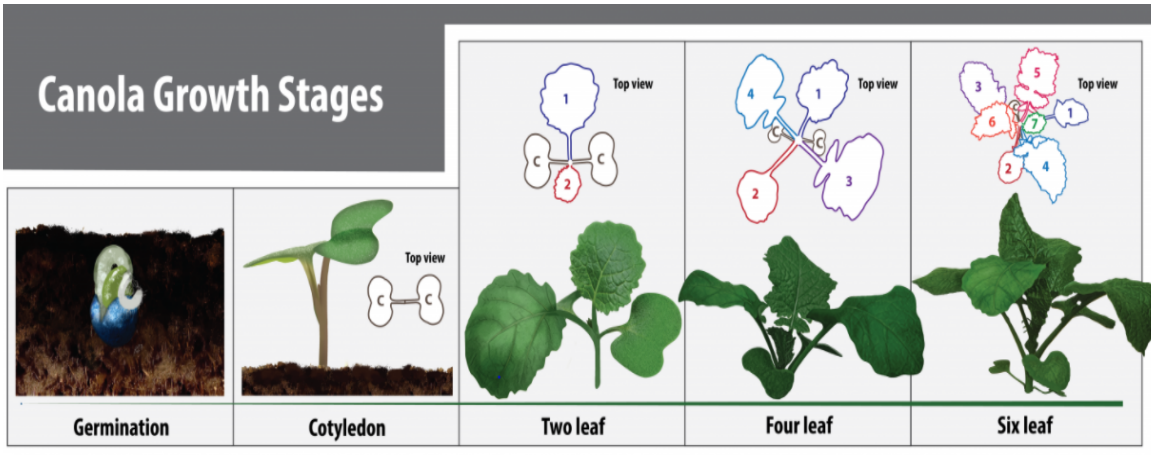

Figure 2.2 Canola leaf visualisation at different growth stages [18].

Plant taxonomy is a method of classifying plants according to their leaf characteristics. For example, Figure 2.3 shows a canola plant at different growth stages. From the figure, it is possible to see leaf characteristics of the same plant at various growth stages, which can bring about classification. This is in the form of venation, shape, texture, contours, size, and number of leaves. As a result, plant leaves can be classified into several classes based on specific characteristics, making growth stage classification possible. The site in [19] is an example of a tool designed to classify leaves with respect to the petiole, edge, blade, and veins. These features are discussed in detail as some of the most useful features for classification.

2.3.1 Petiole recognition for classification

Figure 2.3. Petiole identification [19].

A petiole is a part that joins the leaf blade to the stem. It varies in length depending on the plant. The above figure (Figure 2.4) is an example of how leaves can be categorised based on their petiole.

2.3.2 Classification using the blade

The blade of simple leaves extends up to the apex. In some situations where partitions exist, the blades will not extend to the midrib. Compound leaves, on the contrary, have a divided blade with divisions reaching to the midrib. On rare occasions, one or more of these divisions may resemble a single leaf. These are referred to as leaflets. Figure 2.5. below is how classification can be done according to the blade.

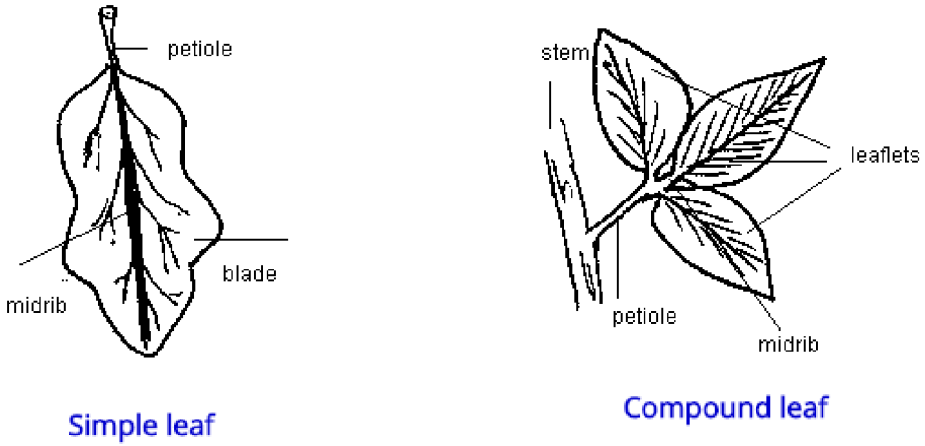

Figure 2.4. Classification by the blade [19].

2.3.3 Classification according to the edge

Classification can be made possible by looking at the plant edge. Figure 2.6 shows leaves with different characteristics at their edges. The entire leaf has a smooth border. Sinuate leaves are those that have some small curvatures that appear to be waves, serrate leaves are those that have a saw-like bend to them, and lobed leaves have divisions that often do not reach the midpoint of the blade.

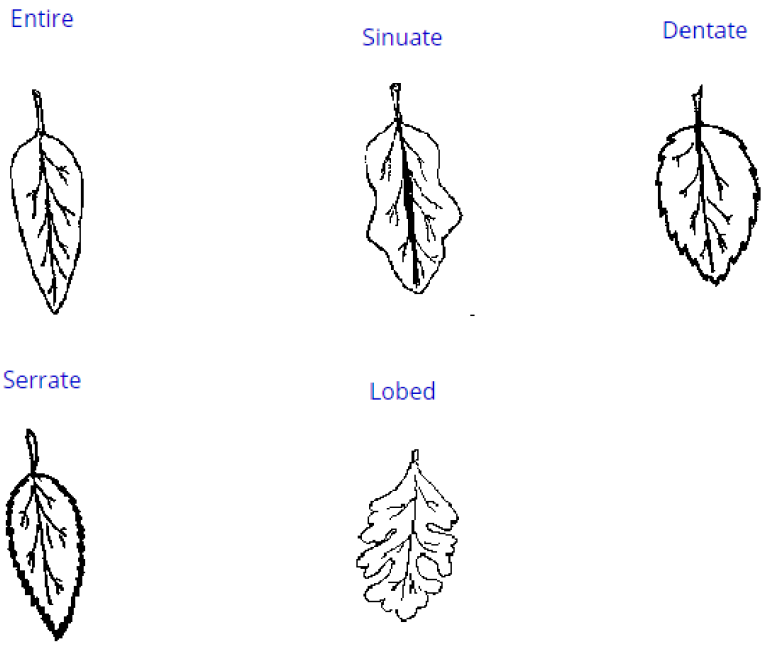

Figure 2.5. Classification by edge [19].

2.3.4 Classification according to the shape of the blade

Linear leaves are the leaves that are strip-shaped. Cordate leaves are the leaf type that has a heart shape. Some leaves have an egg shape, and these are called ovate leaves. The base of their ovate counterparts is broader. There are the leaves that exhibit the three-partition style. Hastate leaves have leaflets that have a broader base and sharp edges. The length of these is multiple times the breadth of their width. There are lanceolate leaves that are in the shape of a spear. The base of these leaves gradually widens while the tip becomes narrower. Acicular leaves are another type of leaves with a needle-like shape, which is longer and less broad and have a pointed tip [19]. Various leaf shapes which can bring about classification are shown in Figure 2.7.

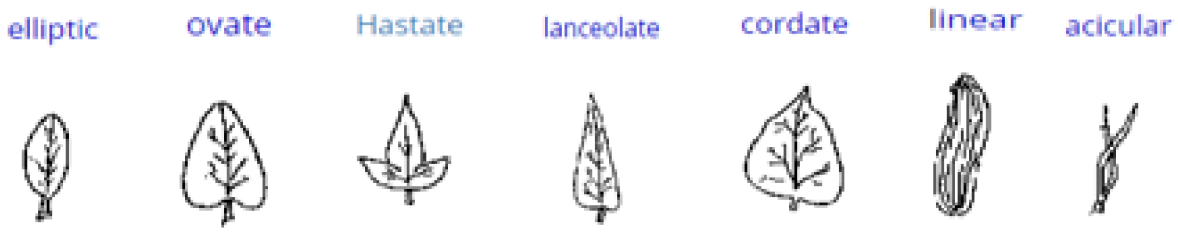

Figure 2.6. Classification according to blade shape[19].

2.3.5 Classification according to veins

The anatomy of the venation is taken into account when classifying according to veins; some leaves have parallel veins, others have a primary nerve in the centre with branches radiating from it, and others have divergent nerves. Parallel-veined leaves have veins that run parallel to one another. Pinnate leaves have a mid-main nerve; palmate leaves have no such thing. Figure 2.8 depicts the classification based on veins.

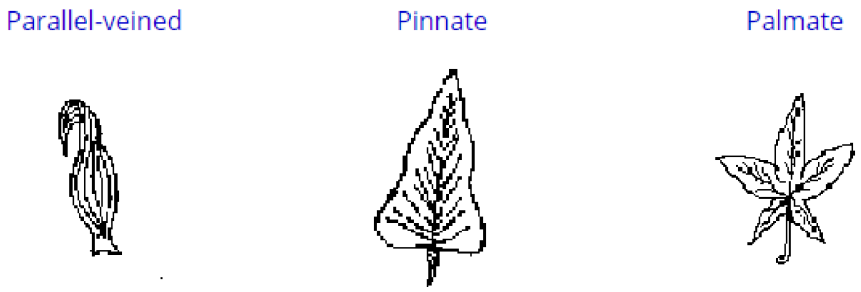

Figure 2.7. Classification based on veins [19].

2.4 How plant features were used for leaf analysis in previous works

In recent research, various cutting-edge approaches for classifying plants based on their leaves have been used. A variety of leaves have been classified based on physical and genetic characteristics. However, the diversity of these traits among plants has posed significant problems, making classification difficult. As a result, scientific approaches were used to automate plant classification based on leaves. Researchers in [20] conducted a comprehensive evaluation of 120 peer-reviewed articles on automatic plant classification published between 2005 and 2015. The study's findings revealed that leaf shape is essential in plant classification. Over half of the studies classified plants based on their leaf shape. Another 13 studies based their conclusions on the shape of flowers. In 24 of the studies, leaf texture was used. Flower analysis was associated with colour classification in 9 of the studies. Researchers used colour to classify leaves in 5 of the studies. Plant organs such as venation were used in 16 studies mentioned in this review work and 8 studies used the leaf margins.

2.4.1 Identification based on the leaf shape

The shape is used by human eyesight to distinguish objects most of the time. As a result, the shape is also important in computer vision. An object's edges are crucial to determining its structure in shape measurement. In real-world situations, variables such as translation, reflection, rotation, and scale can change the appearance of an object. Therefore, the relevant shape characteristic should stay consistent in the presence of these changes. An automated approach was proposed in [21] to classify and recognise leaves based on their shape. Here, two methods were employed; one used a prototype termed invariant-moments, while the other used one centroid-radii. The two were compared

based on their classification accuracy percentages. The whole approach has proven to be highly useful for quickly and accurately classifying plants using leaves. The method's accuracy was comparable to similar studies in this field. This approach has a distinct benefit since it employs a low-complexity data modelling scheme, with feature vectors with dimensionalities generally around 40.

In another study [22], the researchers demonstrated three plant classification techniques based on leaf shape and three models – the Fourier Moment Technique, Support Vector Machine with Binary Decision Tree, and Probabilistic Neural Network – were used to perform multiclass classification. A dataset of 1600 leaf forms from 32 different classes was employed, with classes made up of 50 examples of the same type of leaf. After a series of tests with various algorithms, it was determined that the Support Vector Machine with Binary Decision Tree outperformed the other two models.

Analysing leaves based on their shape has some limitations which cannot be ignored. One of the problematic attributes is that several leaves may have similar margin features but distinct shapes, or vice versa. There is often a lot of morphological variation between leaves of different species, and there is also a lot of morphological variation between leaves of the same species. The findings show that Morphological Shape Descriptors are excessively rearranged to distinguish leaves beyond those with significant differences and they are frequently combined with other descriptors like texture and venation to generate excellent outcomes [22].

2.4.2 Identification based on leaf colour

Colour is an essential component of photographs. Colour characteristics are defined inside a shading space. Researchers have utilized several colour spaces, including hue-max-min-diff (HMMD), hue-saturation-intensity (HSI), Hue-saturation-value (HSV), and red-green-blue (RGB). Several common colour descriptions have been proposed in computer vision for image classification [23]. Colour histograms, colour moments, and

colour correlograms are examples of colour descriptors that are used to extract features from the images. Colour moments were used to classify leaves by Kulkarni et al. [15]. The study showed that colour moments are fundamental approaches that may be used to successfully distinguish images based on their colour highlights. In [20], colour moments proved to distinguish images effectively. When these colour traits were combined with other features such as texture and venation, a classification accuracy of 93.82% was attained. According to work by Waldchen [20] in 2016, colour moments (CM) are the most studied colour descriptors, and the results are high and trustworthy. However, there are significant drawbacks to using colour as a classification feature. Because of the varying concentration, an obscurity of the brightness originating from distinct edges and light variations are special tests for colour-based research. Variations in enlightenment may result in shadowing and changes in light concentration. Images are shot under varied settings, and the difference in illumination can significantly impact the final result [24]. A study found that relying simply on colour data alone to identify flowers is insufficient [25]. Based on the above findings, colour alone as a feature is insufficient and must be supplemented with additional characteristics.

2.4.3 Identification based on leaf texture

The texture of an object is defined as the surface features and appearance of an object [26]. Smooth or rough, coarse or fine, are some examples of texture. The primary goal of texture analysis is to express a true sense of texture digitally and enable the programmatic modification of texture data for models based on computer vision. Leaf texture features are acquired through this method. Texture analysis methods are divided into four categories: model-based approaches, statistical approaches, signal processing methods, and approaches related to structural. Fourier Descriptors and Gabor Filters are two examples of methods to analyse texture. In addition, researchers have employed texture analysis approaches in different ways to extract textures.

In one study done by Cope et al. [27], leaf texture was used to classify plants. This work calculated different plant leaf textures using the Jeffrey-divergence measure of corresponding distributions. This method was compared to traditional texture analysis methods on many datasets: the Gabor filter and the Fourier descriptors. The developed algorithm attained the highest classification score. This work illustrated the effectiveness of texture as a feature for classification.

Although leaf texture has proven to be very useful in automatic plant classification, it must be used with other factors to obtain the best results. On its own, it will not consistently provide effective outcomes. The texture is more precisely defined by the sense of touch and the fact that touch cannot be defined digitally; texture alone as a feature extraction characteristic is limited. Regardless of several texture feature descriptors that have been introduced in agriculture, such as the Gabor filter [27] and Gray level co-occurrence matrix (GLCM) [28], texture extraction is still a problem as the leaf surfaces are not an easily perceptible structure. In addition, environmental factors can affect leaf appearance, which can compromise the leaf segmentation process.

2.4.4 Identification based on leaf vein structure

Veins provide structure to leaves and transport various elements like water and nutrients. Leaf veins come in multiple shapes and sizes, including parallel, palmate, and pinnate, as previously discussed in leaf taxonomy. Because the vein structure of a leaf differs between plant types, using vein structure to characterise leaves is an effective approach. Veins are commonly visible because of their great intricacy compared to what has been left of the leaf's sharp edge.

Sixteen studies focusing on vein structure to classify leaves were reviewed in [20]. Four of these studies focused solely on venation, neglecting to evaluate other leaf features such

as texture. The other twelve studies looked at vein structure in conjunction with leaf shape, while two of them looked at vein structure in conjunction with three features: colour, shape, and leaf texture. Kulkarni et al.[15] is another study that looked at venation features where morphological techniques were applied on grayscale images to extract veins using three different ways shown below:

$$F1 = \frac{P1}{P} \tag{2.1}$$

$$F2 = \frac{P2}{P} \tag{2.2}$$

$$F3 = \frac{P3}{P} \tag{2.3}$$

Where:

- F1, F2, and F3 are the vein features,
- P1, P2 and P3 are the total vein pixels, and
- P signifies the total leaf pixel number.

Venation, like the other features described above, has drawbacks. Wang et al. in [30] extracted shape and vein using Simple Canny Edge Detection (SCED) and Scale Invariant Feature Transform (SIFT). They realised that vein patterns aren't always effective for SCED categorisation. Their conclusion was reached due to output distortions during their experiments: the utilisation of these venation patterns in the context of shape resulted in inconsistent performance percentages. A study [31] claims that dissecting the vein structure of leaves is a challenging task. According to the researchers, this is due to the minimal discrepancy between the vein structure and the structure of the blade of leaves.

2.4.5 Identification based on leaf contours

In [32], the leaf shape or contour extraction approach, which depicts the lines between the centroid and leaf points in an image, was presented. First, a histogram is made to represent the leaf contour. Following that, a classifier is coupled to a contour feature extraction model to determine the layout and question the leaf's classification accuracy. A 92.7% success rate was attained, demonstrating this method's limits. Other features used by researchers include the leaf margin. Like the others we've looked at, these features have some flaws.

2.5 Summary

The above section demonstrates that more than one feature should be extracted to achieve high plant classification accuracy. Also, feature extraction and choosing the most appropriate and effective features for a classification task are neither simple nor straightforward. Therefore, an important question arises, "is it possible to automate and concatenate these three processes (feature extraction, feature selection, and classification) in a single model?" A means to address this issue is seen through machine learning and deep learning. Traditional machine learning focuses on handcrafting, the feature extraction method targeting key features. Machine learning tends to be the best choice when the features to extract are known as only those features are targeted. As a result, the need for expensive computational systems is minimised and execution is speeded up. In deep learning, feature extraction is done automatically. Data is transferred from the first layer to the last layer as features are being extracted. The neural networks decide which features to extract. One drawback of this approach is that it requires a lot of computation as the networks are very deep. Also, as the image data is passed through each layer in deep networks, the classification process tends to be slower unless expensive hardwire such as GPUs are employed. Therefore, in this study, we have decided to develop and test

a computer vision model using the traditional machine learning approach against a model based on deep learning. The following section presents a review of the works done in this field. Moreover, a detailed explanation of the three approaches used in this work, handcrafting a feature extraction method using Gabor filters and morphological operations for machine learning, deep learning approach through transfer learning, and the use of a deep neural network as a feature extractor then concatenate it with a traditional machine learning classification algorithm is given in Chapter 4 and 5.

2.6 Computer vision in agriculture

The application of computer vision in agriculture is not new. Little research and publication have been made on the application of computer vision in agriculture. This is because of its complexity resulting in unsatisfactory accuracy [56]. Computer vision was utilised to grade fruits and vegetables, and in defect detection [13], [14], [33]–[35]. A potential success was shown in robotic weed control [36] using computer vision, with a few challenges discussed later in this chapter. Researchers [19]–[21] applied computer vision to identify and distinguish plants from weeds. Several image processing techniques were developed to guide the computer vision method in different fields, such as indoor and outdoor agriculture. Image segmentation is also a part of image processing, which comprises image preprocessing and object detection. Therefore, we present an overview of some steps in image processing for plant leaf detection. One problem in the image processing of plant images is illumination variation. This research aims to circumvent this problem.

2.6.1 Image preprocessing

Image preprocessing involves two fundamental processes, noise removal processes and enhancement processes. Image enhancement aims at highlighting the most important information while suppressing unnecessary information, thus improving the interpretability of an image. Image enhancement's ability has been of great importance in many applications, such as fingerprint identification [37], remote sensing [38], medical imagery [39], and plant disease identification [40]. Some of the important image enhancement processes are thresholding, morphological operations of opening and closing. All these processes aim at highlighting important details in the image. Morphological transformations have been successful in enhancing medical images [39]. We can convert coloured images to grayscale to address the illumination invariance issue. In grayscale images, variation in the lighting of images has less effect. Researchers in [41] converted all their images to grayscale images to reduce the impact of illumination differences in their dataset. Thresholding transformation is a technique that can perform image segmentation. Thresholding has been significant in remote sensing in removing noise and enhancing the images [42].

2.6.2 Plant segmentation

Image segmentation is the initial step in image classification problems. Through image segmentation, we can divide an image into various uniform regions. Researchers in [43] have viewed image segmentation as a bridge between image processing techniques, such as image enhancement and noise reduction, and vision processes, such as object detection. In [44], image segmentation was used to separate the plant from the background. The researchers investigated if image segmentation can have any effect on the results. An unsegmented dataset and a segmented dataset were used to train a convolutional neural network for classification. The results showed better performance

and higher accuracy from the model trained using the segmented data. The segmentation process must be excellent to construct a high-performing model for classifying plant growth stages since it has significant impacts.

Several studies have employed the percentage of green pixels method to segment plants from images [45]–[47]. This method works well when the green channel of the RGB channels contains more information. However, image segmentation techniques based on the colour index are not efficient in segmenting plants from the background in real life, as they require precise thresholding adjustments. Also, in case of abnormal lighting conditions such as low lighting and bright lighting, which can obscure the green colouration, these methods perform poorly. Researchers suggested several methods to solve the mentioned problems, such as the Environmentally Adaptive Segmentation Algorithm (EASA) [41], Mean-shift algorithm with Back Propagation Neural Network (MS-BPNN) [48] Particle Swarm Optimisation clustering, and Morphology Modelling (PSO-MM) [49]. A detailed summary of the different methods used for plant segmentation can be found in [50]. Regardless of these methods being very efficient under different lighting conditions, we should not ignore their computational demands. High computational requirements make them less fit for real-time applications. It is worth mentioning that these methods are only applied to segment plants from the background but not in distinguishing plant growth stages, which is the primary aim of this study.

2.7 Plant Detection

Researchers have employed image processing techniques based on colour, shape, and texture to identify plant growth status. They have achieved satisfactory performance on distinguishing growth stages of cereal crops [51]–[55]. The colour feature has been used to separate plants from the background [56], [57]. The colour-based approaches are efficient when the background is uniform and simple. When the background is complex and non-uniform, misclassification increases with these methods. Light invariance affects

colour-based approaches. Researchers in [58] tried to correct the lighting invariance in colour-based feature extraction methods by developing a method called the "environmentally adaptive segmentation algorithm (EASA)". This method is robust. The primary concern with this method was its requirement for a vast amount of training data and much human interaction to achieve substantial adaptation to the environment. [51] developed a method that combines colour features and morphological features. In this method, the researchers used the hue-intensity look-up table and affinity propagation algorithm to develop a method that does not require much human interaction and a large dataset.

Instead of using traditional and manual feature extraction methods, researchers have adopted convolutional neural networks (CNN) to do automatic feature extraction. In [59], researchers have used CNN to estimate the growth stages of weeds. CNN extracts the morphological features of the weeds, such as size and leaf number. This method suffers when there are overlapping plants, infection, and nutrient deficiency. Growth stage classification through leaf counting depends mostly on image segmentation [44]. If the background is not separated accurately, the classification accuracy of these methods becomes very low. When there is plant overlapping or leaves overlapping, image segmentation becomes difficult. A method for segmenting images by converting RGB images to log-polar space has been proposed in [60]. This method extracts plant images from the background better. Researchers in this work employed a vector regression technique to count plant leaves. The main limitation of this method is that the segmentation process must be applied to both the training and testing data. This slows down the method and makes it less fit for automatic systems.

2.8 Plant growth stage identification

Several researchers have developed image processing techniques to identify and classify growth stages. Colour features, texture, shape, size, and the number of leaves are used in

these methods to enhance classification. Different colour spaces, such as RGB, HSV, CIELab, and YCbCr, are used to quantitatively represent the spectrum of colours. Many applications portray colours as RGB components, which are a mixture of different intensities of red, green, and blue. The YCbCr colour space is another. The Y component represents brightness, the Cb component represents the blue-yellow spectrum, and the Cr component represents the red-green spectrum in this colour representation. Colours are represented in the HSV colour space by the letters H, S, and V, which stand for hue, saturation, and value. Hue is a numerical value ranging from 0 to 1.0 that depicts the colours that range from red to black to red. Saturation, on the other hand, is a numerical number ranging from 0 to 1.0 that represents the range of colours from unsaturated to completely saturated. Finally, the value denotes brightness, which ranges from 0 to 1.0. The CIE Lab colour space is formed from the tristimulus values of the CIE XYZ system. Luminosity 'L,' chromaticity 'a,' and chromaticity 'b' make up the CIE Lab colour space. 'L' is for the brightness layer, while 'a' stands for chromaticity measured along the red-green axis and 'b' stands for chromaticity measured along the blue-yellow axis. Pocholo et al. [61] compared some colour spaces (RGB, HSV, ClElab, and YCbCr) to see which colour space can better classify lettuce plants. They observed the ClElab as the best colour space for identifying plant development stages. Using the CiElab colour space to develop a plant growth stage classification algorithm helps analyse complex images, which other algorithms based on different colour spaces struggle with. Concepcion et al. [62] developed a colour-based method to classify lettuce growth stages. Instead of using standard pixels in this method, the researchers used super-pixels to represent the original images and then used multi-fold transformation to segment these plant pixels. A single super-pixel has more spatial information than conventional pixels; as a result, it gives a compact, deep, and almost perfect representation of the abstraction of the original image. To enhance the classification, they extracted morphological features such as the lettuce head perimeter, area, and the length of the dominant leaf. In classifying three stages, this algorithm got an accuracy of 88% on stage 1, 86% on stage 2, and 79% on stage 3. Including morphological features enhances classification. In addition, using super-pixels

reduces computational costs as a super-pixel is a single pixel representing a group of similar pixels. This results in a compact representation that contains all the significant image information. We can notice that this method requires much human annotation of the dataset to apply effectively. In [53], researchers classified rice growth stages using the NDVI image processing and CNN. An accuracy of 89.3% was obtained in classifying the six growth stages of paddy rice plants. This method is efficient for remote sensing, where images used are captured from a distance.

2.9 Dealing with occlusion in computer vision

Segmenting overlapping objects is a challenging task. Overlapping objects appear as a single object. In agriculture, computer vision can detect overlapping plants as a single plant. In other computer vision applications, such as medical, remote sensing and object detection, overlapping is an immense problem that causes high misclassification rates. Many research works published on image processing and computer vision ignored or overlooked overlapping. In [65], a recent review on occlusion detection methods mentioned that self-occlusion and inter occlusion are still problems in various areas. In that research work, Himanshu and others reported that when the same algorithm is used to track the occluded and occluder objects, it cannot handle occlusions well. The watershed segmentation algorithm [64] was employed in agriculture to handle inter-object occlusion by recognising an object and then distinguishing objects from each other. Although the watershed algorithm succeeded, over-segmentation was a significant problem [65]. This motivated some researchers to change and improve this algorithm to solve the occlusion problem. Lee et al. in [64] compared five modified watershed algorithms with the original algorithm on occluded tomato leaves. The performance of the two modified algorithms was better than the original algorithm. The researchers reported that the watershed algorithms are inefficient for long and thin occluded plants such as corn. In [51], the authors applied an improved watershed in the HSV colour space

to identify and segment ripe tomato fruits in complex environments where occlusion and some environmental constraints are challenging. Fair detection accuracy of 81% was obtained. An improved watershed algorithm was applied in the segmentation of cotton leaves. First, colour-based feature extraction was used to separate the green pixels, the cotton leaves, from the background and shadows. Then, they employed a canny edge detector to detect the leaf boundary lines and to detect discontinuities. This improved the watershed algorithm in segmenting occluded cotton leaves.

Researchers have also applied deep learning to handle the occlusion problem in agriculture and other areas. In [67], the YOLOv3 network was used to deal with overlapping and occlusion in detecting apples in environments that are not ideal. The authors achieved a commendable quick apple detection with this network, a detection time of 0.3 seconds at 3000 x 3000 resolution and overall detection accuracy of slightly above 80%. More emphasis in this work was put on improving the detecting speed. Liu et al. also applied the YOLOv3 model to recognise tomatoes in complex environments [68]. The algorithm introduced a dense architecture to optimise the YOLO model. As a result, they achieved tomato recognition accuracy of 94.58% in a complex environment.

In summary, over-segmentation is still a problem for the machine learning algorithms presented above. Over segmentation happens when the segmented objects being segmented from the background are further segmented into sub-objects. This can cause high misclassification rates. On the other hand, deep learning models produce excellent results, but they require extensive training samples to optimise the models.

2.10 Growth stage classification using machine learning and deep learning

Precision agriculture, which aims to maximise productivity while reducing losses, has been achieved in agriculture through machine learning techniques. However, in [35],

Hamuda et al., in their survey, reported that although machine vision has obtained some successes in agriculture, challenges are still faced when applied in real-world situations in the extraction and segmentation of plants. This is due to factors such as complex background, lighting invariance, occlusion, and overlapping.

Various methods have recently been proposed to classify plant growth stages. These include the use of supervised and unsupervised machine learning methods. In [69], unsupervised machine learning techniques, k-means, and self-organising maps (SOM) were applied to classify lettuce growth stages based on their morphological features. The model yielded an accuracy of 91%. The deep learning approach has also been used in computer vision applications in agriculture to classify the growth stages of chilli plants. In [70] Mask R-CNN models based on (ResNet-50, ResNet-101) were used. These models were trained on a small dataset consisting of 256 plant images. The Mask R-CNN ResNet-50 and Mask R-CNN ResNet-101 models attained an accuracy of 96% and 97%, respectively, in identifying the actual age of the chilli plants. In [55], SVM was used to classify cannabis crop growth stages of spectroscopy data obtained through the NIR spectroscopy technique. SVM was used to classify the growth stages of the potato crop in [51]. Their technique involved the application of the SVM model to the spectral and sensitivity data obtained from the potato plant. The SVM model obtained an accuracy of 100% on the training set and 97% on the testing set. In [72], [56], Rasti et al. used convolutional neural networks to classify two wheat and barley growth stages. Their work created a five convolutional layer network and trained from scratch alongside a pre-trained network (VGG-19) and SVM model. They found an accuracy of 91.1%, 99.7% and 63.6% was obtained by the 5-layer model, VGG-19 and SVM, respectively.

Machine vision has also been applied in crop and weed classification. For example, researchers in [20] applied transfer learning to segment weeds, and 93.9% accuracy was obtained. Sixteen plant species were also classified using transfer learning models in [73]. A comparison was made with a radial basis function (RBF) nonlinear support vector machines (SVM) kernel model concatenated with LBP and a General Search Tree (GIST). The transfer learning model obtained a classification accuracy of 97.4%, the

SVM with LBP operator model obtained 74.92%, and the SVM with GIST features obtained 83.88%.

CNN has been applied in detecting plant diseases, and satisfactory results have been achieved. An investigation was conducted to see if deep convolutional neural networks can detect plant diseases[29]. Their experiment was performed on a dataset comprising images of around 50.000 infected and healthy plants. The developed CNN model was trained and achieved an accuracy of 99%. Deep network architectures of AlexNet and GoogleNet were tested on images on different images (Grayscale, RGB, leaf segmented). Fine-tuning was done in these experiments. Hyperparameters were also tuned. With 80% of the RGB image dataset used as training samples and 20% as testing samples, GoogleNet yielded an accuracy of 99.2%.

CNN has gained more traction in computer vision problems in the above-reviewed research works. However, it can be observed that large datasets must be supplied for deep convolutional neural networks to achieve high accuracy. Furthermore, a deep convolutional neural network requires high computational power, which can be a challenge when the system is to be implemented in real-time situations, such as installations in drones to make real-time detections. Nevertheless, SVM models obtained fair accuracies, although most of these researchers used linear models where the data was nonlinear; an example is a work by researchers in [56].

2.11 Conclusion

This chapter has outlined various computer vision methods used in smart agriculture, focusing on plant identification, plant classification and plant growth estimation. Different computer vision techniques have been employed for plant segmentation, detection, and classification. We have also discussed approaches to deal with overlapping and occlusion with their limitations. The primary aim of this dissertation is to design a

plant growth estimator while addressing the problems mentioned in this chapter. The following chapter presents the dataset used in this study and discusses appropriate preprocessing and feature extraction methods.

Chapter 3

Dataset

In this chapter, we introduce the dataset used in this study. We also discuss the data source, the preprocessing steps done to the dataset by the creators of this dataset, and the processing steps taken to fit the dataset for this study.

3.1 Introduction

This chapter gives a detailed description of the dataset adopted in this study. It is important to note that, although computer vision has gained a lot of attraction in agriculture, there is a scarcity of datasets made publicly available [66]. This slows down progress in developing, evaluating and improving the existing computer vision algorithms in this field. To develop a plant growth estimation model, a dataset that consists of many plant images at all the crucial growth stages is important. A vast number of plant images is important at each growth stage to extract the key features that can be fed into a machine-learning algorithm to distinguish the growth stages of new plants. A dataset published online with the name "bccr-segset" [12] consists of 30 000 plant images of three plant species (canola, radish, and corn) taken at four growth stages. The number of images contained in this dataset is enough for all the approaches taken in this work to

develop a plant growth estimator. The dataset was primarily created to distinguish weeds and crops at four defined growth stages using the Local binary pattern (LBP) method [20]. Our problem is the classification of the growth stages of different plants. Therefore, instead of classifying plant species, we are classifying the growth stages of these different species. So, some preprocessing steps will be taken to the dataset to fit it to our goal and make automatic growth stage classification possible.

3.2 Data source

This dataset was collected at a facility at Edith Cowan University, Australia [12]. They installed the camera on a mobile trolly to capture the plant images where its optical focus lens faced the testbed that held the plants. The moving trolly holding the camera is set to a moving speed of 1 m/s. Images are captured in real-time. The camera was 980 mm above the ground and had a focal length of 9 mm. The size of each captured image is 228 x 228 pixels. The plants are held in pots using soil, making the background uniform. Two fluorescent tubes are used to illuminate the testbed uniformly. Several images of the individual plants were collected by running the camera multiple times on the testbed. Image variation was introduced in the images by rotating the camera equipment at different angles. Image variation is important as it prevents data leakage and overfitting when developing the machine learning model. Generally, image variation through taking images at different angles is the same as image augmentation in deep learning. It also increases the robustness of the model. Therefore, there is no need to perform data augmentation when developing our models, as this is done manually during data collection.

3.3 Image segmentation

Image segmentation is dividing an image into uniform areas. In plant detection, we base segmentation on separating the green pixels of the plant matter from the background pixels, which are the non-green pixels. Accurate segmentation improves the feature extraction process. In this dataset, a colour index-based approach, ExG-ExR (Excess Green minus Excess Red Indices), was used to extract the plant matter from the background. The ExG component extracts the plant matter, while the ExR removes the background and noise [67]. This approach has proven to be more useful in plant segmentation than the other approaches, such as the Otsu method [74]. Then after image segmentation, the images are converted to greyscale. The figures below, Figure 3.1, Figure 3.2, and Figure 3.3, illustrate image segmentation performed on a random image collected from the testbed. This process is performed on all the images in this dataset.

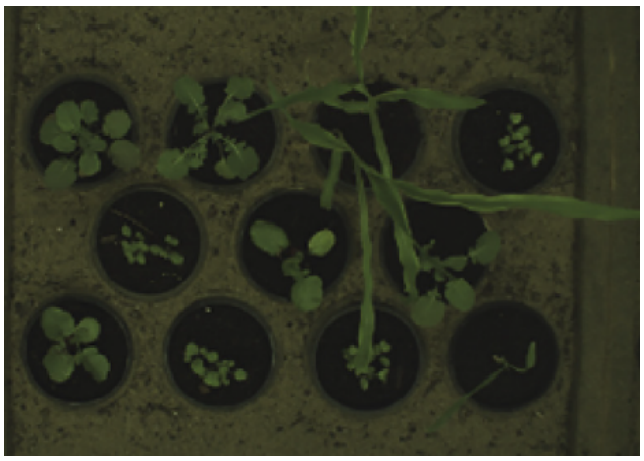

Figure 3.1. The original RGB image captured from the testbed before any segmentation [12].

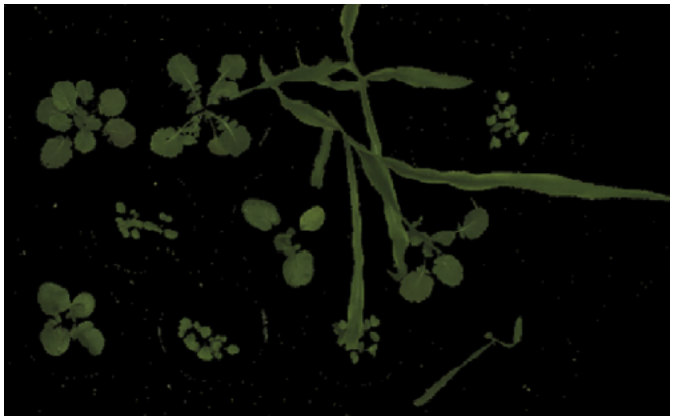

Figure 3.2. The segmented image with only the plant matter and no background [12].

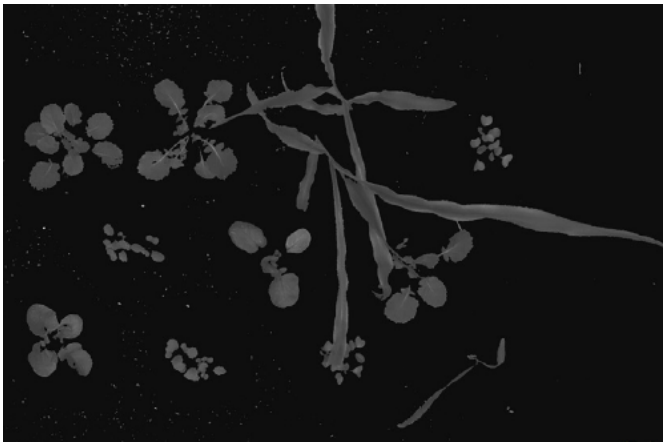

Figure 3.3. The segmented image is converted to greyscale [12].

3.4 Data labelling

A specialist labelled the images according to their species and their respective growth stage levels to have the correct growth stages and ground truth regarding each plant species. Since the images are captured using a camera mounted on a moving trolly, capturing partial plant images was possible. Also, manually labelling the images can cause a high error rate. Attempting to label the images manually can lead to biasing and class imbalance as it is hard to manually identify plants with not enough plant matter to

label them as background or plant. Therefore, the labelling of images according to their respective growth stages and plant species was made autonomous using the ground truth provided by the specialist. This involved the use of thresholding to remove images with little to non-plant matter at each growth stage. The images with very little to no plant matter were grouped and labelled as background. Thresholding of the leaf edge area was done to identify and label partial plants from some growth stages. By doing so, images with partial plants were not removed from the dataset. The thresholds used to label the plants are presented in Table 3.1. These are the images fed into our models to train them to identify and distinguish growth stages in real life. Automated labelling helps provide correct data to the model, which does not consist of confusion and bias.

Table 3.1: Canola and radish thresholds used as default to automate the labelling process of the dataset.

Threshold in (cm^2)	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**
Threshold (Inner, Edge) - Canola	(1.4, 3.3)	(3.0, 6.7)	(7.0, 10.0)	(8.0, 12.2)
Threshold (Inner, Edge) - Radish	(2.5, 4.0)	(3.2, 6.7)	(7.0, 10.0)	(8.0, 13.8)

3.5 Dataset partitioning

Now that the dataset has been preprocessed, our work was to group and rearrange the dataset from grouped according to plant species to grouped according to growth stages. We then created two datasets from the main "bccr-subset" dataset. A dataset consisting of each plant species was created, a canola and a radish dataset. We restricted the scope of this work to broadleaf plants grown under hydroponics. As a result, we left corn plant images in this study. The following figure presents images from the dataset at each growth stage.

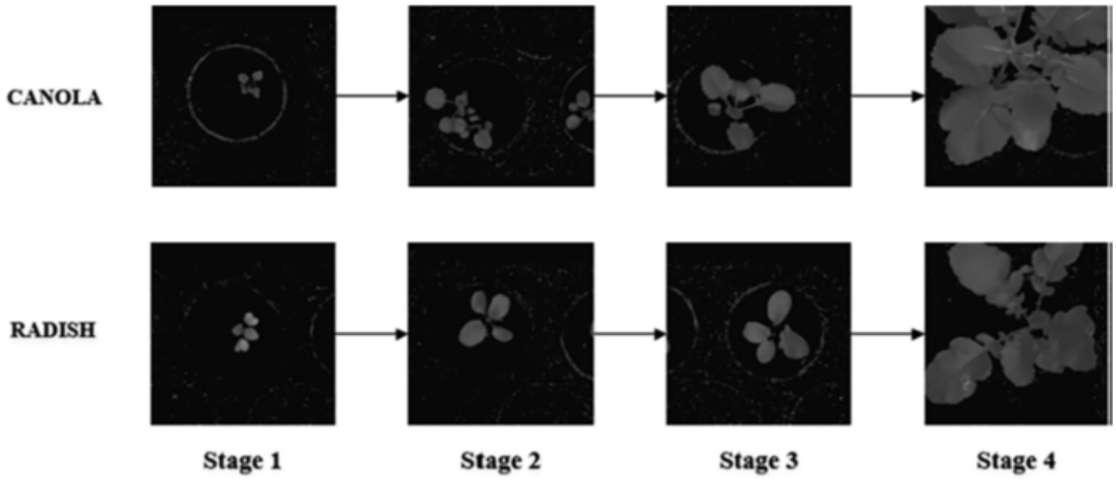

Figure 3.4. Random segmented and grayscale images of radish and canola at four growth stages (Stage 1, Stage 2, Stage 3, and Stage 4).

Table 3.2. The two datasets (canola and radish) and the respective number of samples in each class were created. For the canola dataset, plant images are distributed as follows (Background = 2387, Stage 1 = 527, Stage 2 = 360, Stage 3 = 1629, and Stage 4 = 475). In the radish dataset, the images are distributed as follows (Background = 799, Stage 1 = 782, Stage 2 = 780, Stage 3 = 822, and Stage 4 = 1436).

Canola dataset	**Radish dataset**
Background - 2387	Background - 799
stage 1 - 527	stage 1 - 782
stage 2 - 360	stage 2 - 780
stage 3 - 1629	stage 3 - 822
stage 4 - 475	stage 4 - 1436
Total = 5378	**Total = 4619**

3.6 Dataset visualisation

To visualise the structure and the correlation between each of the growth stage classes, we used principal component analysis (PCA) [75] and t-SNE [60] techniques. We

implemented this technique in python with the help of the t-SNE user guide [60]. PCA and t-SNE are dimensionality reduction techniques. We used t-SNE after PCA to further reduce the dimensionality.

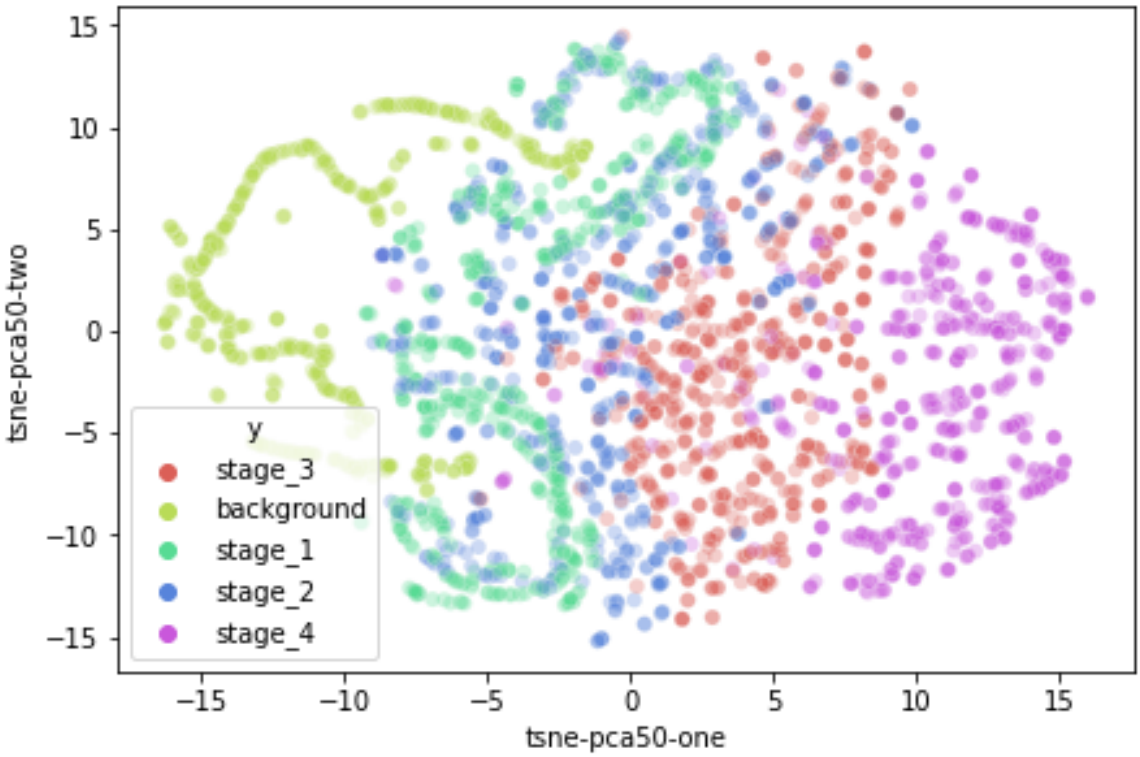

Figure 3.5. Visualisation of the canola dataset's five classes (background, stage 1, stage 2, stage 3, and stage 4).

Figure 3.5 is a two-dimensional representation of the canola dataset obtained using PCA and t-SNE. This before applying feature extraction. From the figure above, PCA and t-SNE try to cluster the classes in our dataset. The main parameters of the t-SNE, number of iterations *(n_inter*) were set to 300, and perplexity was assigned to 700. From the figure (Figure 3.5), the background and the stage 4 classes can be separated better than the other classes. However, the other classes, stage 1, stage 2 and stage 3, show overlapping patterns. This can result in high misclassification between these classes. The need for an algorithm that can extract features that can minimise misclassification is motivated.

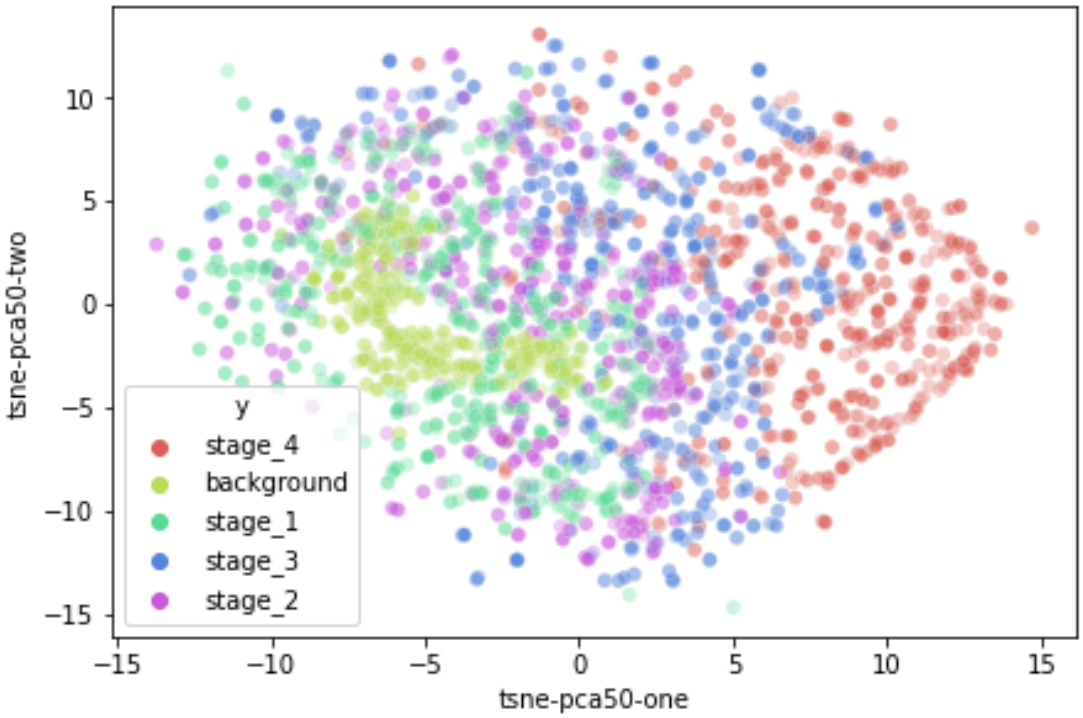

Figure 3.6 Visualisation of the radish dataset's five classes (background, stage 1, stage 2, stage 3, and stage 4).

Figure 3.6 shows the two-dimensional representation of the radish dataset. Clustering is seen in the background class and also stage 4 class. This makes the separation between these two classes easier. However, the other classes, stage 1, stage 2, and stage 3, overlap, and higher misclassification is most likely to be achieved in these classes. Therefore, the problem of growth stage classification is complex and requires a high-performance feature extraction algorithm to make classification possible.

3.7 Conclusion

This chapter introduced the dataset used in the following chapters of this dissertation to develop different models using different approaches. First, an outline of the preprocessing

done to the dataset by the creators is given. This includes the segmentation method used to eliminate the background from the plant matter, the method used to label the dataset, and the handling of images with very little plant matter to no plant matter. We have also described the method used to avoid bias and error. Lastly, we have regrouped the dataset into growth stage classes. Two datasets of two different crops, canola and radish, are created from the "bccr-segset" dataset. We visualised the dimensionality of the two datasets using PCA and t-SNE to understand the relation between the growth stage classes. From the visualisation, we observe a close similarity and overlapping between growth stage 1, growth stage class 2 and growth stage class 3 in both datasets. This motivates the development of a novel algorithm that can separate closely similar plants in different growth stage classes.

Chapter 4

Machine learning approach to develop a plant growth estimator

---

This chapter describes the method followed in developing a novel algorithm for plant growth classification using the dataset prepared in the previous chapter. We also outline the experimental setup and the procedure to fine-tune the SVM and XGBoost kernel hyper-parameters to achieve the best performance.

4.1 Introduction

In the previous chapters, we have provided discussions on the problems of plant identification, particularly in smart agriculture. We have also shown some challenges encountered in plant identification for classification problems. This chapter proposes a novel algorithm based on textural and morphological features for the growth stage classification of broadleaf plants grown under hydroponics systems. This algorithm can extract the plant from the background under different lighting conditions (natural and artificial). The algorithm uses Gabor filters to extract low-level textural features for feature extraction and a Sobel edge detector to extract the morphological features. We employed opening and closing morphological operations to generate contour masks

highlighting the leaf shape, size, and edges. We vary the Gabor filter parameters to generate kernels that extract the wanted features.

4.2 Algorithm design

This section motivates the principles and concepts to establish a novel growth stage identification algorithm. Gabor filters have shown excellent performance in many computer vision applications [27], [77]–[83]. Consequently, we have applied it in this study combined with morphological operations and a Sobel edge detector. Furthermore, plants at different growth stages pose closely similar features, which are complex to distinguish, motivating why a novel algorithm is required.

We tested two approaches to come up with this algorithm: the original input images to the feature extraction part and the other was to input both the original images and the masked images. Results from both approaches are captured and compared with other machine learning and deep learning methods to identify the best plant growth estimator.

4.2.1 The first proposed machine learning method

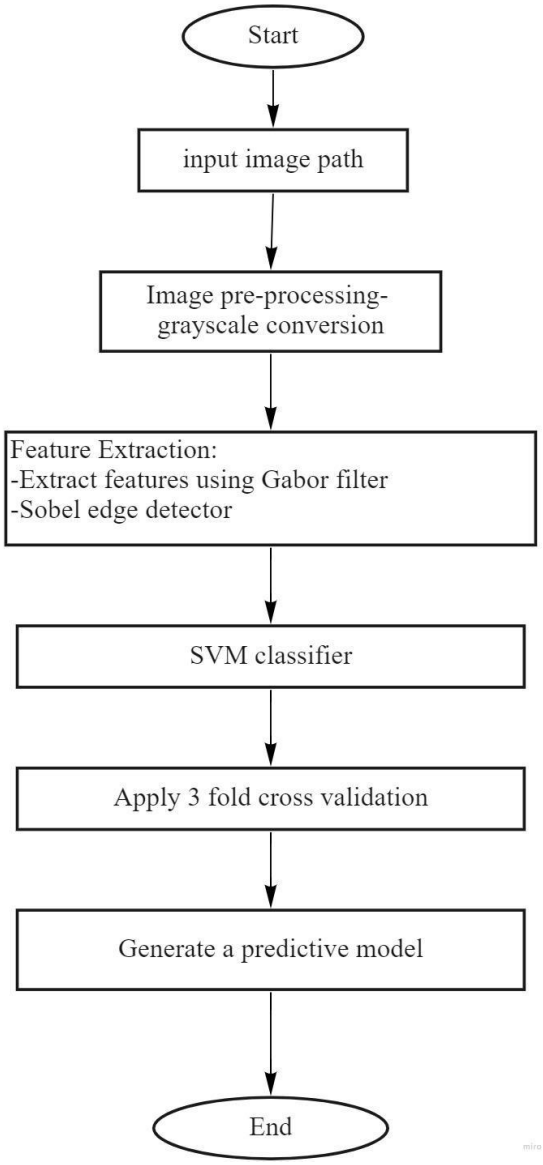

Figure 4.1. Proposed first method workflow

Figure 4.1 shows the flow of the first proposed algorithm. The steps are discussed in detail as follows:

- Firstly, the preprocessing explained in Chapter 3 is performed on the image dataset. Then, after all the images are converted to greyscale, and finally, the images are passed to the feature extraction stage.
- **Feature extraction stage:**

At the feature extraction stage, a bank of Gabor filters is generated by varying and combining different Gabor filter parameters such as wavelength, orientation, phase offset, sigma, and aspect ratio. The Gabor filters are developed in python. Gabor filters are nonlinear filters that extract and analyse texture features in images. A Gabor filter analyses if the image contains frequency content in the specified direction and a confined zone surrounding the analysis region. They are also rotation sensitive, just like the human eye. Filtering using Gabor filters is performed through convolution, where an image is convolved with a set of Gabor kernels created to detect specific frequencies and orientations. Computer vision scientists such as Daugman [84] have claimed that the frequency and orientation of Gabor filters are identical to that of the human visual system. This is important in discriminating texture.

Works that use Gabor filters refer to the original work presented by Dennis Gabor in 1946, where he proposed that signals are a collection of elementary functions [85]. An extension of this work was presented by Granlund in [87], where 2-D Gabor filters of the same elementary functions as the 1-D introduced by Dennis were introduced. Daugman [84] extended this work even further by introducing a generalisation of the 2-D Gabor filter and concluded the 2-D Gabor functions to be a model of cells in the human visual cortex.

A Gabor filter bank comprises kernels that detect four distinct wavelengths at eight different orientations from 0° to 180°. The orientation is limited to a half cycle of 180° because the response remains the same even if it is increased to a full cycle as only the phase is rotated by 180°. The response from each filter from the filter bank is saved for comparison. For four wavelengths/ frequencies over eight orientations, 32

responses are generated. The process to generate filter responses can be done in parallel without any interference. Therefore, no time is wasted in generating the filter response.

The following expressions (4.1 to 4.3) obtained from [80] are the Gabor filter formulas for detecting and analysing texture. A Gabor filter kernel comprises two parts, one is the real part, and the other is the imaginary part. The real part of the formula detects texture from the real part of an image and the imaginary part detects texture from the imaginary part of an image. Equation 4.1 is the real part expression of a Gabor kernel. Equation 4.2 is the imaginary part expression of a Gabor kernel. A combination of these two formulas creates a complex formula which is represented in equation 4.3.

Real

$$g_R(x, y, \lambda, \theta, \varphi, \sigma, \gamma) = exp\left(\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)cos\left(2\pi\frac{x'}{\lambda} + \psi\right) \tag{4.1}$$

Imaginary

$$g_I(x, y, \lambda, \theta, \varphi, \sigma, \gamma) = exp\left(\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)sin\left(2\pi\frac{x'}{\lambda} + \psi\right) \tag{4.2}$$

where:

$x' = x\cos\theta + y\sin\theta$ and $y' = -x\sin\theta + y\cos\theta$

λ = the wavelength of the sinusoidal factor.

θ = represents the orientation of the filter.

A combination of g_R and g_I, as $g = g_R + ig_I$, generates a complex formula defined as:

$$g(x, y, \lambda, \theta, \varphi, \sigma, \gamma) = exp\left(\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right) \tag{4.3}$$

ψ = is the phase offset.

σ = is the sigma/standard deviation of the Gaussian envelope.

γ = the spatial aspect ratio (the kernel's extension across the kernel wave pattern).

Varying the above parameters of the Gabor filter kernel expressions can change the orientation of the Gabor kernel, which is applied to the image. This can facilitate feature extraction.

In a discrete domain, a two-dimensional Gabor filter kernel is expressed as the cosine, G_c and the sine G_s as follows.

$$G_c[i,j] = Be^{-\frac{(i^2+j^2)}{2\sigma^2}} \cos cos\,(2\pi f(i\cos cos\,\theta + j\sin sin\,\theta)) \tag{4.4}$$

$$G_s[i,j] = Ce^{-\frac{(i^2+j^2)}{2\sigma^2}} \sin sin\,(2\pi f(i\cos cos\,\theta + j\sin sin\,\theta)) \tag{4.5}$$

where:

- B and C are normalizing factors that need to be determined,
- f is the frequency of specific texture to be extracted in the image.

By varying θ, we can look for texture in a certain direction, and by varying σ we can change the kernel size, thereby changing the size of the region being analysed in an image.

An example of how the Gabor filter works is presented below in Figure 4.2. Images can be represented with a matrix of pixel values.

Input image =

150	200	0	70	140
230	38	62	132	161
30	50	122	200	22
95	116	154	200	21
100	182	255	15	80

Gabor Kernel =

-1	0	1
2	1	2
1	-2	0

Output after kernel operation =

			70	140
	351		132	161
			200	22
95	116	154	200	21
100	182	255	15	80

Figure 4.2. Convolution process. An input image of size 5×5 is convolved by a Gabor filter kernel of size 3×3.

Convolutional operation through introducing a Gabor filter to an image to detect features by an element-wise multiplication is done as below,

*(-1*150) + (0*200) + (1*0) + (2*234) + (1*38) + (2*62) + (-1*30) + (-2*50) + (0*122) = 351*

The output value from the convolutional operation replaces the values where the operation was done. We moved the kernel to the next part of the image through a process called striding. From literature, a stride of 1 is used. A Gabor kernel can extract features such as texture and edges through this operation. According to literature, Gabor filters outperform other feature extracting methods in texture analysis [27], [77], [83], [137] and [139]. Since one of our goals is to reduce computation while maintaining classification, we have chosen Gabor filters which are excellent in texture analysis.

We can enhance classification with Gabor filters for feature extraction. The essential features for growth stage identification are texture, leaf shape, size, and edges. However, broadleaf plants at different growth stages have almost the same colour. Therefore, the colour feature is not significant for classification. For this reason, we did not consider colour features in the development of this method.

We apply the Gabor filter kernels to the images and features are generated. We then passed the features to the classification stage.

A detailed explanation of how a bank of Gabor kernels can be generated is given in Chapter 6, the implementation section.

- Sobel edge detector [87] is also applied at the feature extraction stage to enhance the extraction of edges.

- **Classification stage**

  At the classification stage, we employed stratified cross-validation. Stratified cross-validation [88] makes sure that each class is represented in each fold with the same number of samples. This is to have an equal mean response value (MRV) in all folds. With stratified cross-validation, over and underrepresentation of certain classes in the cross-validation folds can be avoided. Also, with stratified cross-validation, generalisation and variance are improved.

  Fig 4.2 Shows k-fold cross-validation where the samples are distributed across five-folds, and Fig 4.3 shows stratified cross-validation where an equal number of samples from each class are distributed across all folds.

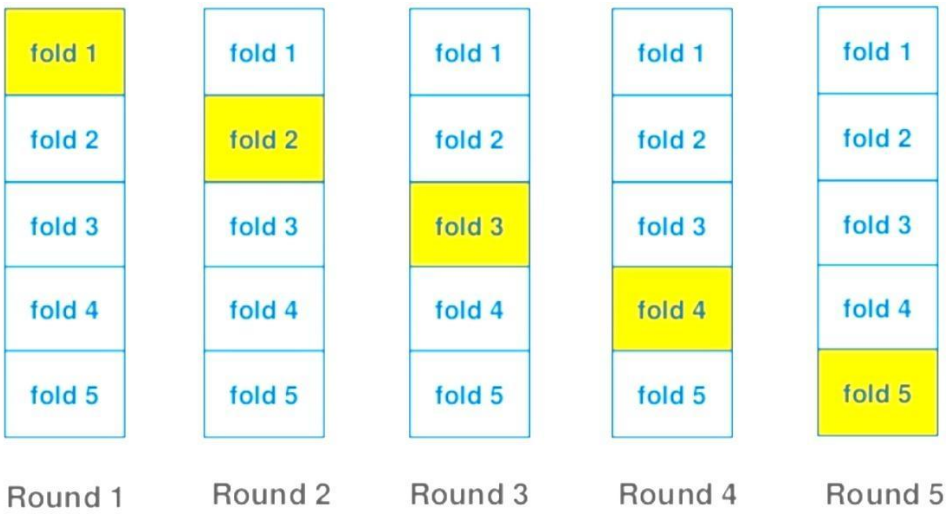

Figure 4.3. The distribution of samples in K-fold cross-validation. K-fold divides the training samples into K-folds.

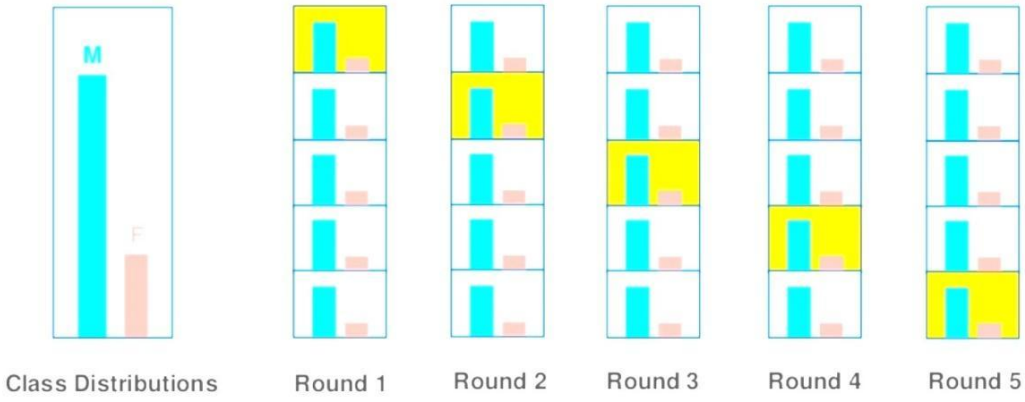

Figure 4.4. The distribution of samples in 5-fold stratified cross-validation. Stratified cross-validation ensures each class is represented equally in each fold by dividing the samples in each class by the number of folds and then distributing that across all folds.

- **SVM model**

Support vector machines (SVM) are among the best machine learning supervised classification models. Boser et al. [89] introduced SVMs in 1942. SVM maps the training elements into classes and tries to extend the hyperplane, separating them to distinguish them accurately. Among the classification methods, SVM has proved to perform better in many applications, such as facial recognition [90], bioinformatics [91], and diagnosis of diseases [64]. When classifying nonlinear data, SVM can recognize a pattern while reducing noise and misclassifications [93]. Mathematical functions known as kernels are used in SVM algorithms to perform nonlinear classification. An SVM has many kernels functions whose performance varies according to the problem at hand. An appropriate SVM kernel can be selected experimentally by testing and visualizing the response while adjusting the kernel hyperparameters. Radial basis function (RBF) and

polynomial kernels are the most well-known kernels for high-dimensional data classification. In this work, we have selected the RBF kernel to classify growth stage classes after experimenting with both RBF and polynomial kernels. As described above, the plant growth stage could be classified by combining an SVM with Gabor filters, Sobel edge detectors, and morphological operators.

The radial basis function (RBF) kernel on two samples x_i and x_j is expressed as:

$$k(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2) \tag{4.6}$$

Where: $k(x_i, x_j)$ is the kernel function, $\gamma > 0$ or $\gamma = \frac{1}{2\sigma^2}$, x_i and x_j are feature vectors whose Euclidean distance is calculated, $\|x_i - x_j\|^2$ is the square Euclidean distance measured between two feature vectors, and σ is the variance and the hyperplane.

The maximum value of the RBF kernel function is 1. This can be attained if $x_i = x_j$. Meaning the points are similar.

When the points x_i and x_j are far from each other, the RBF kernel value is less than 1 and closer to 0, which means the two points are not similar.

The RBF kernel of the SVM is implemented in Scikit-library. It has two hyperparameters, γ (gamma) and '*C*'. Here, γ (gamma) is inversely proportional to σ (sigma).

$$\gamma \propto \frac{1}{\sigma} \tag{4.7}$$

Optimising these hyperparameters can lead to a classifier classifying nonlinear data.

SVM hyperparameter tuning procedure

Firstly, to select the best non-linear kernel, experiments are carried out using the non-linear kernels such as polynomial, RBF, and quadratic given in the Scikit-library.

Since our data is non-linear, the linear kernel is dropped. For the remaining non-linear kernels, polynomial, quadratic, and RBF, a randomised search with cross fold cross-validation is done to find the best combination of C and γ parameters. The most important step in this method is defining the correct scale to search for hyper-parameters. But firstly, data normalisation is done to the feature data frame to reduce redundancy. A list of the SVM kernels to try and observe if they can achieve a better score is also specified during the randomised search. Finally, we set the number of iterations to run the search. Also, cross-validation is used to ensure that the best combination of parameters is achieved.

A detailed explanation of finding the SVM hyperparameters is given in Chapter 6.

Generation of a predictive model

After performing the steps as illustrated in the flowchart (Figure 4.1), a model is created, which is the first proposed method. This model is trained and validated using 3-fold stratified cross-validation. In 3-fold stratified cross-validation, images are randomly shuffled and one set is taken as the testing set, while the other two are used for training. This process is repeated three times and at each round, a different class is used as the testing set. There is no data leakage in cross-validation as it makes sure the data in the testing set is not used for training at the same time. The implementation of cross-validation and the whole algorithm is explained more in detail in

Chapter 6. After cross-validation, a predictive model which is ready to make predictions is created. This model's performance, robustness, and generalisation are tested using separate test data partitioned as shown in Figures 6.6 and 6.7.

4.2.2 The second proposed machine learning method

Figure 4.4 shows the second proposed method, which is a few extra steps in addition to the previous one. The same image dataset, preprocessing, feature extraction method, and classification algorithm are still used in this algorithm. The extra steps are the morphological operations, which are the opening and closing processes performed to all the images to extract their contour masks.

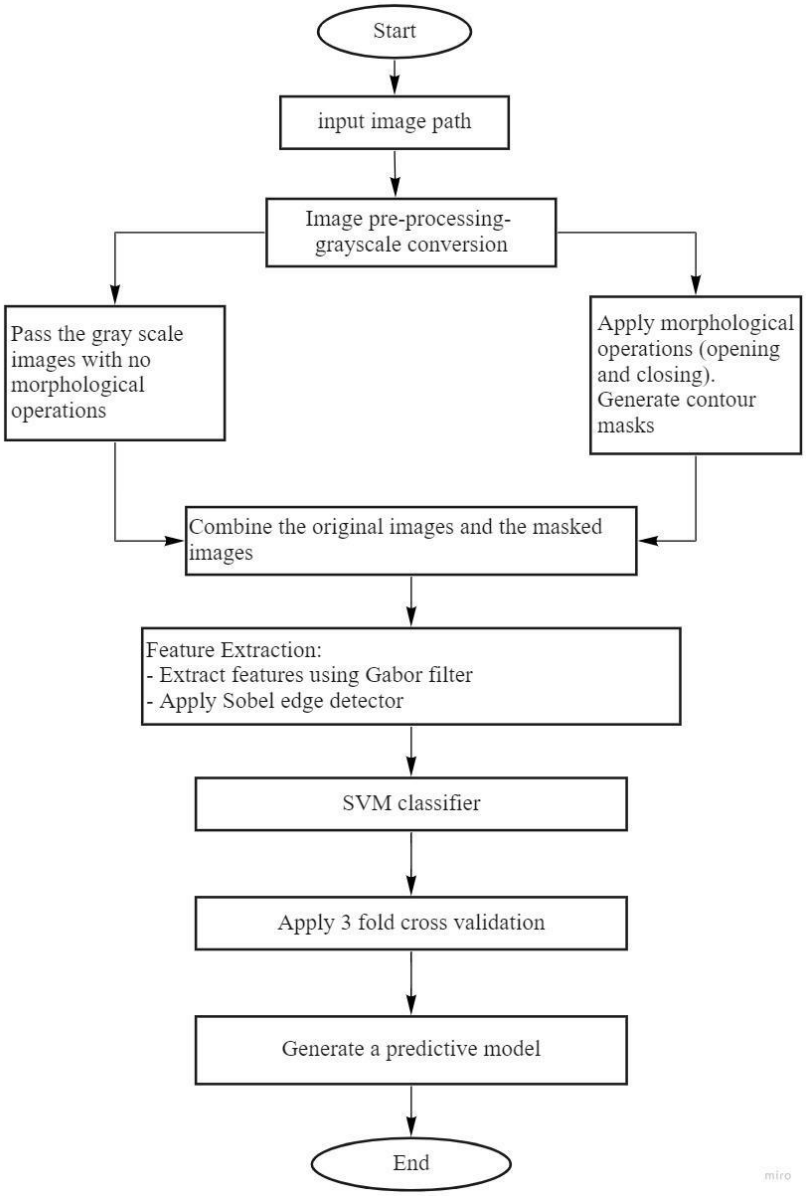

Figure 4.5. The second machine-learning method's development steps include morphological operations, Gabor filter kernels, Sobel edge detector, Stratified cross-validation, and a classification algorithm (SVM or XGBoost).

Algorithm steps

- The image dataset is preprocessed and converted to greyscale. The image dataset is passed through two paths and these are explained in detail as follows.

- **The left branch**

We pass the grayscale images to the feature extraction stage with no morphological operations performed in the left branch.

- **The right branch**

We pass the grayscale image dataset through the right branch, where it goes through a pre-processing phase of morphological operations.

Morphological operations

Morphological operations aim to remove noise in the images and highlight the edges [96]. This improves the feature extraction process. The morphological image processing in this work requires a grayscale image and a structuring element as the inputs. It works by transforming an image from one set to another while searching for the object using a structuring element. The information obtained from the transformation process is saved on the transformed set. We have employed two morphological operations (opening and closing) of size 3×3. Opening and closing morphological operations extend the basic morphological operations: erosion and dilation [96]. The erosion process is performed first in the opening operation, followed by the dilation process. This smoothens the object's contours. In the closing process, the dilation process is performed first, then the erosion process follows. Finally, closing fills any gaps in the contour region.

Erosion:

The erosion process uses a structural element to remove pixels in the boundary region, shrinking the object. Pixels are set to zero if the neighbouring pixels have a pixel value of zero. Small objects are eroded through the morphological erosion process, leaving only the significant objects.

Dilation:

The dilation process uses a structural element to expand the boundary of an object. In a binary image, the dilation process sets a pixel to 1 if some of the neighbouring pixels contain the value 1. Dilation increases the visibility of objects in an image. Small gaps are closed.

As a result, contour masks of the objects in the image are generated through the opening and closing morphological operations by highlighting the edges of the objects. This process is shown in Figures 4.5 to Figure 4.9.

For a grayscale image $I(x, y)$, and a structuring element $S(a, b)$, the erosion operator can be expressed as:

$$I \ominus S = min\{T(x + a)(y + b) - S(a, b)\} \tag{4.8}$$

Where: x and y are the dimensions of an input image (horizontal and vertical axis), a and b are the dimensions of the structuring element, T is translation, and $\ominus$ is the erosion operator.

For the same image $I(x, y)$ and structuring element $S(a, b)$, the dilation operator can be expressed as:

$$I \oplus S = min\{T(x - a)(y - b) - S(a, b)\} \tag{4.9}$$

Where: $\oplus$ is the dilation operator.

From the erosion and dilation, the opening and closing operators become:

$$I \circ S = (I \ominus S) \oplus I, \tag{4.10}$$

$$I \cdot S = (I \oplus S) \ominus I \tag{4.11}$$

Where: $\circ$ is a symbol for the morphological opening operation and $\cdot$ is a symbol for the morphological closing operation

Considering the size of plant images inside the images and the need for a faster model that can do feature extraction in real-time, we selected a structuring element capable

of size (5×5) for the opening and closing operations. The output images from the opening and closing operations showed that this structuring element was effective. Figures 4.5 to 4.9 summarise the process of opening and closing; firstly, Figure 4.5 shows the original image before morphological operations, the opening process is performed on this image, noise is removed and bigger objects are highlighted as shown in Figure 4.6, the closing process is performed on the images after the opening process as shown 4.8, In Figure 4.8, thresholding is done and a thresholded image is generated. Finally, a contour masked image is created, as shown in Figure 4.9, and this is passed to the feature extraction stages.

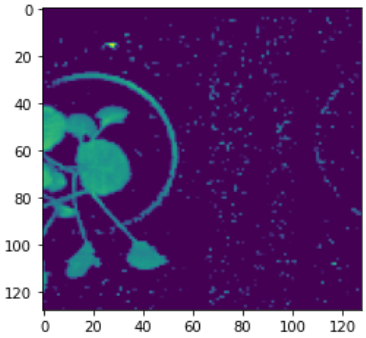

Figure 4.6. Original image.

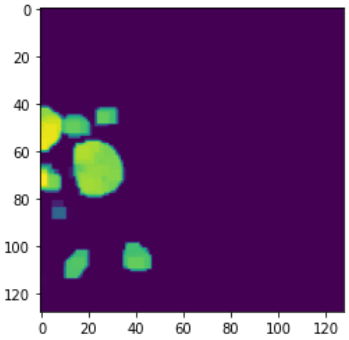

Figure 4.7. Opened image.

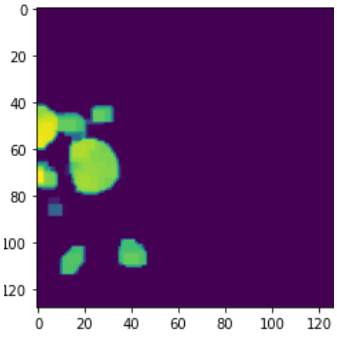

Figure 4.8. Closed image.

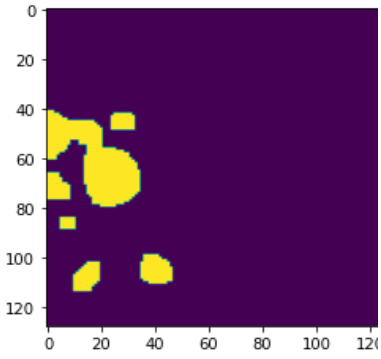

Figure 4.9.Thresholded image.

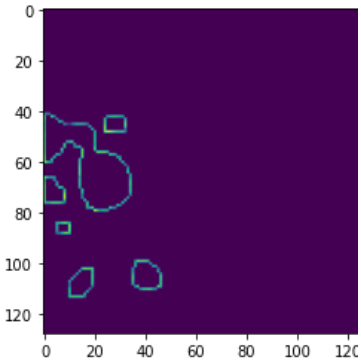

Figure 4.10. Contour masked image.

- **Feature extraction**

The exact process applied in the previous algorithm is applied to the combined dataset of contour masked images and the original images for feature extraction. This is the application of Gabor filter kernels and a Sobel edge detector to the images to extract both the textural features and the morphological features.

The extracted features from the combined dataset are passed to the classification stage.

- **Classification stage**

  The same processes and steps applied in the previous algorithm (the first approach) are used at the classification stage. These are the applications of 3-fold stratified cross-validation, RBF kernel SVM. In addition, the same RBF SVM kernel hyperparameters tuned in the first method are used since the dataset is the same. The second prediction model is generated using the second proposed approach.

Summary of the two proposed methods to come up with the best growth estimation model

In the above sections, section 4.1 and section 4.2, two methods to come up with a growth estimation machine learning models are proposed. One method involves passing the image dataset directly to the feature extraction stage, and the other method involves including morphological operations. These methods are implemented in Chapter 6 of the dissertation. These methods are both tested and evaluated to observe the best method.

4.3 Replacing SVM algorithm with XGBoost algorithm

To verify if SVM is the best algorithm, we replaced it with eXtreme Gradient Boosting (XGBoost) [97] in both approaches explained above. XGBoost is a machine learning algorithm that has attained ground-breaking results in many Kaggle competitions [98]. The main reasons behind the success of this machine learning algorithm are its scalability and the ability to process faster by focusing only on the most important features. Furthermore, XGBoost learns from labelled data during training. For example, in [99], XGBoost was used to focus sales. It obtained the lowest root mean square error (RMS) score of 0.655 while the other experimented supervised models, linear regression and ridge regression obtained RMSSE scores of 0.783 and 0.774, respectively.

XGBoost has also proven to be efficient with time-series data for example, in learning factors affecting the price of crude oil and further predicting the future price in [100]. XGBoost has also been used with random forest (RF) to detect faults in turbines [101]. In this work, Random forest was used to rank the contributing features by their importance. Based on the high-ranked features, XGBoost trains the classifier for each fault. A high-performance classifier is obtained this way. In [101], XGBoost was also used to diagnose kidney disease [102]. This work obtains an accuracy, sensitivity, and specificity of 1.00 across all metrics. Researchers stated that selecting only the most important features to train the classification model is efficient as it saves time and maintains high accuracy, specificity, and sensitivity. We chose to test its efficiency against SVM on this problem for these reasons.

4.3.1 XGBoost hyperparameters

XGBoost has some weights that determine how the algorithm learns. Tuning these hyperparameters can leverage the power of XGBoost to pick up patterns in non-linear data. Because XGBoost is a tree-based algorithm, the learnable parameters are the decision variables at every node. The hyperparameters include the depth of the XGBoost tree, the number of variables to consider when building the tree, and even the number of trees to grow.

There are four types of hyperparameters in XGBoost: the general, the booster, the learning task, and the command parameters. The general booster and task parameters are set before running the XGBoost model. The command parameters are used in the console version of XGBoost.

The general parameters contain three important hyperparameters: verbosity, booster, and thread. The booster parameter is important in choosing the booster to choose at each iteration. The options are GBtree, gblinear, and dart. GBtree and dart use tree-based models while gblinear utilizes linear models. The parameters verbosity has three valid values, which are 0 for silent, 1 for warning, and 2 for debug. Finally, Nthread is the maximum number of threads used to run the XGBoost.

The booster parameters are of two types, the linear and the tree booster. Tree booster, in most cases, outperforms the linear booster. Therefore, it is mostly used. The eta (shrinkage or learning rate) is the step size used to avoid overfitting. At every boosting step, eta shrinks the feature weights, making the process conservative and avoiding overfitting. Gamma is the other parameter used to specify the minimum reduction in loss required by the model to split a node. Max_depth is another parameter that limits the model from overlearning. Allows the model to learn relations up to a certain sample or level. Lambda: handles regularisation in XGBoost. When the lambda value is increased, the model becomes more conservative. Alpha: also handles regularization in the model.

When there is very high dimensionality, alpha speeds up the algorithm. Scale_pos_weight[default=1], balances the positive and the negative weights. It is instrumental when there is a class imbalance in the input data. Max_leaves: specifies the maximum number of nodes to be added.

The learning task parameters are used to define and set the metrics to be calculated at each step. These parameters specify the learning objective. Some of the objective options are: **objective [default = reg:squarederror].** This defines and specifies the loss function to be minimized. Some of the common variables are:

reg: squared error: regression with loss squared.

beg:logistic: logistic regression

binary:logistic: logistic regression for binary classification.

multi:softmax: use SoftMax to perform classification in XGBoost.

Booster parameters: These parameters guide the individual booster (tree) at each step.

min_child_weight**:** This is the minimum sum of weights of all observations required in a child. It also controls overfitting. Higher values prevent a model from learning relations that are unique to the sample used to build a tree.

max_depth [default=6]**:** The maximum depth of a tree.

colsample_bytree [default=1]**:** This value represents the ratio of columns that will be randomly sampled for each tree.

Gamma [default=0]**:** A node is split only if the resultant split reduces the loss function by a positive amount. The minimal loss reduction required to do a split is specified by gamma.

These are some of the XGBoost hyperparameters that can be tuned to increase performance.

XGBoost hyperparameter tuning procedure

To tune the XGBoost hyperparameters, a small dataset with samples that represent the dataset well can be created to reduce the computation power and time. Then, the random search can be employed to search for the best hyperparameter combination using the Scikit learn library. The most important XGBoost hyperparameters which can impact results are the learning rate, max_depth, min_child_weight, gamma, colsample_bytree. Random search looks into the specified values to find the ones that give the highest accuracy score. In random search, we specify the classifier, the XGBClassifier, the parameters we listed above, the number of iterations (n_iter), cross-validation (cv), and verbose. Then we visualised the best parameters for our data from the random search by calling the function (random.search.best_params_).

4.4 Conclusion

This chapter has proposed two methods to develop a plant growth estimation algorithm based on the traditional machine learning approach of handcrafting the feature extraction method and the classification algorithm. We developed these methods on the 'bccr-segset' dataset described in Chapter 3. We design the algorithms to extract the low-level features such as texture, size and shape to classify plants at different growth stages. However, it is challenging when closely similar plants need to be distinguished. Therefore, we have constructed these algorithms to address this challenge. We have also discussed XGBoost as a potential classification algorithm in both proposed approaches. The implementation of this algorithm is discussed in Chapter 6.

Chapter 5

Deep learning approach to develop a plant growth estimator

This chapter discusses the deep learning approach to developing a plant growth estimator using the dataset prepared in chapter 3. We provide background on the component of a deep learning network and how transfer learning can be applied using the deep learning network as a classifier.

5.1 Introduction

This chapter presents a deep learning approach for plant growth estimation through transfer learning. Deep learning allows models to do classification tasks directly from images, text, or voice. Deep learning is implemented using neural networks. The term “deep” from “deep learning” refers to the number of layers in a network; a deep network has more layers.

Data is transferred between nodes in these multi-layered deep neural networks. The result is a nonlinear transformation that becomes progressively abstract. While deep learning

requires an enormous amount of data to construct a high-performing system, these models do not require the handcrafting of the feature extraction part like machine learning. Convolutional neural networks are image-processing algorithms specifically developed to deal with images. The technique of transforming an image by applying a kernel to it is called 'convolution'. The kernel is a matrix of a certain size and values that determine the transformation effect.

5.2 Convolutional neural networks

CNNs are the typical deep learning models used often in computer vision tasks. Their architecture consists of an input layer, hidden layers, and an output layer. This architecture is similar to the ones of ANNs. In addition, CNNs have linear operators that employ core grid geometry. For example, the k^{th} layer of a network can be represented by an $(m \times m)$ grid as shown in Table 5.1 below:

Table 5.1. CNN grid representation [103]

$h_{1,1}^{(k)}$	$h_{1,2}^{(k)}$	. . .	$h_{1,m}^{(k)}$
$h_{2,1}^{(k)}$	. . .		
. . .		. . .	. . .
$h_{m,1}^{(k)}$			$h_{m,m}^{(k)}$

The function $h_{1,1}^{(k)}$ can be defined by convolving a matrix of size 2 x 2 and a nonlinear function g in the layer [103]:

$$h_{i,j}^{(k+1)} = g\left(a^{(k)}h_{i,j}^{(k)} + b^{(k)}h_{i+1,j}^{(k)}c^{(k)}h_{i,j+1}^{(k)} + d^{(k)}h_{i+1,j+1}^{(k)}\right) \quad (5.1)$$

The variables $a^{(k)}$, $b^{(k)}$, $c^{(k)}$, $d^{(k)}$ dependents on the layer, not a particular square i, j. After the convolution process where a (2×2) square and a nonlinear function g is applied to the function $h_{i,j}^{(k+1)}$, these functions are replaced with an average or a maximum of all the neighbourhood functions, termed pooling. An example is shown below [103]:

$$h_{i,j}^{-(k+1)} = \frac{1}{4}\left(h_{i,j}^{(k+1)} + h_{i+1,j}^{(k+1)} + h_{i,j+1}^{(k+1)} + h_{i+1,j+1}^{(k+1)}\right) \quad (5.2)$$

CNN uses the idea of convolution to replace generic matrix products in its layers. Neurons in these layers take input, compute dot products, and use an activation function such as ReLU to prevent forwarding negative values to the subsequent neurons in the network. For example, image data is input on one end and a class score is an output on the other end. CNN assumes the input data as images always. This makes researchers specify parameters related to images in their models. As a result, the forward capacity becomes more efficient to realize, and the number of parameters in the network may be reduced indefinitely. CNN layers are organised into three dimensions: width, height, and depth. Each layer applies a mathematical algorithm to transform a 3D image into 3D image data. The three primary convolutional neural network layers are convolutional, pooling, and fully connected layers.

Convolution

The convolutional layers pass input images through convolutional filter kernels. Each convolutional filter kernel extracts certain features from the image. The convolution is done by sliding the kernel across the image, usually starting at the top left corner, and moving it through all the points where the kernel fits entirely within the image's bounds. Convolution also moves the data through an arrangement of convolutional channels, every one of which extracts certain features. The parameters at each layer are an arrangement of learnable channels. Each channel stretches out through the full input

volume. Each passage in the 3D yield might also be decoded as a neuron's yield, which looks at a tiny segment of the data and provides parameters with all neurons to one side and right. Each kernel location corresponds to a single output pixel, whose value is computed by multiplying the kernel value and the underneath image pixel value for each kernel cell, then summing the products. A mathematical representation of the convolution process is presented in Equations 5.1 and 5.2. The convolution process is also used in Gabor filters as illustrated in Chapter 4 using Gabor kernels and Figure 4.2.

In CNN, there is Local Connectivity, which means that each neuron is only related with neurons in the image volume's near area, rather than with all neurons in the previous volume. The open field of the neuron is a hyperparameter that describes the spatial extent of this availability. The depth of the input volume is always equal to the degree of availability along with the depth pivot. The span of the yield volume is controlled by three hyperparameters: depth, stride, and zero-padding, according to the spatial course of action. The depth corresponds to the number of filters we would want to use, each of which learns to look for something different in the input.

From Table 4.2, the output at the first location of the filter kernel is given by:

*(-1*150) + (0*200) + (1*0) + (2*234) + (1*38) + (2*62) + (-1*30) + (-2*50) + (0*122) = 351*

Pooling for dimensionality reduction in CNN

Pooling reduces the number of parameters in the neural network by downsampling [105]. The pooling layers, found after the convolution layers, reduce the dimensionality of the network while still returning the most crucial information. The network's dimensionality is reduced by reducing some network parameters (model coefficients). Network parameters are called weights. The model selects these weights during training which is called optimisation. The model developer specifies an optimisation strategy, and an array of parameters that reduces the error to the smallest possible value is returned. Some of the

widely used pooling techniques are max pooling and average pooling. Max pooling is when the maximum number is considered from the image area where the kernel filter was positioned. Average pooling is when an average pixel value is calculated from the filter kernel's image area. The output matrix is a dimensionally reduced image in height and width. With pooling, overfitting can be reduced, and the network's training time can be speeded up.

Rectified linear unit (ReLU) activation function

An activation function specifies how the weighted sum from the convolution process is turned into the output [106]. An activation function choice significantly influences the network's capabilities and overall performance. In CNN, various activation functions are used in different network parts. In hidden layers, ReLU activation function [107] is used. This is because it is susceptible to vanishing gradients that can stop model training. ReLU speeds up training by preventing negative values from being passed forward in the network and only allowing positive values.

ReLU is calculated as:

$$y= (0, x) \tag{5.3}$$

When the input (x) is negative, 0.00 is returned as the output. When the input (x) is positive, that value (x) is returned.

The three above operations are repeated many times during feature extraction in many layers.

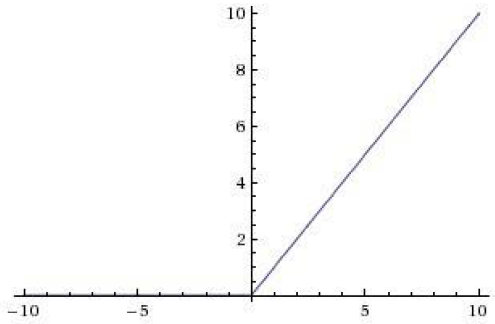

Figure 5.1. The ReLU activation function outputs the input *(x)* only if it is positive. Otherwise, the output is 0. ReLU does not suffer from the vanishing gradient problem, which other activation functions such as Sigmoid and tanh suffer from [108].

Stride and how striding works

The movement of the convolutional kernel to the next column pixel or the next row pixel of an image is called a stride. When a stride is set at (1,1), the kernel is moved by 1 pixel to the next column and then it is moved 1 pixel to the next row. A stride of 2 means the kernel is moved by two pixels or two units. For instance, if a filter of size 3×3 is used for convolution. 9 underlying pixels of the image will be converted to 1 pixel in the output layer. Striding works alongside padding. Padding avoids the shrinkage of an image by adding empty pixels to the boundary of the image. Figure 5.2 below is a visual representation of the striding.

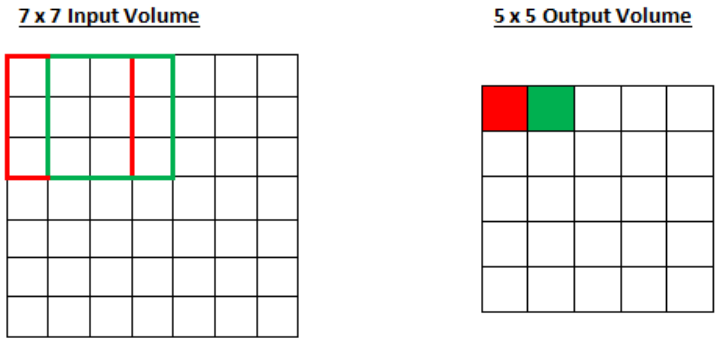

Figure 5.2 Stride and padding. A 3×3 filter convolves over an input image of size 7×7 and a stride of 1. Red is the initial position and green is after a stride of 1 [109].

From the above Figure where a filter of size 3×3 is used, the 9 pixels occupied by the filter will be converted to 1 pixel in the output layer. As the stride is increased, the output decreases. Therefore, the main aim of striding is to compress an image or video data by determining how much of the previous pixels' information is included in the output.

Padding

The image size is not always an integer multiple of the kernel size and stride. Therefore, information at the boundary of the images gets lost sometimes. The solution to this problem is to stuff the image with a border of additional pixels to make the kernel fit the input image. There are different types of padding such as zero padding, same padding, and causal padding.

Zero padding: Zero padding is a common approach for making the size of an input sequence equal to a power of two. You add zeros to the end of the input sequence in zero padding to make the total number of samples equal to the next higher power of two.

Same padding: In this type of padding, the padding layers attach zero values to the outer frame layer of the image.

Causal padding. Works well with one-dimensional data, especially time-series data by adding zeros at the beginning of the data. This helps in predicting early time step values.

When an image of size *(n x n)* is being filtered by a kernel of size *(f x f)*, an output matrix of size *(n-f +1) x (n-f +1)* is produced. For example, an image of size *(6 x 6),* a kernel filter of size *(3 x 3),* and a stride of 1, the dimension of the output matrix is:

(n-f +1) x (n-f +1) = (6-3 +1) x (6-3 +1) = (4 x 4)

We can see that the image has been shrunk after this convolution. This means the image is losing some information after each convolution. To avoid this, padding can be applied before convolution. When padding is equal to 1 (*p = 1*) for an input image of size (*6 x 6)* and a kernel filter of (3 x 3), the output dimension can be obtained as follows:

(n+2p-f+1) x (n+2p-f+1) = (6+2x1-3+1) x (6+2x1-3+1) = (6 x 6)

The input size is maintained by padding.

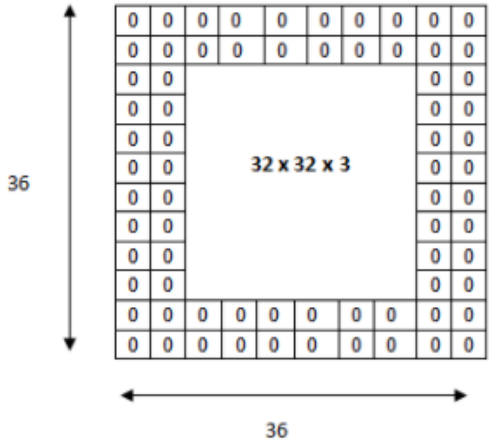

Figure 5.3. Two borders of zero have been added around the image of input size volume 32×32×3, this results in a 36×36×3 volume. When a convolutional layer with three 5×5×3 filters and a stride of 1, then an output volume of 32×32×3 is obtained.

Weights and bias

In neural networks, connections between nodes in a layer and nodes in different layers are weighted, which means that the weight represents how much influence the input from the previous node has on the next node [110]. To correctly compute an artificial neuron mathematically, the products of the inputs ($x_1 \; to \; x_n$) and their associated weights ($w_1 \; to \; w_n$) are summed, bias (b) is added to the sum, as shown in Figure 5.4. The total sum of all the operations is fed to the activation function (f), which becomes the output.

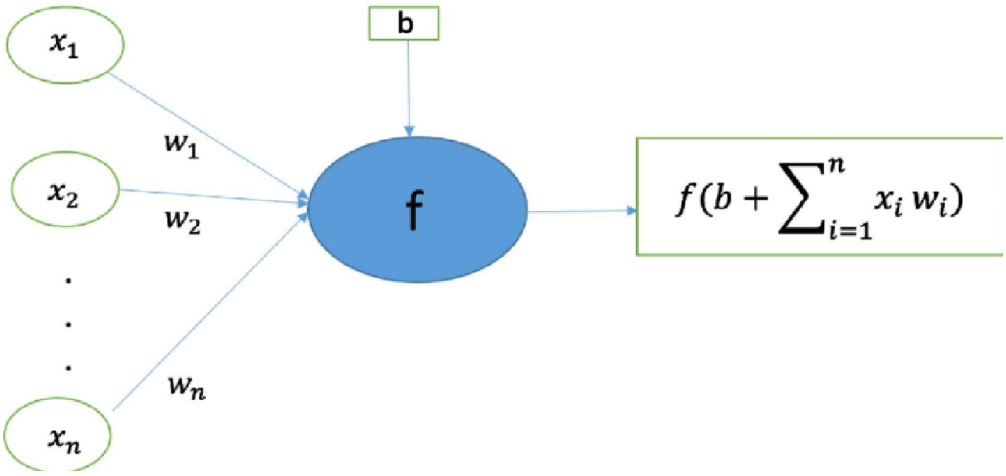

Figure 5.4. Artificial neuron computation: With input $(x_1 \, to \, x_n)$, weights $(w_1 \, to \, w_n)$, bias (b), and activation function (f) [111].

Biases

A bias is an additional input to the artificial neuron, as shown in Figure 5.4. Bias allows the activation function curve to be moved to the left side or the right side on the coordinate graph, allowing the neuron to produce the desired output value.

Backpropagation

Backpropagation is a short form for backward propagation of errors. It facilitates neural networks in learning their parameters, mainly through error predictions [112]. Backpropagation is done using gradient descent [113]. The gradient is computed backwards across the network; the gradient of the last layer is calculated first, and the gradient of the first layer is calculated last. This backward flow of error information enables efficient and accurate gradient computation at every layer.

A good mathematical description of backpropagation can be found in [114].

Loss function

A loss function is a type of error measure that may determine how accurate predictions are. The difference between the predicted output and the outcome generated by the machine learning model is quantified by the loss function in a neural network. We can obtain the gradients that are utilized to update the weights from the loss function. The cost is calculated as the average of all losses. A neural network model tries to understand the probability distribution underlying the given data set observations. The statistical framework of maximum likelihood estimation is often used as a basis for model development in machine learning. This basically means we try to find a set of parameters and a prior probability distribution such as the normal distribution to construct the model that represents the distribution over our data. Deep learning models aim to reduce this loss function value, which is referred to as optimization [108]. Types of loss functions are, binary cross-entropy, categorical cross-entropy, sparse categorical cross-entropy and mean squared error. Cross-entropy based loss functions are mainly used in classification problems. The difference between two probability distributions is measured by cross-entropy. We wish to determine the difference between the probability distribution produced by the data generating process and the distribution represented by our model of that process in a machine learning environment using maximum likelihood estimation.

Binary cross-entropy:

The binary cross-entropy is used in binary classification situations to acquire one of two possible outputs, as the name indicates. The loss is estimated using the method below, where y denotes the predicted outcome and y hat is the outcome generated by our model.

$$L = -\left(y_i \log log\,(\hat{y}\right) + (1 - y_i) \log log\, 1 - \hat{y}_i)) \tag{5.4}$$

Categorical cross-entropy:

The categorical cross-entropy is appropriate in multiclass classification situations. In the binary cross-entropy formula, we multiply the actual outcome by the logarithm of the model's output for each of the two classes, then add them together. The same approach applies to categorical cross-entropy, only we now sum across more than two classes. The formula is given below where M is the number of classes.

$$L = \sum_{j=1}^{M} y_j \log log (\hat{y}_j) \tag{5.5}$$

Mean Squared Error

Mean squared error is employed in regression situations where your anticipated and predicted outcomes are real number values. The loss is calculated using a simple formula. It's just the difference in squares between the expected and predicted values.

$$L = (y_i - \hat{y}_i)^2 \tag{5.6}$$

Learning rate and Adam optimizer

In optimisation processes, a learning rate is the step size of each iteration. Convergence takes longer if the learning rate is set too low, and if the learning rate is set high, the model takes less time to converge [115]. The Adam algorithm is one of several optimization algorithms (optimizers) available. Adam is an optimization technique that updates network weights iteratively, Adam combines Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). AdaGrad improves performance with sparse gradients by maintaining a per-parameter learning rate. (RMSProp), additionally keeps per-parameter learning rates adjusted depending on the average of current gradient magnitudes for the weights (e.g., how quickly it changes). This indicates that the technique is effective for online and non-stationary issues (e.g., noise).

Some of the benefits of Adam are:

- Implementation is simple.
- Effective in terms of computation.
- It does not require much memory.
- Ideally suited to problems with a lot of data.
- It is a good choice for non-stationary goals.
- Appropriate for exceedingly noisy gradients.
- Hyper-parameters are easy to read and usually do not require much adjustment.

Fully connected layers

The fully connected layers do classification in convolutional neural networks. The fully connected layers are the last in the network. The output of the convolution layers is the input of the fully connected layers. This output is flattened and passed to the classification layers. Flattening is transforming the 3-dimensional matrix of the extracted feature data into a 1-dimensional vector. The 1-dimensional vector is coupled to the fully connected layers and calculations are done for each coupled fully connected layer as follows:

$$g(Wx + b) \tag{5.7}$$

Where.

x – input vector with height and width dimensions of $[p - 1, 1]$

W – Weight with height and width dimensions of $[p - 1, n - 1]$ where $p - 1$ is the sum of the previous layer's neurons. $n - 1$ Sum of neurons in the current layer.

b – is the bias vector of the same dimensions as the input.

g – is the activation function. ReLU is the most used activation function in the fully connected layers.

After the series of fully connected layers, the last layer of the network utilizes a SoftMax activation function to convert the one-dimensional data from the fully connected layer to a probability vector.

5.3 Feature extraction

This section explains how deep learning automates the feature extraction process to classify plant leaf images. Unlike traditional machine learning, where feature engineering is required, deep learning learns features automatically. In most cases, determining which features to consider is difficult, resulting in the limitation of machine learning in such scenarios. However, it is noteworthy that when essential features are selected in machine learning, the method becomes very effective and faster, which sometimes outperforms deep learning methods [99]. Figure 5.5 shows the lower layers of a CNN model are responsible for feature detection. These layers include the convolution layers and the pooling layers.

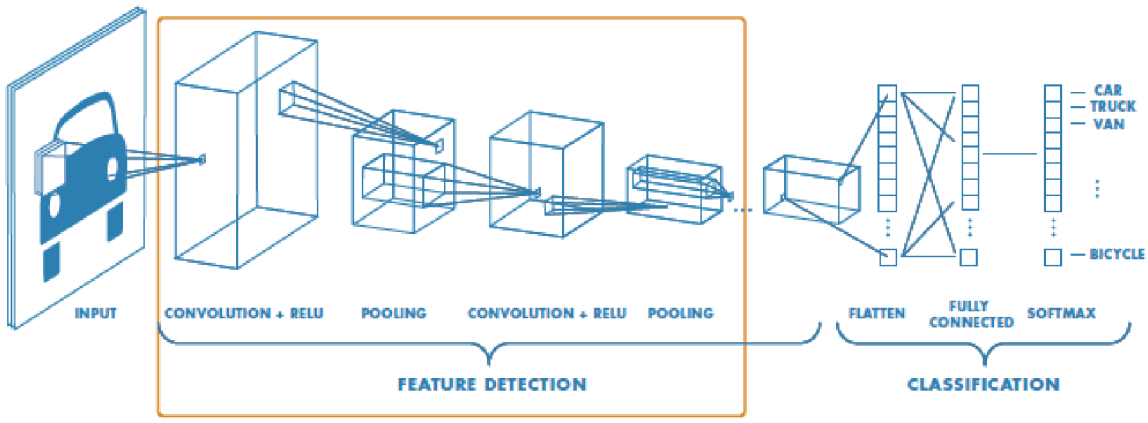

Figure 5.5. Feature extraction layers of a deep learning model [104]

5.3.1 Feature extraction using a pre-trained network

Deep learning's rapid adoption was mainly due to its exceptional performance on various tasks. Deep learning makes the process easier by automating the extraction of features. On the other hand, traditional machine learning approaches require transforming the input data into subsequent representation spaces using algorithms such as SVMs. Moreover,

machine learning approaches are sometimes incapable of producing the precise representations required by complex tasks. It requires more pre-processing effort from the scientists to make the algorithm handle the input data. This is referred to as feature engineering. Deep learning completely automates this process. Instead of selecting features, you pass the input data into the network once, and the network learns the important features. This is an improvement in machine learning. Feature extraction is the process of separating important features from a new dataset using knowledge acquired from a previous task. The extracted features are fed into a classifier which is completely retrained using the new dataset. CNN consists of stacked convolution and pooling layers that are connected to a dense layer which is the classifier. Convolution layers are responsible for extracting general features; new data is passed through the convolution layers of a pre-trained model and features can be extracted. The figure below illustrates this more.

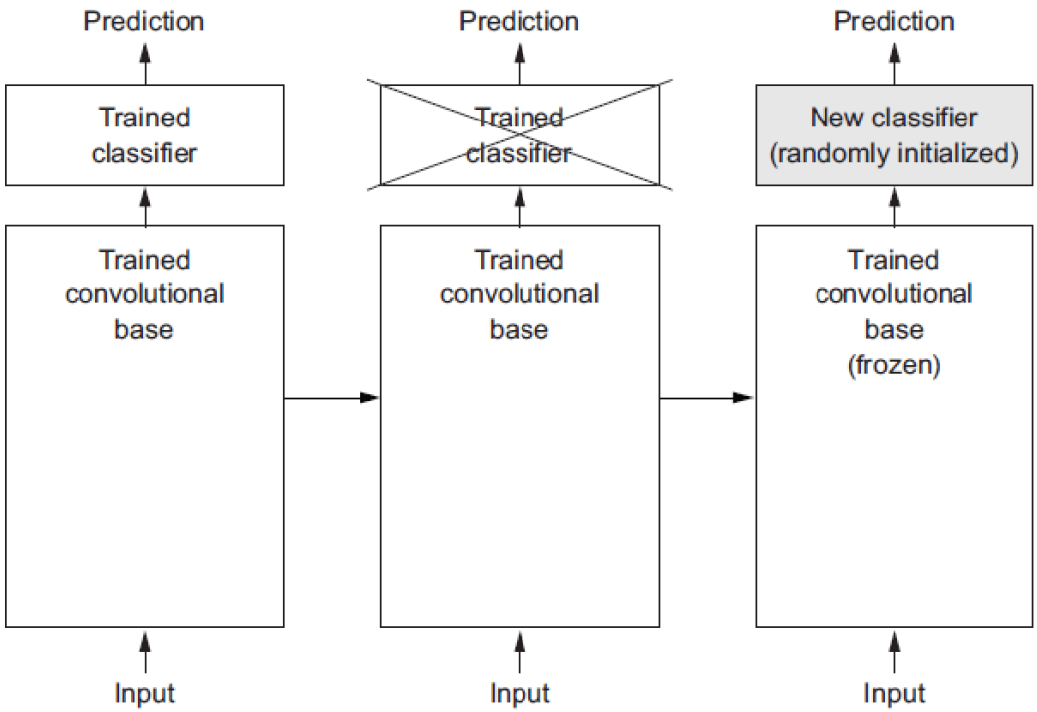

Figure 5.6 Using a pre-trained model for feature extraction [116].

From Figure 5.6, the model is frozen, except the classification layer is retrained from scratch. The convolutional layers of transfer learning models are useful in many problems as the knowledge learned is general and useful [117]. For example, the convolutional layers of a network trained on ImageNet pose knowledge such as detecting edges, colours, and texture. However, layers closer to the top (the fully connected layers) extract specific features limiting their reusability. Hence, the first few layers of a pre-trained model are frozen when reusing them to repurpose the knowledge on the new task. Reusing a pre-trained network while freezing the convolutional layers is computationally economical. No further training is done, but the model's previous knowledge on a different problem is reused. Therefore, in this work, this technique was employed to develop a deep learning approach to solve the problem of plant growth stage classification.

As discussed in the previous sections, convolutional neural networks are well known for their excellence in feature extraction across various computer vision application areas, such as visual recognition [118], plant disease identification [44], plant classification [10], and plant weed classification [119]. In a convolutional network, the dense layers (hidden layers) handle feature extraction purposes, and the fully connected layers, also known as the output layers, handle classification [104]. Layers in a convolutional neural network are interconnected through neurons. CNN layers comprise (3) three dimensions (width, height, and depth). We get height and width from the input image’s pixel numbers along the respective dimensions, and the depth dimension is the number of channels of the input image. For example, a grayscale image has one channel and a colour image has three channels: red, green, and blue (RGB).

5.4 Transfer learning and related works

Transfer learning [120] is a technique in deep learning where the knowledge gained by a model on a certain task is repurposed on another task; for example, pre-trained models

are repurposed to solve computer vision tasks in deep learning. Only the fully connected layers of the pre-trained model are retrained using the target dataset. This is helpful as these tasks require many resources, such as computation, large datasets, and data pre-processing, which is a challenge. By transfer learning, training is speeded up; computational power requirement becomes less as the models are not trained from scratch. Overly, performance has improved. Many research works have proved transfer learning to be very effective. Researchers in [121] compared transfer learning with models created from scratch for plant classification through plant leaf images. Their results showed high performance by the transfer learning technique of fine-tuning a VGG-16 pre-trained model on ImageNet. The experiment was run on four datasets. Transfer learning technique was also used for plant identification in [121]. In this work, transfer learning obtained an accuracy of 80% in plant identification and this was a 15% increase from the results obtained in a campaign [123] held on the same dataset. The effectiveness of transfer learning is also seen in pest classification in tomato plants.

An accuracy as high as 88.83% is obtained using a pre-trained model (DenseNet169). This pre-trained network is finetuned by removing the fully connected layers and replacing them with a layer that fits their dataset and the number of classes in their dataset. In [124], an experiment to evaluate the effectiveness of the transfer learning technique in crop pest classification was done. The pre-trained models, VGGNet, GoogleNet, ResNet, and AlexNet, were considered. Data augmentation was also employed in this work to increase the performance. A CNN model developed from scratch was also run side by side with these pre-trained models. The highest accuracy of above 95% was observed from the CNN model, which was better than all the transfer learning models. This work also proves that transfer learning is not always the solution to all classification problems. Therefore, in this work, we have chosen to test the well-known transfer learning models, which have attained ground-breaking results in many competitions against some handcrafted machine learning models in classifying plant growth. VGG-16 [125], VGG-19 [125], and ResNet-50 [126] are selected as the

transfer learning models to be used in this study to develop a plant growth estimation model.

5.5.1 VGG-16

VGG-16 [125] is a deep learning model originally trained on an ImageNet [127] dataset comprising millions of images of over 1000 classes. VGG-16 has 13 hidden layers and 3 fully connected layers. Researchers reached this architecture experimentally by fixing the network parameters while increasing the number of hidden layers from 11 to 16. We know that as the network becomes deeper, the more time it requires training and the more computational power the model requires to complete a task [125]. A (3 x 3) convolutional kernel, a stride of 1, and a padding of 1 are used for this network. The architecture has no spatial pooling between the convolutional layers. ReLU activation is used after each convolution to reduce dimensionality. Max pooling of 2×2 kernel size with a stride of 2 is also used to minimise the spatial dimensions. The creators of this VGG network proved that a stack of small convolutional filters of *(3 x 3)* receptive field size and a stride of 1 is better than the configurations considered top-performing in ILSVRC 2012 and 2013 competitions [125]. One was the work of Krizhevsky et al. [128], which used a receptive field of size $7x7$, a stride of 2, and spatial pooling in between layers. To prove this, the researchers assumed C as the number of channels. Therefore, a stack of small receptive fields (3 x 3) requires: 3 $(3^2C^2) = 27C^2$ Weights.

A (7 x 7) requires: $7^2C^2 = 49C^2$ Weights.

As a result, a small receptive field requires less than half of the weights needed by big receptive fields. The argument made by the creators of the VGG network attracted a lot of attention from many researchers as it does not require much computation because of fewer weights. In the 2014 ImagNet challenge, the VGG architectures came first and second in image classification and localisation problems. Since this architecture has been

used in many classification problems, some of the works are explained in the previous section of this chapter. These are the reasons why this network is selected in this work.

The VGG-16 model has the disadvantage of being expensive to assess. Even though it has fewer weights than the previous models, it still has about 138 million parameters which is a lot [129]. Therefore, it can be argued that a traditional machine learning model is quicker as it has fewer parameters than the VGG-16 model.

5.5.2 VGG-19

VGG-19[125] is similar TO VGG-16 except that it has a depth of 19. in contrast to VGG-16, which has a depth of 16. Both models are trained on the ImageNet dataset [127]. Similar to VGG-16, VGG-19 has convolutional layers of kernel size 3×3 with a stride of 1. Five max-pooling of 2×2 pixel window with a stride of 2 is used to minimise spatial dimensionality. There are two fully connected layers in the network comprising 4096 nodes. Lastly, the last fully connected layers consist of a SoftMax activation function to make predictions and classifications. All the convolutional layers consist of the ReLU activation function. Table 5.2 below shows the VGG network configurations; the columns show an increase in depth or number of layers from left to right. Depth is incremented from left to right while other parameters are fixed. The network parameters are presented as “Conv(receptive field size)-(number of channels)”. Where receptive field size is the region in input space that a particular CNN’s feature is affected by. While the channel number of output channels. The receptive field size is the same throughout the network. The number of channels increases as the number of rows increases but remains constant as depth increases.

The only difference between VGG-19 and VGG-16 is the number of layers and the total number of parameters in both networks. VGG-19 has more parameters than VGG-16.

They both follow the same idea of keeping the convolution kernel size small while designing a deep network. We selected VGG-19 for its success and state-of-the-art results in ImageNet classification [128]. Also, VGG-19 has been used in many applications and satisfactory results have been achieved. Table 5.2 shows the network configurations of VGG models and the network parameters as the layers increase from 11 to 19.

Table 5.2. VGG network configuration, from left to right, the depth of the network increases from 11 to 19 while other parameters such as receptive field and number of channels are kept constant [125]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

5.5.3 ResNet-50

ResNet-50 is a 50-layer residual network. It is part of residual models which use residual learning [126]. ResNet-50 is similar to VGG-16, except it contains an extra identity mapping feature. ResNet-50 is similar to VGG-16, except it includes an additional identity mapping feature. Figure 5.7 shows this identity mapping.

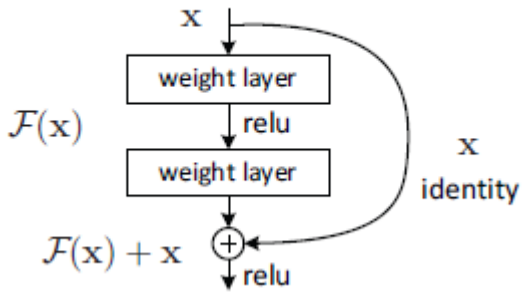

Figure 5.7. A building block of residual learning [126].

ResNet-50 forecasts the delta needed to get from one layer to the next and arrive at the final prediction [126]. ResNet-50 addresses the vanishing gradient problem experienced in many deep networks by enabling gradients to flow along an additional shortcut path. ResNet-50 can bypass or skip the layer if the current layer is not required by identity mapping. As a result, overfitting is prevented.

The following are some of the properties of ResNet-50 [126]. These properties are also shown in detail in Table 5.3.

- In the first convolutional layer, ResNet-50 has 64 kernels of size (7 x 7) and a stride of 2. Next is a max-pooling layer of stride 2 then a convolutional layer consists of 64 kernels of size (1 x 1) followed by another convolutional layer of 64 kernels of size (1 x 1), and lastly, a convolutional layer of 256 (1 x 1) size kernels. Finally, these three convolutional layers are repeated three times to produce 9 layers.

- The next layer consists of 128 kernels of size (1 x 1), followed by another layer with 128 kernels again of size (3 x 3), and the last layer of 512 kernels of (1 x 1) size.
- The next convolutional layer consists of 256 kernels of size (1 x 1), followed by another two layers of 256 kernels of size 3×3 and 1024 kernels of size 1×1. This is repeated 6 times to produce 18 layers.
- Next is a convolutional layer with 512 kernels of size 1×1, followed by two layers with 512 kernels of size 3×3 and a layer with 2048 kernels of size 1×1. This is done three times to give 9 layers.
- Average pooling is added to a fully connected layer with 1000 nodes with a SoftMax activation function to form the output player.

In total, ResNet-50 layers can be summed as follows: $1 + 9 + 12 + 18 + 9 + 1 = 50$.

Table 5.3. ResNet-50 architecture with a down sampling conv3_1, conv4_1, and conv5_1 using a stride = 2 [126].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet-50, through transfer learning has attained satisfactory success across various fields, such as in medical fields where it was used to classify malaria cell images [132]. It has also been used to classify breast cancer images [131]. Furthermore, in agriculture for plant disease classification [117]. Due to the success of ResNet-50 and its advantages, such as the elimination of vanishing gradients and the addition of residual connections which prevent overfitting, we selected it for this study.

5.6. Deep neural network feature extracting model.

From the presented information in the previous sections, it can be seen that although transfer learning through the discussed models has obtained excellent success in various applications, the models still contain a lot of parameters that require memory. For instance, the VGG network contains more than 188 million parameters. Most of these parameters are contained in the Fully connected layers (the classification layers) [129]. Therefore, we propose replacing these layers with a classification machine learning algorithm. By doing so, the pre-trained models will only be used to perform feature extraction. The classification algorithm such as SVM and XGboost will be used to perform classification, which significantly reduces the number of parameters in the model. Figure 5.8 and 5.9 shows how a model can be developed by combining transfer learning and a machine learning algorithm. This approach has not been explored much, specifically in agriculture, although promising results have been achieved in studies such as malicious software detection [132] and multi-parameter patient monitoring [129].

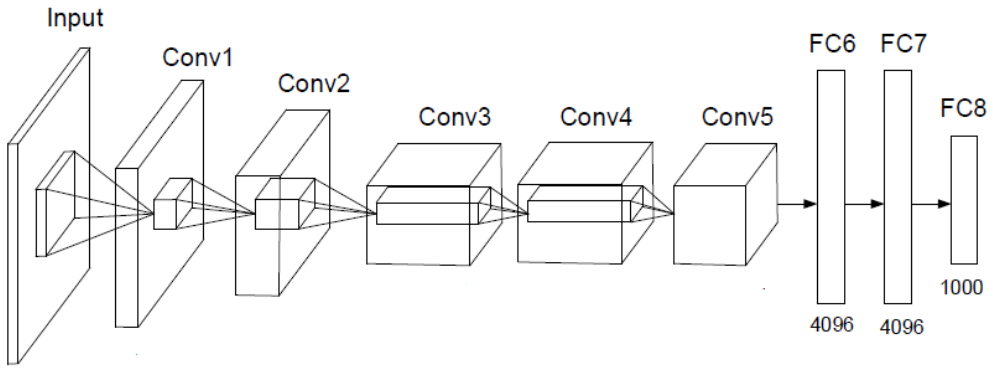

Figure 5.8. The general architecture of a pre-trained neural network.

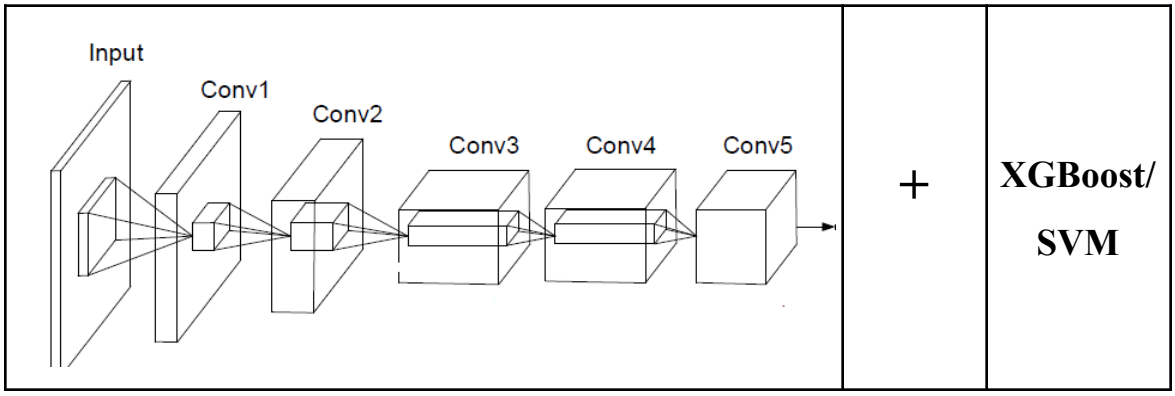

Figure 5.9. Removing the fully connected layers of a CNN model then concatenating it with a machine learning algorithm XGBoost or SVM.

5.7 Conclusion

This chapter has discussed the deep learning approach to develop a plant growth estimator. In addition, we have discussed the transfer learning approach, its advantages, and how a pretrained network performs feature extraction. We have also discussed the deep learning models employed in this study and we also give the motivation behind every deep learning model selected in this study. Lastly, we have proposed a third approach of using deep learning as a feature extractor and a machine learning algorithm as a classifier to solve this dissertation's primary problem: plant growth estimation.

Chapter 6

Implementation

This chapter discusses the procedure taken to implement the approaches (the traditional machine learning, deep learning models, and the combined deep learning and machine learning models) discussed in the last two chapters. The procedure for comparing the two is explained. Hyperparameter tuning of the deep learning models is also discussed in this chapter.

6.1 Introduction

The implementation of the methodologies discussed in the previous chapters, Chapter 4 and Chapter 5, follows the workflow shown in Figure 6.1. We start by splitting the dataset into two sets, training and testing, with a 90% to 10% ratio. The training dataset is used to train the model. Next, cross-validation is performed on this dataset. Then, a model is generated and tested on the test dataset. Finally, the developed models are compared using the metrics; Accuracy, confusion matrices, precision, recall and F1-score discussed in Chapter 7. The hardware and software specifications are also outlined in this chapter.

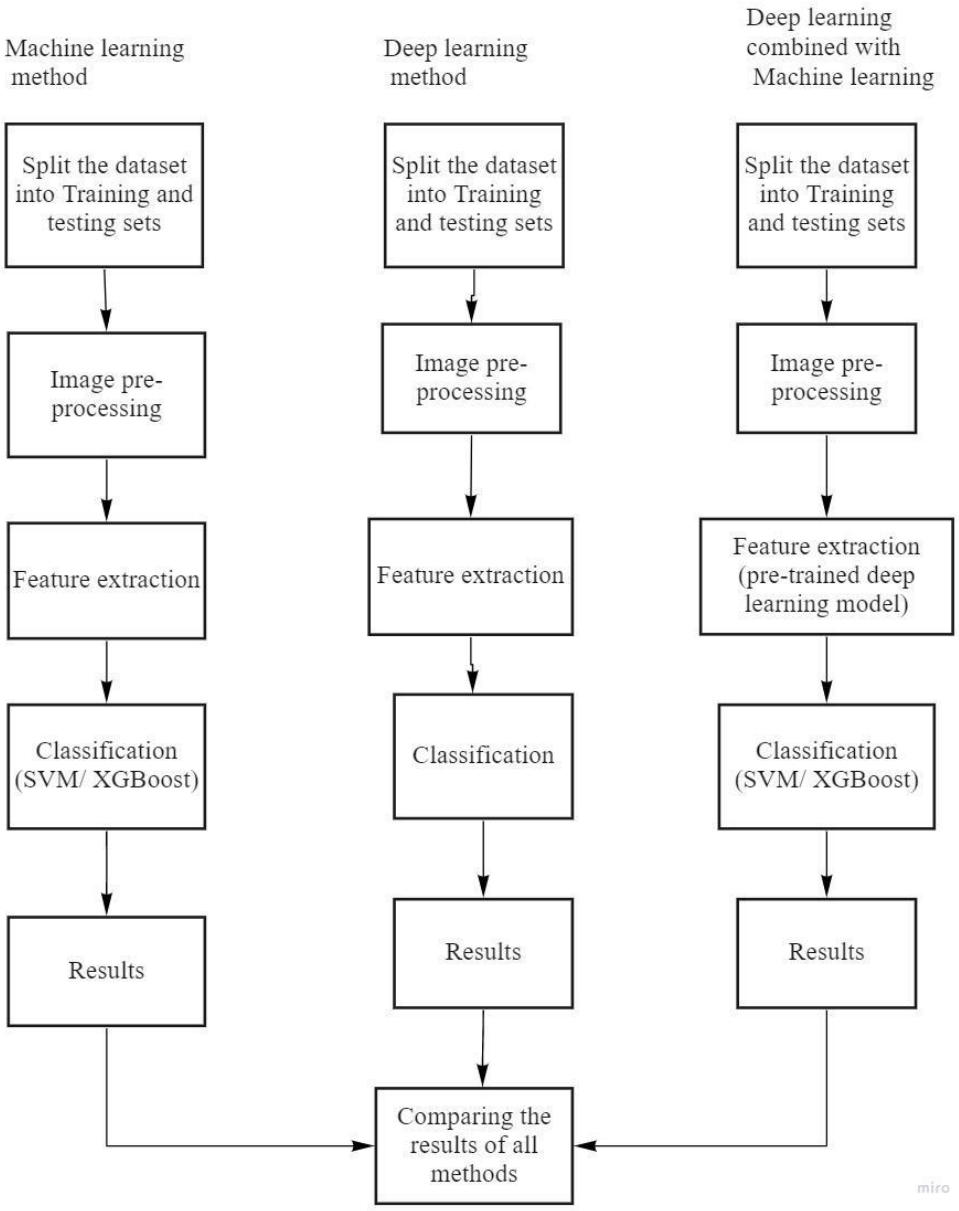

Figure 6.1 Algorithms implementation workflow.

6.2 Hardware specifications

We summarized the hardware used for this study in this section. Using different hardware to perform the same experiment may cause the output to differ. Therefore, the same hardware should be used across all experiments. The outputs investigated in this work are the training time and the testing time, which heavily depend on the hardware used. We used one hardware system in this work, the local machine.

Table 6.1. Hardware specifications of the local computer.

System	**HP ProBook 450 7th Generation**
Processor	Intel Core i7-105101U processor, 1.8 GHz- 4.9GHz frequency. 8MB L3 cache and 4 cores. Intel UHD Graphics 620
Memory	16.00 GB
System type	Windows 10 Pro 64-bit,

6.3 Software Framework

We implemented the entire study using python language with Spyder as the Integrated Development Environment (IDE) [133]. Python contains several scientific and mathematics libraries useful for machine learning, such as Scikit-Learn [134] and TensorFlow [135]. Also, libraries such as NumPy, Matplotlib, OpenCV, and pandas make model implementation and data analysis easy.

Table 6.2. Software specifications.

Operating system	Windows 10 Pro
Language	Python 3.8.0
Integrated development environment (IDE)	Spyder (Anaconda)

6.4 Input data handling

As discussed in Chapter 3, we subdivided the 'bccr-segset' dataset to create the canola and radish datasets, as shown in the following table.

Table 6.3 Dataset details

Canola dataset	Radish dataset
Background - 2387	Background - 799
stage 1 - 527	stage 1 - 782
stage 2 - 360	stage 2 - 780
stage 3 - 1629	stage 3 - 822
stage 4 - 475	stage 4 - 1436
Total = 5378	Total = 4619

Separating the testing dataset from the training dataset

In Chapter 3, we discussed how the image in the dataset was captured. Basically, the images were captured at various angles (45°, 90°, 225°, 270°, 315°, 360°), and this introduces randomness in the dataset and improves generalisation. The same process can be done automatically by introducing augmentation through rotation and scaling. In this work, we did not have to introduce augmentation during training as the images are already diverse enough from the capturing phase. Figures 6.2 to 6.5 show consecutive images from the same class (stage 4) in the dataset. It can be seen that these images are not similar as they are taken at different angles, although they are of the same plant. This is done to all the images. This way, data leakage and overfitting are avoided during cross-validation and training. We then manually removed 10% of images from each class to create our testing set.

Figure 6.2. Canola plant at 123°

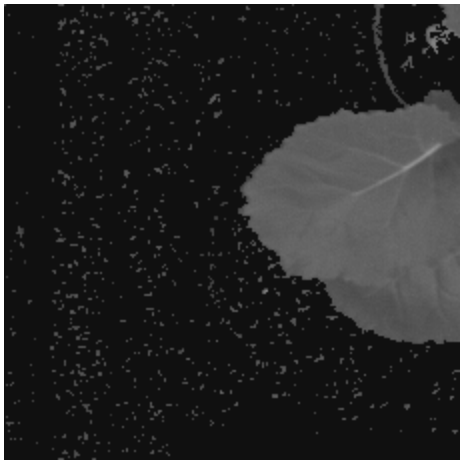

Figure 6.3. Canola plant at 124°

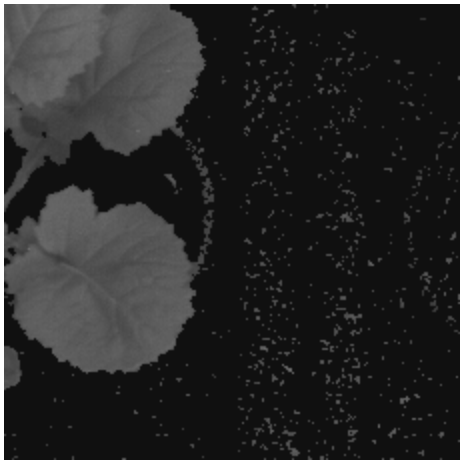

Figure 6.4. Canola plant at 126°

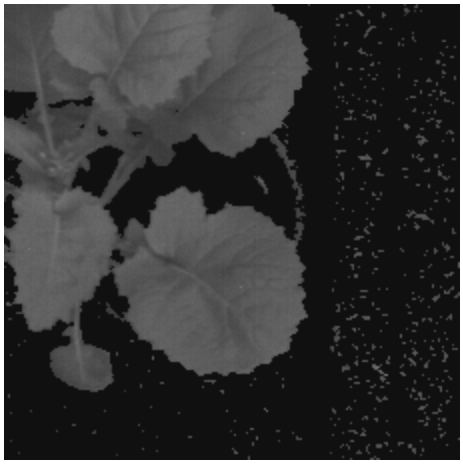

Figure 6.5. Canola plant at 127°

Stratified cross-validation [136] was employed to train all the models with the training dataset. Stratified cross-validation randomly shuffles the training images before splitting them into different folds. We used 3-fold stratified cross-validation in this work, which means the models were trained three times. At each time, a different set was used as a testing set. In other words, a separate testing set is generated at each iteration of the 3-fold cross-validation. First, we removed ten per cent of the training data to create a test set that was not used during training. Then, the test set was used to test the performance of the generated model. A detailed illustration of how we partitioned the datasets is given in Figures 6.6 and Figure 6.7. The testing set in both datasets is 10% of each total dataset.

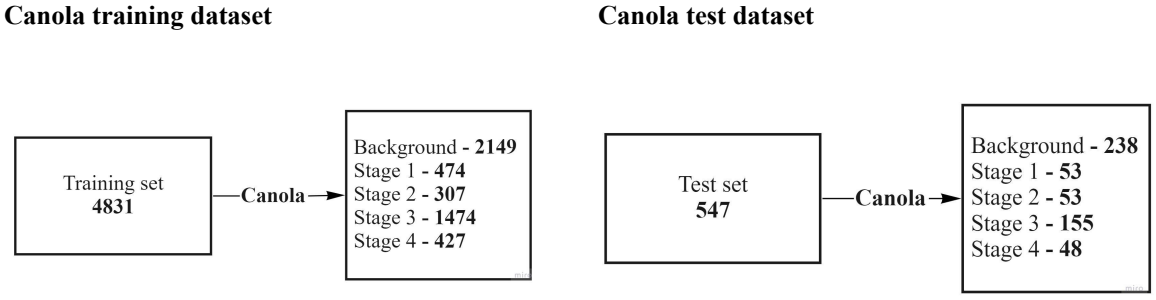

Figure 6.6. Partitioning the "canola" dataset into training and testing datasets of five classes (Stage 1, Stage 2, Stage 3, Stage 4, and Background).

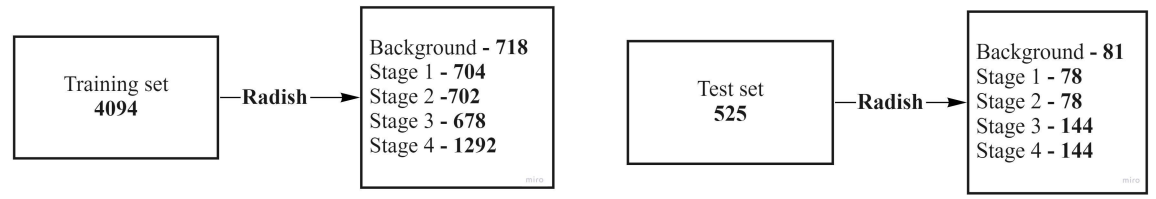

Figure 6.7. Partitioning the "radish" dataset into training and testing datasets of five classes (Stage 1, Stage 2, Stage 3, Stage 4, and Background).

The following table shows the input image sizes used in the different models of this study.

Table 6.4. Input image sizes

Model	Image size
RBF SVM Approach 1	128X128
RBF SVM Approach 2	128X128
ResNet-50	224X224
VGG-19	224X224
VGG-16	224X224
CNN- RBF SVM	224X224

6.5 Implementation of the two machine learning methods

The proposed machine learning algorithms are centered on the assumption that growth classification can be distinguished by analysing plant leaves' images. The leaf texture, shape, and size are different for plants at different growth stages. Texture can be in the form of the visibility of veins on the leaves, pores on the leaves, and the smoothness on the leaves. Our goal is to detect these features and use them to distinguish different growth stages. The workflow of the proposed methods is given in Figure 6.1, and we discussed the steps taken in the algorithm's design in Chapter 4. The implementation of each step is now going to be discussed.

Steps taken

6.5.1 Data pre-processing

- We convert the images to grayscale.
- We resize the images to fit the input size of the model. We use the size of 128×128 as the input size of the proposed method. We selected this input size because it returns all the important information in an image while using less memory.
- We passed the resized and grayscale image dataset to the feature extraction stage directly for the first approach. We passed the resized and grayscale images through two branches for the second approach. One branch (left branch), we do no morphological operation to this branch. In the second branch (right branch),

we applied morphological operators (opening and closing) to a copy of the dataset.

- Morphological operations were applied using a 5×5 structuring element. A contour masked parallel dataset was generated.
- We combined the two datasets (the original image and the masked dataset).
- Then we send the combined dataset to the feature extraction stage.

6.5.2 Feature extraction

This section explains how we extract plant leaf textural and morphological features. The motivation behind selecting Gabor filters is given in Chapter 4. Also, a more detailed explanation of the expressions used in this section is given in Chapter 4. In brief, Gabor filters are excellent in extracting salient features from images. This is shown in various works [27], [77]–[83], [137]–[139]. A discrete Gabor filter function is given in equation 4.1. The "Uncertainty Principle" governs this function to give accurate time-frequency location [140]. Gabor filters are robust and resistant to changing brightness and contrast in images. Because of their ability to model the visual cortex cells, we have employed the Gabor filter in this study to extract the key plant features to enhance plant growth estimation.

By varying the λ, θ, ψ, σ and γ parameters of the Gabor expression 4.1, a bank of Gabor filters can be generated. We tried different parameter combinations on a few images from each class to find the best performing combination on feature extraction. Table 6.5 below shows the Gabor parameters we selected after trying different combinations.

Table 6.5. Gabor filter parameters

Filter Number	**Kernel size**	θ	Λ	φ	σ	γ
1	9×9	$\frac{\pi}{4}$	$\frac{\pi}{10}$	0	0.5	0.5
2	9×9	$\frac{\pi}{4}$	$\frac{\pi}{8}$	0	0.5	0.5
3	9×9	$\frac{\pi}{2}$	$\frac{\pi}{10}$	0	0.5	0.5
4	9×9	$\frac{\pi}{2}$	$\frac{\pi}{8}$	0	0.5	0.5

Figures 6.8 to 6.11 shows the orientations of the generated filters. The four filters were applied to all the images in the dataset to generate features. The output features of a random image (Figure 6.12) filtered using the filters is shown in Figures 6.13 to 6.16. From these images, it can be seen that different filters highlight various characteristics while suppressing others. Notably, the texture and shape are highlighted with Gabor filters. These are the features that are sent to the machine learning model. In the second approach, we also applied the same Gabor filters to the contour mask image dataset to focus more on highlighting leaf edges. Sobel edge detector using the scikit-learn library was also applied as an additional filter to enhance morphological feature extraction. The Gabor filters were applied to the datasets in both proposed machine learning models using Python 3.8. Morphological filters were also applied to the image dataset using python 3.8. The source codes for all these models are published on GitHub.

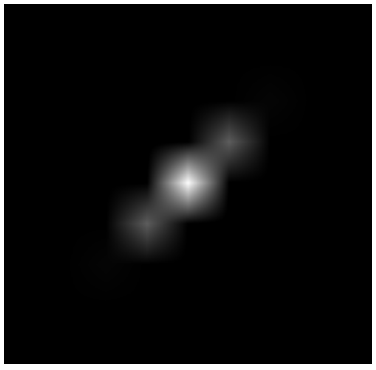

Figure 6.8. Gabor Filter 1 orientation.

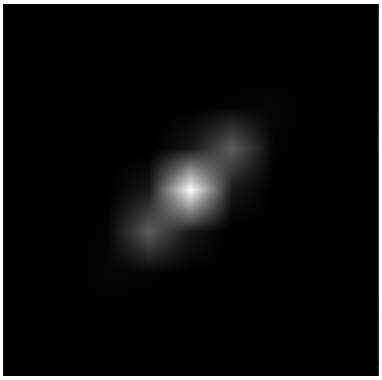

Figure 6.9. Gabor Filter 2 orientation.

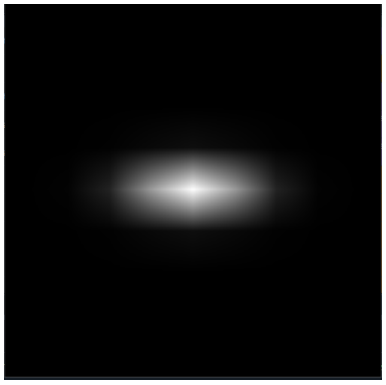

Figure 6.10. Gabor Filter 3 orientation.

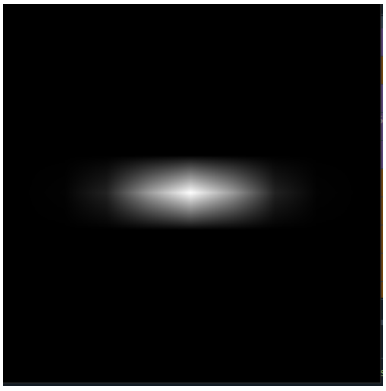

Figure 6.11. Gabor Filter 4 orientation.

Figure 6.12. Original image.

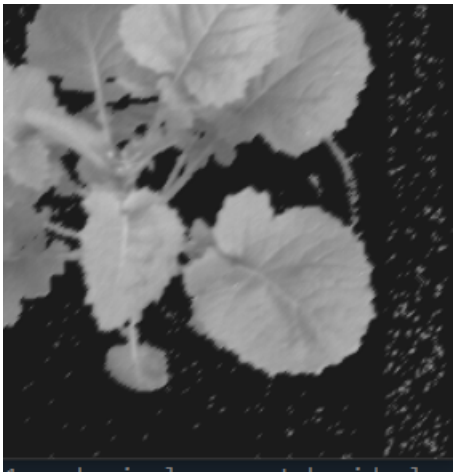

Figure 6.13. filtered image by filter 1.

Figure 6.14. Image features by filter 2.

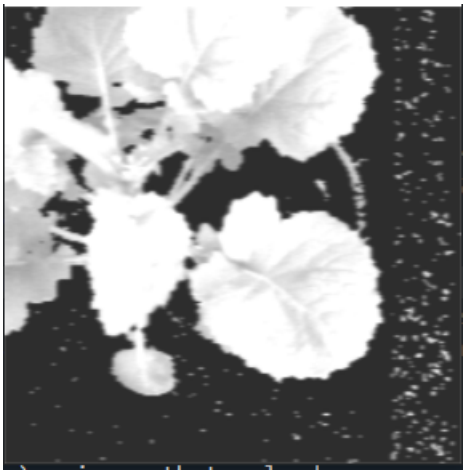

Figure 6.15. Image features by filter 3.

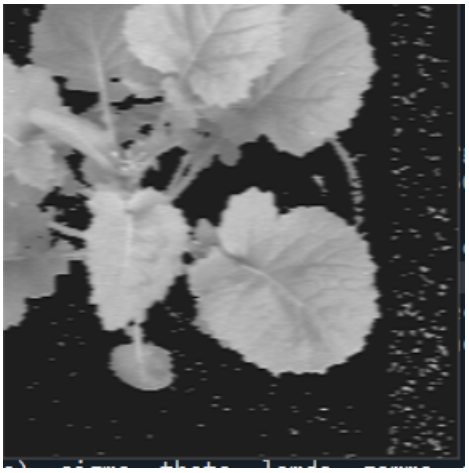

Figure 6.16. Image features by filter 4.

6.5.3 Classification

We selected the SVM radial basis function (RBF) nonlinear kernel for classification. To separate non-linear data, SVM, through its kernels, maps the data into a high dimensional space to make the separation possible. Then, data reformulation is done with the SVM kernels to map the data into a new space explicitly. Figure 6.17 shows how the mapping is done.

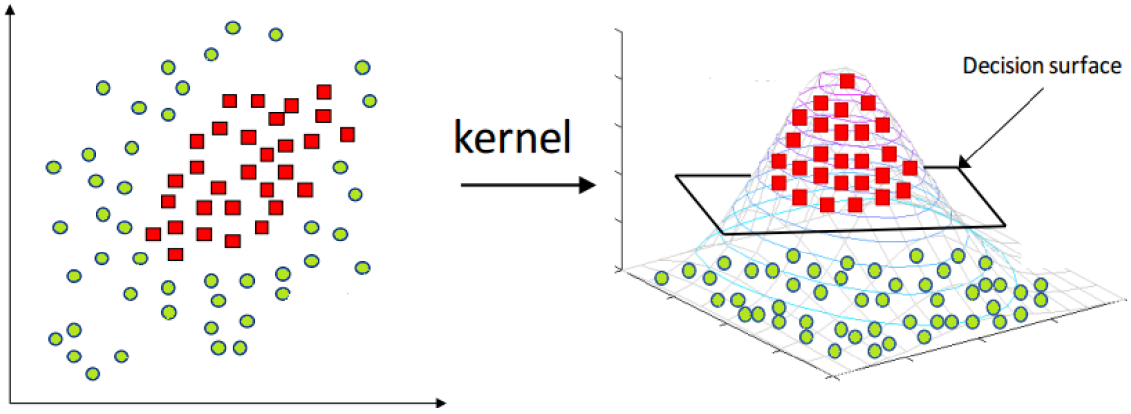

Figure 6.17. Data transformation into a higher space for classification using an SVM kernel function [141].

The RBF kernel function is given as below [141].

$$K(X, Y) = exp(-\gamma|X - Y|2)\, \gamma > 0 \quad (6.2)$$

Where: $K(X, Y)$ is the high dimensional space where the input data is transformed to. X and Y are the space vectors. γ is the gamma parameter used to set the kernel spread (size). If this value is overstated, the exponential behaviour will nearly become linear, making the high-dimensional projection linear and losing its effectiveness. When the gamma parameter value is set small, the decision boundary curve becomes relatively low, resulting in the broader decision region. When the gamma value is set high, the decision boundary curve also becomes high, resulting in islands of decision boundaries surrounding similar data points.

The *C* parameter does not affect the kernel chosen. The *C* parameter is a misclassification penalty parameter. When *C* is large, the SVM classifier takes the best performing hyper-plane with a smaller margin between the classes. When C is small, the classifier takes a hyper-plane that has a bigger margin between classes.

Using cross-validation and a small custom dataset, we test the linear and non-linear kernels to decide the best SVM kernel. The linear SVM produced the least results, followed by the sigmoid kernel. The RBF and polynomial kernels had better results that could be improved by hyperparameter tuning; as a result, these two kernels were considered. The same small custom dataset was used to search for the best hyperparameters. RandomisedSearchCV method [142] was used to search the best C, gamma parameters and degree. Parameter tuning is computationally expensive and time-consuming hence why a small dataset that represents the dataset accurately was used to do a grid search. Also, defining the correct scale to perform the search is important to speed the process. The first step was to normalise the feature data frame to eliminate redundancy. We then defined the scale to search for the *C* parameter as {C= 2^{-2}, 2^{0}, 2^{2}, 2^{4}, 2^{6}}We defined the scale for the gamma parameter as {γ = 0, 2^{-2}, 2^{5}, 2^{-10}, 2^{-15}}A Hyperparameter search was carried out for the RBF kernel.

The highest classification score was obtained with *C=32* and $\gamma = 0.00001$. Polynomial kernel attained the second-best classification score with *C = 30* and $\gamma = 0.00001$ and *degree = 2.*

Hyperparameter was also done for XGBoost. Table 6.7, shows the range and parameters we randomly searched. Table 6.8 shows the optimum parameter combination found.

Table 6.7. The search space for XGBoost hyperparameters

Hyperparameter	Range
Learning rate	[0.25, 0.3, 0.4, 0.5, 0.6, 0.7]
Max_depth	[5 , 6, 8, 10, 12]
Min_child_weight	[1, 3. 5, 7]
gamma	[0.3, 0.4, 0.5, 0.6, 0.7]
Colsample_bytree	[0.3, 0.4, 0.5, 0.6, 0.7]

Table 6.8. Optimum hyperparameters for XGBoost.

Hyperparameter	Range
Learning rate	[0.5]
Max_depth	[8]
Min_child_weight	[7]
gamma	[0.6]
Colsample_bytree	[0.7]

6.5.4 Cross-validation

For classification, we used stratified cross-validation [136]. As discussed in the previous chapters, stratified cross-validation makes sure that each class is represented equally in each fold. This is important when there is an imbalance of data across the input classes. Our dataset is imbalanced. Therefore, we use 3-fold stratified cross-validation. Cross-validation prevents overfitting as the training data is split into multiple training and testing sets, allowing evaluation, and tuning of the model and leaving the testing data unseen for prediction testing. Also, in stratified cross-validation, the training data is shuffled and partitioned into respective folds. In our case, 3-folds. This means the model was trained three times. A distinct set is used as a testing set at each training iteration, while the other two sets are used as the training sets. Training is done until each of the three folds is used as a testing set in these three different iterations.

After cross-validation, a model is generated. This model is tested on the testing set, the images that the model never saw—section 6.4, Figure 6.6, and Figure 6.7 shows how the training and testing set was created. The model's performance on the testing set is observed using several metrics explained in Chapter 7.

6.6. Implementation of the deep learning approach

For implementing the deep learning approach using transfer learning, we used the Keras library to download the pre-trained models, VGG-16 [125], VGG-19 [128], and ResNet-50 [126], together with their weights. Based on the literature survey conducted in

chapter 2, the transfer models chosen in these models have produced satisfactory results across many fields.

6.6.1 Data pre-processing for transfer learning

- We converted all the images to grayscale.
- We resized images to fit the model's input size. For VGG-16, we used an image size of 224 x 224. For VGG-19, a size of 224 x 224, and for ResNet-50, we used a size of 224 x 224.

6.6.2 Fine-tuning and optimisation

As discussed in the previous chapters, all the transfer learning models employed in this study were originally trained on the ImageNet dataset, comprising over 1.28 million images belonging to over 1000 categories [127]. That means the models have an excellent knowledge of distinguishing general features. We have maintained these models' convolutional layers (hidden layers) with their respective weights to transfer this knowledge to the feature extraction problem in this current study. First, we changed the output layer from 1000 classes to 5 classes (background, stage 1, stage 2, stage 3, and stage 4) corresponding to our dataset's exact number of classes. Then, we added a SoftMax layer and a fully connected layer to the output layer of all three deep learning models. These three layers are the only layers trained with the dataset. The other layers are loaded with their ImageNet weights. So, we optimised these last layers.

Optimisation

This section discusses the optimisation method used to tune and choose hyperparameters of the deep learning models.

When deciding which hyperparameters to tune, we focus on those that significantly impact the convergence and generalisation of the model. The values for the respective hyperparameters are chosen after multiple preliminary trials to identify hyperparameter combinations that output the best results in terms of convergence and validation accuracy. Optimizer choice, learning rate, training seed, and the number of epochs are the hyperparameters that impact convergence most. Adam is the optimiser used in CNN, and this is because of its excellent adaptive estimations of lower-order moments compared to SGD and other optimizers. In this work, we are using pre-trained network architectures; therefore, we are not changing the convolutional layer parameters of these models. Instead, we focus on optimising the fully connected layers (the output layers). When looking for the best hyper-parameter values, we optimised across three random training seeds to maximise the robustness of the results. Notably, random seeds are only employed to verify that the best hyperparameters are selected. We utilised iterative grid search to find the optimal hyperparameters. Cross-entropy is the loss function used in these models, and to build a probability distribution, the SoftMax layer normalizes the output values. The preference for cross-entropy over mean squared error (MSE) was purely empirical; literature [103] supports the usage of cross-entropy, mainly when using SoftMax functions. The batch size was also optimised. Table 6.10 to 6.12, shows the optimum hyperparameters obtained after the search for each of the three deep learning models (VGG16, VGG19, and ResNet50). Table 6.9 shows the search space which was chosen for the three models.

Table 6.9. The search space was used for the three models.

Hyperparameter	Range
Learning rate	[0.01, 0.001 0.0001]
Batch size	[10, 30, 60, 100]
Dropout	[0, 0.1, 0.3, 0.5]
Number of Epochs	[10, 30, 50]

Table 6.10. The optimum hyperparameters were obtained through a grid search for VGG16.

Hyperparameter	Range
Learning rate	[0.001]
Batch size	[60]
Dropout	[0.3]
Number of Epochs	[10]

Table 6.11. The optimum hyperparameters were obtained through a grid search (VGG19).

Hyperparameter	Range
Learning rate	[0.001]
Batch size	[60]
Dropout	[0.3]
Number of Epochs	[10]

Table 6.12. The optimum hyperparameters were obtained through a grid search for ResNet50.

Hyperparameter	Range
Learning rate	[0.001]
Batch size	[60]
Dropout	[0.0]
Number of Epochs	[50]

The optimum grid search parameters found were tested on the testing data. For VGG19, an accuracy of 99.3% was achieved on the testing data, an accuracy of 97.0 % on the testing data for VGG-16 and ResNet50 attained 78.80 % accuracy on the testing data.

6.6.3 Feature extraction using CNN models

- As discussed in the previous chapter (chapter 5), pre-trained models with their original weights are used in this work. The convolutional layers of a pre-trained network perform feature extraction.
- The first step is to resize the images to fit the input sizes of these respective models.
- Secondly, we normalised the pixel data to range between 0 and 1 by dividing every pixel value by 255. This is crucial as it reduces the computational requirement of the feature extraction process.
- The third step is to encode the labels into a numerical value, in other words, background as 0, stage 1 as 1, stage 2 as 2, stage 3 as 3, and stage 4 as 4. This is to normalise the class labels and make them machine-readable.
- We pass the resized and normalised image data to the fine-tuned CNN model and the data passes through the dense layers where the respective model kernel filter does convolution. In VGG, a kernel of size (3 x 3) is used to perform feature extraction.
- After the data has passed the convolutional layers, it goes to a flattening layer where the feature data is converted to one-dimensional data, which fits the input of the fully connected layers.

  Figure 5.8. shows how data is passed from the first convolutional layer to the output layer. ReLU activation function is used in all the dense layers to stop negative values from moving forward in the network.

6.6.4 CNN classification

For classification, the SoftMax activation function with five classes (Stage 1, Stage 2, Stage 3, Stage 4, and Background) is used in the fully connected layer to make output predictions. For performance analysis, we investigate the accuracy, confusion matrix, F1 score, recall, and precision (presented in the next chapter). Finally, further performance analysis is done by investigating the training and execution times.

6.7 CNN as Feature extractor and a machine learning model as a classification model

The deep learning models, VGG-19, VGG-16, and ResNet-50 are used as feature extractors by freezing the convolutional layers with their ImageNet weights and removing the fully connected layers by setting the top to False. A feature extractor is constructed this way and this feature extractor is applied to the datasets. Finally, the features extracted are passed to SVM or XGBoost and the performance metrics (accuracy, confusion matrices, precision, recall and F1-score) are examined on the models.

Figure 5.9 visualises how these models are created.

6.8 Conclusion

This chapter discussed the steps taken to implement all the approaches discussed in Chapter 4 and Chapter 5 (two proposed machine learning approaches and two deep learning approaches). We started by describing the hardware and software used in this study. We also discussed how the dataset was partitioned into the training and testing sets

to avoid data leakage. Then, we discussed each step mentioned in the methodology of each respective algorithm in detail. The implementation of the feature extraction through Gabor filters used in the traditional machine learning approaches is discussed in detail. We also visualised the Gabor filter kernels developed in this study. The implementation of the classification method through the use of RBF kernel SVM and 3-fold Stratified cross-validation is also discussed. We have also discussed hyperparameter tuning to find the optimum parameter values for the deep learning approach. Then, the implementation of feature extraction through a pre-trained model is given. The implementation of the classification method is also discussed. The last approach combines a pre-trained model as a feature extraction with a machine learning classification algorithm such as XGBoost and SVM. These three approaches are tested on two different datasets (canola and radish), the results and comparison are presented in the next chapter.

Chapter 7

Results and performance analysis

This chapter discusses the results obtained from all the experiments carried out in this study in relation to the project's objectives. Side-by-side experiments on the same datasets and same hardware setup are done to verify the best-performing approach. Different metrics supported by literature such as confusion matrices, F1-score, recall, and precision are used for performance analysis. Finally, comparison analysis is done using the achieved results.

7.1 Introduction

In this chapter, we have trained the traditional machine learning models, the three deep learning models, and the combined deep neural network model with a machine learning algorithm to evaluate their performance. The results obtained on both datasets and corresponding discussions are given in this chapter. Firstly, we provide a detailed description of the performance metrics used to evaluate all the models. Then, we

visualise the features obtained using the different feature extractors. Finally, a comparison of the results obtained with similar works is presented.

7.2 Performance evaluation metrics

In this project, we are trying to solve a classification problem. Therefore, metrics that are not prone to bias where there is a class imbalance in the dataset must be used. Machine learning algorithms assume an even data distribution within classes; therefore, if there is a class imbalance in the input dataset, the model will predict the majority class more because it has more information about that class. Therefore, accuracy as the only metric is not enough to evaluate classification models since a model can accurately predict the majority class but less on the minority class. Therefore, we used confusion matrices, F1-score, recall, and precision as the performance metrics.

A confusion matrix records the predictions of each class to calculate the classification accuracy. In other words, it is an $N \times N$ matrix where N represents the number of target classes. Table 7.1 illustrates how a confusion matrix compares the target values to the model's predicted values. It comprises the following variables:

TP is true positive: We predicted a growth stage class which is the targeted class.

FP is false positive: We predicted a class that is not the targeted class.

FN is false negative: We predicted that a particular class is not targeted, but the class is the targeted class.

TN is true negative samples: We predicted that a certain class is not the targeted class and the class truly is not the targeted class.

Table 7.1. A confusion matrix for binary classification.

Class	**Positive**	**Negative**
Positive	TP	FP
Negative	FN	TN

Overall Accuracy

Overall accuracy indicates the correctness of a classifier, and it is obtained using equation 7.1. We used both validation accuracy and test accuracy as performance metrics. The use of both validation and test accuracies can detect overfitting. For this reason, we decided to use both in this study.

$$accuracy = \frac{tp+tn}{tp+fp+fn+tn} \tag{7.1}$$

$$recall = \frac{tp}{tp+fn} \tag{7.2}$$

Focusing on a single class triumphs in many applications where samples in some classes are fewer than others. The experimental setup follows: one class is taken from a collection of classes as a special attention class. The remaining classes are left alone (multiclass classification) or merged (binary classification). The choice measure based on the positive class is:

$$precision = \frac{tp}{tp+fp} \tag{7.3}$$

$$F - measure = \frac{2}{\frac{1}{recall}+\frac{1}{precision}} \tag{7.4}$$

Precision, recall and F-measure distinguish the accurate classification of labels within distinct classes. They focus on a single class. Recall is a function of correctly classified (true positive) labels and wrongly classified labels (false negatives). A mathematical representation of recall is given in Equation 7.2. Recall is a useful metric when false negative is greater than false positive. In our case, recall is significant because we don't want to misclassify the growth stages of particular plants and mix them with plants of different growth stages, thereby leading to an inadequate supply of nutrients or an oversupply of nutrients.

Precision is instrumental when false positives are of greater concern than false negatives. True positives and examples that have been misclassified as positives (false positives) are used to calculate the precision. Equation 7.3 shows how precision can be calculated.

In practice, increasing the precision of our model decreases the recall and vice versa. Therefore, the F1-score or F1-measure is a single number that encapsulates both trends, as shown in Equation 7.4. F1-score is maximum when precision is equal to recall.

7.3 Analysis of the feature extraction methods

One of our goals is to evaluate how well Gabor features compare with other feature extraction approaches such as the transfer learning models (VGG-16, VGG-19, and ResNet50). We used principal component analysis (PCA) and the t-Distributed Stochastic Neighbour Embedding (t-SNE) method to generate feature data visualisation for qualitative analysis. t-SNE reduces data dimensionality to make closer data points in the original high dimensional space closer to each other in the two-dimensional space.

Figures 7.1 to 7.8 show the t-SNE visualisation of the different feature extraction methods, the Gabor filters, VGG-16, VGG-19, and ResNet-50. Each colour indicates a growth stage class or the background class. It is noteworthy that the t-SNE dimensionality reduction procedure is fully unsupervised and the labels are simply used to colour the nodes at plotting.

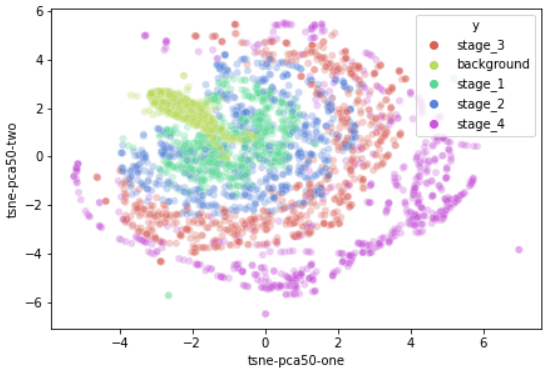

Figure 7.1. Gabor filter features from the canola training dataset.

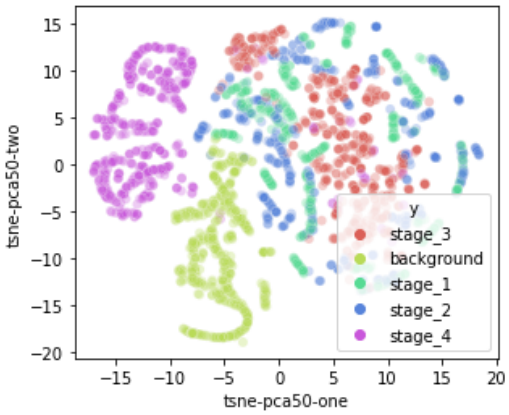

Figure 7.2. VGG-16 features from the

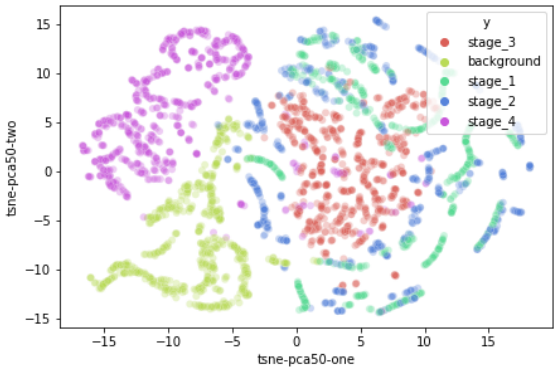

Figure 7.3. VGG-19 features from the canola training dataset.

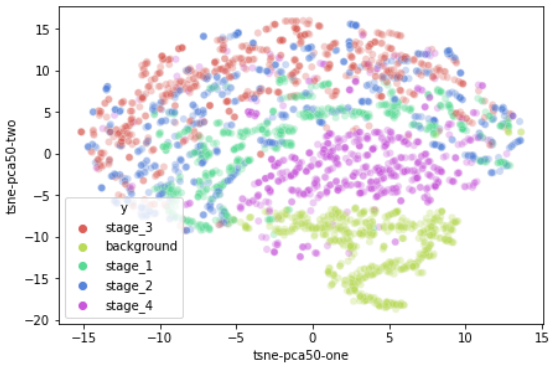

Figure 7.4. ResNet-50 features from the canola training dataset.

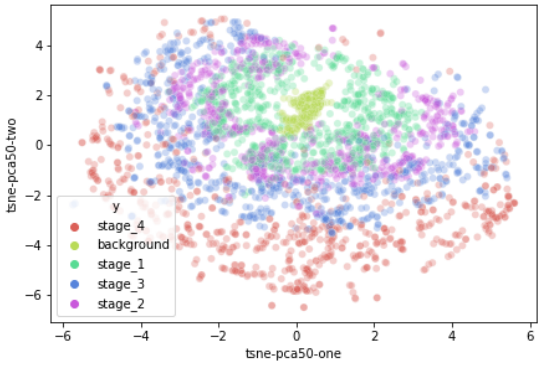

Figure 7.5. Gabor filter features from the radish training dataset.

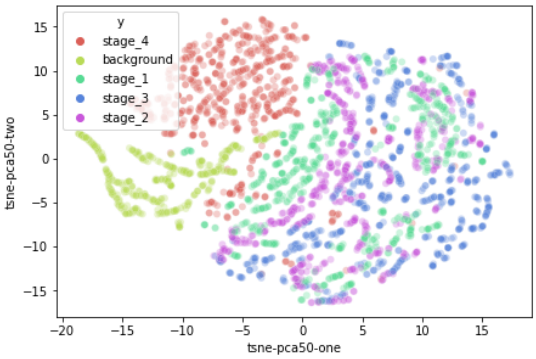

Figure 7.6. VGG-16 features from the radish training dataset.

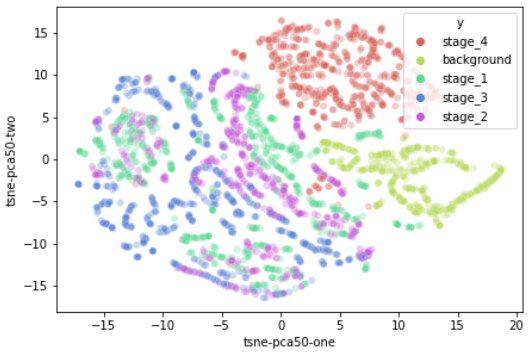

Figure 7.7. VGG-19 features from the radish training dataset.

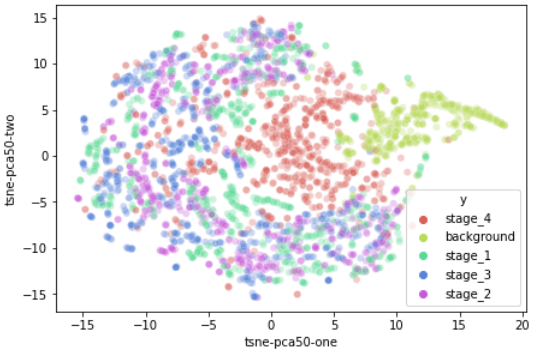

Figure 7.8. ResNet-50 features from the radish training dataset.

Figures 7.1 to 7.8 show that the VGG convolutional layers projected the plant image pixels into a more separable space. As a result, there is more clustering of data samples in the VGG models (as can be seen from the t-SNE visualisation). The Gabor filter features space that is also separable. ResNet-50 performed the least with less clustering and more overlapping of samples. Overly, we can conclude that the feature extraction of all the approaches can extract features that can represent the growth stage classes. However, overlapping can still be observed in all the approaches. Therefore, unsupervised machine learning algorithms such as K-Means will weakly classify the extracted features in these models. However, a non-linear classification algorithm such as XGBoost and RBF kernel SVM can identify patterns in these features and generate a higher classification accuracy.

7.4 Results

We trained and evaluated all of the previously presented models using the two datasets, the canola, and the radish datasets, to assess the performance of the two techniques (machine learning and deep learning) in estimating plant growth stages. Parallel experiments are conducted utilizing the transfer learning approach with models VGG-19, VGG-16, and ResNet-50. The output of each model is recorded and compared. The performance metrics discussed in section 7.2, accuracy, precision, recall, F1-score, and confusion matrices, are used to evaluate the models' performance.

7.4.1 Comparison of the canola dataset

Classification accuracies on the canola dataset

The results of the performance metrics are recorded on the partitioned canola dataset. To avoid overfitting, we employed 3-fold stratified cross-validation. With stratified cross-validation, the dataset is shuffled randomly. At each iteration, one of the three folds is selected as the testing set, while the remaining folds are considered the training set. The process is repeated until all folds are utilised as a testing set. This validation process used only the training data presented in Figures 6.6 and 6.7. Accuracies per each fold are recorded and the average of these fold accuracies is taken as the final accuracy. The testing set is used after cross-validation to obtain the prediction accuracy of the particular model. For the testing data, the prediction accuracy is recorded. Table 7.2 shows the model's classification accuracy on the testing data. The deep learning models were loaded with their original ImageNet weights. These models were fine-tuned using the transfer learning method discussed in Chapter 6, and a categorical cross-entropy loss function is used during the training of the deep learning models. We selected the Adam optimizer because of its higher performance after experimental trials using different optimizers. The learning rate was 0.001. A batch size of 60 for VGG-16 and 30 for ResNet-50.

The traditional machine learning approach had two methods developed and tested; the first method, Method 1 presented in Figure 4.1, consisted of Gabor filters and XGBoost or SVM. XGBoost, with its optimum hyperparameters shown in Table 6.8. This method obtained an average classification accuracy of 91.1% with XGBoost and 90.1% with SVM.

The second machine learning method, Method 2 presented in Figure 4.4, consisted of Gabor filters, morphological operators and XGBoost or SVM. An accuracy of 92.3% was obtained with method 2 was combined with XGBoost, and 96.7% was obtained with RBF SVM kernel. The deep learning models had slightly higher classification accuracies than

the four developed machine learning methods. ResNet-50 obtained the least accuracy of 76.3% from all the tested models on this dataset, as shown in Table 7.2.

The third approach of using a pretrained deep learning model as a feature extractor and machine learning as a classifier achieved the highest classification accuracies. A VGG-19 network and RBF SVM combination obtained the highest accuracy of 98.4% among all the models. It is noteworthy, the hyperparameters of all the trained models were randomly searched before training the models. Table 7.2 presents the classification results of all models.

Table 7.2. Classification accuracies of all the models on the canola test dataset.

Method	Fold 1	Fold 2	Fold 3	Average
Method 1: Gabor Filter & XGBoost	91.3%	91.2%	91.1%	91.2%
Method 1: Gabor Filter & RBF SVM	90.0%	90.0%	90.3%	90.1%
Method 2: Gabor Filter & XGBoost	92.6%	92.0%	92.3%	92.3%
Method 2: Gabor Filter & RBF SVM	96.8%	96.6%	96.6%	96.7%
VGG-19	97.7%	97.8%	97.8%	97.8%
VGG-16	95.4%	95.7%	95.6%	95.6%
ResNet-50	76.2%	76.2%	76.3%	76.3%
VGG19 & XGBoost	97.8%	96.7%	97.8%	97.3%
VGG19 & RBF SVM	**98.4%**	**98.4%**	**98.4%**	**98.4%**
VGG16 & XGBoost	96.2%	95.6%	94.0%	95.3%
VGG16 & RBF SVM	92.9%	91.8%	93.4%	92.7%
ResNet-50& XGBoost	90.2%	90.2%	90.1%	90.2%
ResNet-50& RBF SVM	95.0%	95.1%	95.1%	95.1%

Confusion matrices provide more insight into the classifiers' performance. Table 7.3, summarizes the confusion matrices for all models evaluated on the canola test set. One thing that stands out from the confusion matrices is the misclassification of Stage 1 and Stage 2. As indicated in Table 7.3, there are more false positives (FP) and false negatives (FN) between Stage 1 and Stage 2 in most models. This is owing to the plant's leaf features being so identical. As a result, the task of estimating growth stages is complex, requiring a high-performing model to identify and classify the growth stages. Although the developed machine learning models performed satisfactorily, the transfer learning models exhibited fewer FN and FP across all classes. The third approach of using a deep neural network model as a feature extractor and a machine learning algorithm for classification had a significantly lower misclassification rate than the other approaches, particularly the VGG19 and the RFB SVM models. The rationale for using this approach as explained in detail in section 5.7 is its low computation requirement as the total number of parameters is reduced as the fully connected layers (Classification layers) are replaced by a traditional machine learning classification algorithm such as SVM or XGBoost. Precision, recall, and F1-score can be calculated from the confusion matrices using the equations in section 7.2.

Table 7.3: The average Confusion matrices on the canola test dataset.

Method	Classes	Background	Stage 1	Stage 2	Stage 3	Stage 4
Method 1: Gabor Filter & XGBoost						
	Background	80	0	0	0	0
	Stage 1	5	12	0	0	0
	Stage 2	0	5	13	0	0
	Stage 3	0	0	4	47	1
	Stage 4	0	0	0	1	15
Method 1: Gabor Filter & RBF SVM						
	Background	80	0	0	0	0
	Stage 1	6	11	0	0	0
	Stage 2	0	6	12	0	0

	Stage 3	0	0	4	47	1
	Stage 4	0	0	0	1	15
Method 2: Gabor Filter & XGBoost						
	Background	79	1	0	0	0
	Stage 1	2	13	2	0	0
	Stage 2	0	4	13	1	0
	Stage 3	0	0	2	49	1
	Stage 4	0	0	0	1	15
Method 2: Gabor Filter & RBF SVM						
	Background	80	0	0	0	0
	Stage 1	1	15	1	0	0
	Stage 2	0	2	15	1	0
	Stage 3	0	0	1	51	0
	Stage 4	0	0	0	0	16
VGG-19	**Background**	80	0	0	0	0
	Stage 1	1	15	1	0	0
	Stage 2	0	2	15	0	0
	Stage 3	0	0	0	52	0
	Stage 4	0	0	0	0	16
VGG-16	**Background**	79	1	0	0	0
	Stage 1	1	14	2	0	0
	Stage 2	0	3	14	0	0
	Stage 3	0	0	1	51	0
	Stage 4	0	0	0	0	16
ResNet50	**Background**	79	1	0	0	0
	Stage 1	0	10	4	2	1
	Stage 2	0	3	13	2	0
	Stage 3	0	1	2	36	13
	Stage 4	1	1	0	12	2

VGG-19 & XGBoost						
	Background	80	0	0	0	0
	Stage 1	0	16	1	0	0
	Stage 2	1	3	14	0	0
	Stage 3	0	0	0	52	0
	Stage 4	0	0	0	0	16
VGG-19 & RBF SVM						
	Background	79	1	0	0	0
	Stage 1	0	17	0	0	0
	Stage 2	0	1	16	0	0
	Stage 3	0	0	0	52	0
	Stage 4	0	0	0	0	16
VGG-16 & XGBoost						
	Background	80	0	0	0	0
	Stage 1	1	14	2	0	0
	Stage 2	0	2	16	0	0
	Stage 3	0	0	0	51	0
	Stage 4	0	0	0	1	15
VGG-16 & RBF SVM						
	Background	79	1	0	0	0
	Stage 1	3	11	4	0	0
	Stage 2	0	1	13	3	0
	Stage 3	0	0	0	52	0
	Stage 4	1	0	0	0	15
ResNet-50 & XGBoost						
	Background	80	0	0	0	0
	Stage 1	3	7	4	0	3
	Stage 2	0	2	11	5	0
	Stage 3	1	0	0	51	0
	Stage 4	0	0	0	0	16

ResNet-50 & RBF SVM	**Background**	80	0	0	0	0
	Stage 1	2	13	1	0	1
	Stage 2	0	0	15	3	0
	Stage 3	1	0	0	51	0
	Stage 4	0	1	0	0	15

Table 7.4 below shows the average precision-recall and F1-score after three folds of all the trained models. From the table, the developed four machine learning approaches showed the capability to learn the key features which can bring about classification. In general, the model (Gabor and XGBoost) performed well, except for the stage 1 class, where it obtained an average precision, recall, and f1 score of 73%. The machine learning model with the highest performance is (Method 2 of Gabor filters, morphological operations, and RBF SVM). This model performed excellently in terms of precision, recall, and F1 score in all classes. The least effective model is the ResNet-50 with F1-score values less than 50 percent.

Table 7.4. Average precision, recall, F1-score after 3-fold cross-validation on the canola test dataset.

Method	Class	Precision	Recall	F1-score	Support
Method 1: Gabor Filter & XGBoost					
	Background	1.00	0.94	0.97	80
	Stage 1	0.71	0.71	0.71	17
	Stage 2	0.72	0.76	0.74	18
	Stage 3	0.90	0.98	0.94	52
	Stage 4	0.94	0.94	0.94	16
Method 1: Gabor & SVM					
	Background	0.93	1.00	0.96	80
	Stage 1	0.65	0.65	0.65	17
	Stage 2	0.75	0.67	0.71	18
	Stage 3	0.98	0.90	0.94	52
	Stage 4	0.94	0.94	0.94	16
Method 2: Gabor Filter & XGBoost					
	Background	0.99	0.98	0.98	80
	Stage 1	0.76	0.72	0.74	17
	Stage 2	0.72	0.76	0.74	18
	Stage 3	0.94	0.96	0.95	52
	Stage 4	0.94	0.94	0.94	16
Method 2: Gabor Filter & RBF SVM					
	Background	1.00	0.99	0.99	80
	Stage 1	0.88	0.88	0.88	17
	Stage 2	0.83	0.88	0.86	18
	Stage 3	0.98	0.98	0.98	52
	Stage 4	1.00	1.00	1.00	16

VGG-19	**Background**	1.00	0.99	0.99	80
	Stage 1	0.88	0.88	0.88	17
	Stage 2	0.88	0.94	0.91	17
	Stage 3	1.00	1.00	1.00	52
	Stage 4	1.00	1.00	1.00	16
VGG-16	**Background**	0.99	0.99	0.99	80
	Stage 1	0.82	0.78	0.80	17
	Stage 2	0.82	0.82	0.82	17
	Stage 3	0.98	1.00	0.99	52
	Stage 4	1.00	1.00	1.00	16
ResNet50	**Background**	0.99	0.99	0.99	80
	Stage 1	0.59	0.63	0.61	17
	Stage 2	0.71	0.67	0.69	17
	Stage 3	0.69	0.69	0.69	52
	Stage 4	0.13	0.13	0.13	16
VGG-19 & XGBoost					
	Background	1.00	0.99	0.99	80
	Stage 1	0.94	0.84	0.89	17
	Stage 2	0.78	0.93	0.85	18
	Stage 3	1.00	1.00	1.00	52
	Stage 4	1.00	1.00	1.00	16
VGG-19 & RBF SVM					
	Background	1.00	0.99	0.99	80
	Stage 1	0.89	1.00	0.94	17
	Stage 2	1.00	0.89	0.94	17
	Stage 3	0.98	1.00	0.99	52
	Stage 4	1.00	1.00	1.00	16

VGG-16 & XGBoost					
	Background	0.98	1.00	0.99	80
	Stage 1	0.88	0.82	0.85	17
	Stage 2	0.89	0.89	0.89	18
	Stage 3	0.98	0.98	0.98	51
	Stage 4	1.00	0.94	0.97	16
VGG-16 & RBF SVM					
	Background	0.95	1.00	0.98	80
	Stage 1	0.92	0.61	0.73	18
	Stage 2	0.76	0.76	0.76	17
	Stage 3	0.95	1.00	0.97	52
	Stage 4	1.00	0.94	0.97	16
ResNet-50 & XGBoost					
	Background	0.95	1.00	0.98	80
	Stage 1	0.78	0.41	0.54	17
	Stage 2	0.73	0.61	0.67	18
	Stage 3	0.91	0.98	0.94	52
	Stage 4	0.84	1.00	0.91	16
ResNet-50 & RBF SVM					
	Background	1.00	0.96	0.98	80
	Stage 1	0.76	0.93	0.84	17
	Stage 2	0.83	0.94	0.88	18
	Stage 3	0.98	0.94	0.96	52
	Stage 4	0.94	0.94	0.94	16

7.4.2. Comparison on the radish dataset.

We repeated the same experimental setup on the radish plant dataset, which comprised 9095 images. The same metrics used to evaluate the models on the canola dataset are also

used here. The results are presented in tables in this section. Additionally, stratified cross-validation is employed to minimize overfitting and class representation bias within each fold. Finally, an unseen by the model test set is used to test the model's performance. These models are evaluated on this dataset.

Table 7.5 shows the accuracies of all the trained models on the radish dataset. The best performing machine learning model (Method 2 of Gabor filters, morphological operators, and RBF kernel SVM) achieved an average accuracy of 91.4% across all three folds. The deep learning models achieved accuracy slightly higher than the machine learning models, with VGG-19 and VGG-16 obtaining 98.4% and 98.2%, respectively. The ResNet-50 model obtained the lowest overall score of 71.6%. From the models developed by combining deep learning and machine learning, the combined VGG19 and RBF SVM obtained the highest accuracy of 98.4%, matching that of the VGG-19 model.

Table 7.5. Classification accuracies on the radish test dataset.

Method	Fold 1	Fold 2	Fold 3	Average
Method 1: Gabor Filter & XGBoost	89.6%	89.5%	89.6%	89.7%
Method 1: Gabor Filter & RBF SVM	87.3%	87.3%	87.6%	87.4%
Method 2: Gabor Filter & XGBoost	90.3%	90.4%	90.4%	90.3%
Method 2: Gabor Filter & RBF SVM	92.02%	91.5%	90.8%	91.4%
VGG-19	98.8%	98.8%	97.7%	98.4%
VGG-16	98.4%	98.2%	98.2%	98.2%
ResNet-50	75.4%	71.4%	68.0%	71.6%
VGG19 & XGBoost	96.6%	94.9%	95.4%	95.6%
VGG19 & RBF SVM	**98.9%**	**97.8%**	**98.3%**	**98.4%**
VGG16 & XGBoost	95.4%	94.9%	94.9%	95.1%
VGG16 & RBF SVM	97.7%	98.9%	97.7%	98.1%
ResNet-50& XGBoost	91.4%	86.3%	86.3%	88%
ResNet-50& RBF SVM	92.0%	96.0%	91.4%	93.1%

Table 7.6 below presents the confusion matrices of all the models. False negatives and False positives are also observed between stage 2 and stage 3. The VGG models performed well with less FN and FP. The VGG-19 and RBF SVM models performed best with very few misclassifications.

Table: 7.6. The average Confusion matrices of all the models on the radish test dataset.

Method	Classes	Background	Stage 1	Stage 2	Stage 3	Stage 4
Method 1: Gabor Filter & XGBoost						
	Background	27	0	0	0	0
	Stage 1	1	23	2	0	0
	Stage 2	0	3	20	3	0
	Stage 3	0	1	3	42	2
	Stage 4	0	0	0	3	45
Method 1: Gabor Filter & RBF SVM						
	Background	26	1	0	0	0
	Stage 1	1	25	0	0	0
	Stage 2	0	1	16	9	0
	Stage 3	0	0	8	39	1
	Stage 4	0	1	0	0	47
Method 2: Gabor Filter & XGBoost						
	Background	27	1	0	0	0
	Stage 1	0	26	0	0	0
	Stage 2	0	2	19	5	0
	Stage 3	0	2	1	43	2
	Stage 4	0	2	0	3	43
Method 2: Gabor Filter & RBF SVM						
	Background	27	0	0	0	0
	Stage 1	0	25	1	0	0
	Stage 2	0	1	19	5	1

	Stage 3	0	1	0	45	2
	Stage 4	0	1	0	3	44
VGG-19	**Background**	27	0	0	0	0
	Stage 1	0	26	0	0	0
	Stage 2	0	1	24	1	0
	Stage 3	0	0	0	48	0
	Stage 4	0	0	0	0	48
VGG-16	**Background**	27	0	0	0	0
	Stage 1	0	26	0	0	0
	Stage 2	0	1	25	0	0
	Stage 3	0	0	0	48	0
	Stage 4	0	1	0	1	46
ResNet50	**Background**	27	0	0	0	0
	Stage 1	1	25	0	0	0
	Stage 2	0	1	25	0	0
	Stage 3	0	4	14	22	8
	Stage 4	0	9	5	5	26
VGG-19 & XGBoost						
	Background	27	0	0	0	0
	Stage 1	0	26	0	0	0
	Stage 2	0	1	23	2	0
	Stage 3	1	0	1	45	1
	Stage 4	0	0	0	0	48
VGG-19 & RBF SVM						
	Background	27	0	0	0	0
	Stage 1	0	26	0	0	0
	Stage 2	0	0	25	1	0
	Stage 3	0	0	0	48	0
	Stage 4	0	0	0	1	47
VGG-16 & XGBoost						
	Background	26	1	0	0	0
	Stage 1	1	25	0	0	0

	Stage 2	0	2	21	0	3
	Stage 3	0	0	0	47	1
	Stage 4	0	0	0	0	48
VGG-16 & RBF SVM						
	Background	27	0	0	0	0
	Stage 1	0	26	0	0	0
	Stage 2	0	2	24	0	0
	Stage 3	0	0	0	48	0
	Stage 4	0	0	0	0	48
ResNet-50 & XGBoost						
	Background	26	1	0	0	0
	Stage 1	1	24	1	0	0
	Stage 2	0	7	12	7	0
	Stage 3	0	2	0	46	0
	Stage 4	0	1	0	4	43
ResNet-50 & RBF SVM						
	Background	27	0	0	0	0
	Stage 1	0	26	0	0	0
	Stage 2	0	5	21	0	0
	Stage 3	0	0	2	46	0
	Stage 4	0	0	0	0	48

Table 7.7. presents the precision, recall, and F1-score of all the models. These confusion matrices correspond with the confusion matrices in Table 7.9. The models (VGG-16, VGG-19 and VGG-19, and RBF SVM) with the highest overall accuracy have the highest precision, recall, and F1-score. The VGG-19 and RBF SVM had an F1 score of close to 1 in all the classes.

Table 7.7. Average precision, recall, F1-score after 3-fold cross-validation on the radish test dataset.

Model	Classes	Precision	Recall	F1-score	Support
Method 1: Gabor Filter & RBF SVM					
	Background	1.00	0.96	0.98	27
	Stage 1	0.88	0.85	0.87	26
	Stage 2	0.77	0.80	0.78	26
	Stage 3	0.88	0.88	0.88	48
	Stage 4	0.94	0.96	0.95	48
Method 1: Gabor Filter & XGBoost					
	Background	0.96	0.96	0.96	27
	Stage 1	0.96	0.89	0.93	26
	Stage 2	0.62	0.67	0.64	26
	Stage 3	0.81	0.81	0.81	48
	Stage 4	0.98	0.98	0.98	48
Method 2: Gabor Filter & XGBoost					
	Background	1.00	1.00	1.00	27
	Stage 1	1.0	0.81	0.90	26
	Stage 2	0.73	0.95	0.83	26
	Stage 3	0.90	0.84	0.87	48
	Stage 4	0.90	0.96	0.92	48
Method 2: Gabor Filter & RBF SVM					
	Background	1.00	1.00	1.00	27
	Stage 1	0.89	0.96	0.93	26
	Stage 2	0.95	0.73	0.83	26
	Stage 3	0.85	0.94	0.89	48

	Stage 4	0.94	0.92	0.93	48
VGG-19	**Background**	1.00	1.00	1.00	27
	Stage 1	0.96	1.00	0.98	26
	Stage 2	1.00	0.88	0.94	26
	Stage 3	0.94	1.00	0.97	48
	Stage 4	1.00	0.98	0.99	48
VGG-16	**Background**	1.00	1.00	1.00	27
	Stage 1	0.96	1.00	0.98	26
	Stage 2	1.00	0.88	0.94	26
	Stage 3	0.96	1.00	0.98	48
	Stage 4	1.00	1.00	1.00	48
ResNet50	**Background**	1.00	1.00	1.00	27
	Stage 1	0.71	0.77	0.74	26
	Stage 2	0.80	0.62	0.70	26
	Stage 3	0.85	0.58	0.69	48
	Stage 4	0.61	0.85	0.71	48
VGG-19 & XGBoost					
	Background	1.00	1.00	1.00	27
	Stage 1	0.81	1.00	0.90	26
	Stage 2	0.95	0.81	0.88	26
	Stage 3	0.98	0.94	0.96	48
	Stage 4	1.00	1.00	1.00	48
VGG-19 & RBF SVM					
	Background	1.00	1.00	1.00	27
	Stage 1	0.96	1.00	0.98	26
	Stage 2	0.96	0.92	0.94	26
	Stage 3	0.98	0.98	0.98	48
	Stage 4	1.00	1.00	1.00	48

VGG-16 & XGBoost					
	Background	0.89	0.89	0.89	27
	Stage 1	0.89	0.92	0.91	26
	Stage 2	1.00	0.96	0.98	26
	Stage 3	0.96	0.98	0.97	48
	Stage 4	0.98	0.96	0.97	48
VGG-16 & RBF SVM					
	Background	1.00	1.00	1.00	27
	Stage 1	0.93	1.00	0.96	26
	Stage 2	1.00	0.92	0.96	26
	Stage 3	1.00	1.00	1.00	48
	Stage 4	1.00	1.00	1.00	48
ResNet-50 & XGBoost					
	Background	0.96	0.96	0.96	27
	Stage 1	0.69	0.92	0.79	26
	Stage 2	0.92	0.46	0.62	26
	Stage 3	0.81	0.96	0.88	48
	Stage 4	1.00	0.90	0.95	48
ResNet-50 & RBF SVM					
	Background	1.00	1.00	1.00	27
	Stage 1	0.89	0.92	0.91	26
	Stage 2	0.81	0.81	0.81	26
	Stage 3	0.88	0.96	0.92	48
	Stage 4	1.00	0.90	0.95	48

Due to the close performance of the best-performing models, we decided to include the comparison based on the execution time. The performance of the VGG19, the machine learning model (Method 2, Gabor filters, morphological operators and RBF SVM), and the model based on the combination of machine learning and deep learning (VGG-19 and RBF SVM) were all satisfactory and close. Since this is a model for real-time prediction,

the processing time plays a huge part.

We implemented all the models presented in this work on the same hardware and software system.

7.4.3 Training time

Table 7.8, summarizes the total time required to train each model. All models were trained and tested without the usage of a GPU. The hyperparameters searched in Chapter 6 were used in all the models. In brief, the VGG models were trained using a batch size of 60 and ten epochs, and in the ResNet-50 model, a batch size of 60 and 20 epochs was used. The training time of all the models comprises the time required to resize the images, convert them to grayscale for the machine learning models, pre-processing processes of generating contour masks, feature extraction, and training using three-fold stratified cross-validation.

It is important to note that we did not search for the best system to perform this experiment therefore the results are completely based on our system.

The time it takes to train the models in this study is longer because no GPU was employed. The machine learning models, particularly Method 2 (Gabor filters, morphological operators, and RBF SVM), took longer to train. The pre-processing of morphological processing done to the data initially is the reason for this lengthy amount of time. Also, the contour masked images are concatenated with the original images, doubling the total image number, significantly impacting the training period. The transfer learning models had the second-longest training time because of the deep neural networks that perform feature extraction and pass the image data from the first layer to the last layer. This is seen in detail when the training and testing time for VGG-19 and VGG-16 are compared. Because VGG-19 has a more extensive network than VGG16, more time is taken to train VGG19. The third approach (the combination of machine learning and deep learning) requires the least amount of training time. This is because the machine

learning algorithm (SVM or XGBoost) substitutes the fully connected layers of the transfer learning models. As a result, the network is reduced, and time is reduced.

Table 7.8. The total period for training all the models locally with the two datasets.

Method	**Canola dataset Total Training time in (hrs: min: sec)**	**Radish dataset Total Training time in (hrs: min: sec)**
ResNet-50& RBF SVM	00:05:24	00:06:51
VGG16 & RBF SVM	00:07:30	00:06:51
VGG16 & XGBoost	00:07:59	00:06:45
VGG19 & RBF SVM	00:08:29	00:06:03
VGG19 & XGBoost	00:08:34	00:09:05
ResNet-50& XGBoost	00:10:48	00:03:27
Method 1: Gabor Filter & RBF SVM	01:10:07	00:59:25
Method 1: Gabor Filter & XGBoost	01:19:26	01:07:19
ResNet-50	01:28:15	01:14:47
VGG-16	01:45:47	01:29:38
VGG-19	02:12:55	01:52:38
Method 2: Gabor Filter & RBF SVM	02:20:10	01:58:47
Method 2: Gabor Filter & XGBoost	02:21:59	02:00:20

7.4.4 Predicting time

The average testing time per set and the average testing time per image of the models on the two datasets is presented in Table 7.9 and 7.10. The testing time per image is obtained by dividing the total testing time by the total number of images in each dataset. From Table 7.9, the machine learning models took the least amount of time of 0.04 seconds to predict a single testing image. The average testing time of the best performing model (VGG-19 & RBF SVM) is 0.07 seconds. On the other hand, the VGG-19 model's testing time per image is 1.56 seconds. The predicting time is achieved without using any GPUs to accelerate the testing process.

Table 7.9. Testing times of all the models on the canola dataset (547 images).

Method	**Canola dataset Average Testing time/Test Set (hrs: min: sec)**	**Canola dataset Average Testing time/image (seconds)**
Method 1: Gabor Filter & RBF SVM	00:00:22	0.04
Method 1: Gabor Filter & XGBoost	00:00:28	0.05
VGG16 & RBF SVM	00:00:36	0.07
VGG19 & RBF SVM	00:00:43	0.08
VGG16 & XGBoost	00:00:44	0.08
ResNet-50& RBF SVM	00:00:50	0.09
VGG19 & XGBoost	00:00:50	0.09
ResNet-50& XGBoost	00:00:53	0.10
Method 2: Gabor Filter & RBF SVM	00:00:59	0.12
Method 2: Gabor Filter & XGBoost	00:02:52	0.30
ResNet-50	00:11:14	1.23
VGG-19	00:14:12	1.56
VGG-16	00:12:45	1.40

7.4.5 Execution time on radish dataset

The time it takes to predict a single image in the radish dataset is also tabulated in Tables 7.9 and 7.10. From the tables, a link between the execution time on the radish dataset and the canola dataset. The testing time per image is almost similar in both datasets. The machine learning models that did not use morphological operators (Method 1) had the fastest execution time of 0.04 seconds in both datasets. The combined deep learning and machine learning models had the second-fastest execution time, followed by the machine learning models with morphological operators (Method 2). Transfer learning had the longest execution time. The time to predict a single image is obtained by dividing the average time per set by the total number of images in the dataset. Also, the results in Tables 7.9 and 7.10 show that SVM classifies faster than XGBoost, although the difference is not much.

Table 7.10. Testing time of all the models on the radish test dataset (525 images).

Method	**Radish dataset Average Testing time/Test Set (hrs: min: sec)**	**Radish dataset Average Testing time/image (seconds)**
Method 1: Gabor Filter & RBF SVM	00:00:21	0.04
Method 1: Gabor Filter & XGBoost	00:00:27	0.05
VGG19 & RBF SVM	00:00:32	0.06
VGG16 & RBF SVM	00:00:37	0.07
VGG16 & XGBoost	00:00:43	0.08
ResNet-50& RBF SVM	00:00:41	0.08
VGG19 & XGBoost	00:00:48	0.09
ResNet-50& XGBoost	00:00:56	0.11
Method 2: Gabor Filter & RBF SVM	00:00:59	0.11
Method 2: Gabor Filter & XGBoost	00:02:44	0.31

VGG-16	00:12:10	1.39
ResNet-50	00:10:42	1.22
VGG-19	00:13:35	1.55

7.5 Comparison with similar previous studies

The results of this study are compared to results obtained in the previous studies of plant identification and classification based on machine learning, deep learning, and a combination of both methods and are presented in Table 7.11. The aim of comparing these results to previous works is to see how our proposed approach competes with the current state of the art methods on classification problems. From the table, it can be seen that machine learning has been employed more in plant identification and classification than deep learning. The results also demonstrated higher accuracy level scores in deep learning studies. The use of a deep learning model as a feature extractor and a machine learning model for classification has not been explored much, especially in agriculture. Overall, from the reviewed works together with the results from this study, the utilization of traditional machine learning earned the highest plant classification of 96.7%.

On the other hand, deep learning achieved the highest plant classification accuracy of 99.74%. The combined deep learning and machine learning models achieved the highest classification of 99.29%. The complexity of handcrafting a feature extraction method limited the accuracy of the machine learning approach in the reviewed works. In some deep learning studies, the lack of big datasets restricted the accuracy rate. Nevertheless, the models (combining deep learning and machine learning) attained the highest average accuracy rate from the reviewed work, including ours. Since the models use a pre-trained network for feature extraction, they do not require a huge dataset and converge quicker. Using a machine-learning algorithm to perform classification, the model becomes more effective and efficient.

Table 7.11. Comparison of similar studies with our results.

Study	Method	Research type	Accuracy percentage
Murata et al., 2019 [53]	NDVI image processing and CNN (rice growth stage classification)	Deep learning	89.3%
Loresco et al., 2018 [61]	Colour space feature extraction (CIELab) and KNN (lettuce growth stage identification)	Machine learning	90%
Concepcion et al., 2020 [62]	Using super-pixels and multi-fold transformation on morphological characteristics. (Lettuce plant growth phases)	Machine learning	88.89% on stage 1, 86.67% on stage 2 and 79.63% on stage 3.
Kheirkhah et al., 2019 [73]	SVM and LBP extracted morphological features, SVM and GIST features	Machine learning	74.92% and 83.88%
Rasti et al., 2021 [72]	CNN and transfer learning to classify two growth stages of barley and wheat crops	Deep learning	99.74%
Liu et al., 2020 [51]	Continuous wavelets transform (CWT), successive projection algorithm (SPA), and support vector machines (SVM) on spectral data (Potato growth stages)	Machine learning	97.3%
Alejandrino et al., 2020 [69]	Unsupervised machine learning algorithm based on morphological features obtained using colour-based super-pixel and multi-fold transformation. (Lettuce growth stage classification)	Machine learning	91%
Aldabbagh et al., 2020 [70]	Mask R-CNN model based on (ReNet-50 and ResNet-101) (Chilli plant growth stage classification)	Deep learning	96%, 97%
Teimouri et al., 2018 [59]	Deep convolutional neural network for growth stage estimation of Weed species	Deep learning	70%
Kusumaningrum et al., 2020 [54]	Convolutional neural network based on LANDSAT images	Deep learning	83.4%

Ramadhani et al., 2020 [55]	Mapping growth stages using a combination of visual remote sensing technologies (SENTINEL-1, SENTINEL-2, MOD13Q1)	Machine learning	83.27%
Kulkarni., et al., 2013 [15]	By combining shape, vein, colour, and texture with Zernike Moments	Machine learning	93.82%
Kadir et al., 2013 [143]	Probabilistic Neural Networks (PNNs) on features (shape, venation, colour, and texture).	Machine learning	93.75%
Ehsanirad., et al. 2010 [144]	a) Probabilistic Neural Network with PCA on Geometric and morphological features.	Machine learning	91%
	b) Binary Decision Trees and Support Vector Machines (SVM) with geometric and morphological features,	Machine learning	96%
	c) Support vector machines (SVM) and Fourier moments (FM) on geometric and morphological features,	Machine learning	62%
Garcia. et al., 2020 [145]	Transfer learning as a feature extractor and machine learning for classification (SVM)	A combination of both machine learning and deep learning.	99.29%

Ramaneswaran. et al., 2021 [146]	Image feature extraction is handled by Inception v3, while classification is handled by XGBoost. (Acute lymphoblastic leukaemia based on microscopic pictures of white blood cells)	A combination of both machine learning and deep learning.	98.6%
Situala et al., 2020, [147]	Feature extraction using a pre-trained deep learning model and classification using support vector machines (Breast cancer classification using histopathological images)	A combination of both machine learning and deep learning.	92.2%
Le et al., 2018 [12]	Local binary pattern operators for morphological feature extraction and support vector machines for plant discrimination	Machine learning	91.85%
Le et al., 2020 [148]	a) Local binary pattern operators, morphological operators for morphological feature extraction, and support vector machines for plant and weed discrimination	Machine learning	90.94%
	b) VGG-16	Deep learning	91.55%
	c) VGG-19	Deep learning	89.55%
	d) ResNet-50	Deep learning	89.73%
	e) Inception-V3	Deep learning	90.87%

Our research study	f) Method 2: Gabor filters, morphological operators for texture and morphological feature extraction and XGBoost for classification.	Machine learning	92.3% (canola), 90.3% (radish)
	g) Method 2: Gabor filters, morphological operators for texture and morphological feature extraction and SVM for classification.	Machine learning	96.7% (canola), 91.4% (radish)
	h) VGG-16	Deep learning	95.3% (canola), 98.4% (radish)
	i) VGG-19	Deep learning	97.6% (canola), 98.2% (radish)
	j) VGG-16 feature extractor and XGBoost for classification	A combination of both machine learning and deep learning.	95.3% (canola), 95.1% (radish)
	k) VGG-16 feature extractor and SVM for classification	A combination of both machine learning and deep learning.	92.7% (canola), 98.1% (radish)
	l) VGG-19 feature extractor and XGBoost for classification	A combination of both machine learning and deep learning.	95.6% (canola), 97.3% (radish)

	m) VGG-19 feature extractor and SVM for classification	A combination of both Machine Learning and Deep Learning.	98.4% (canola), 98.4% (radish)

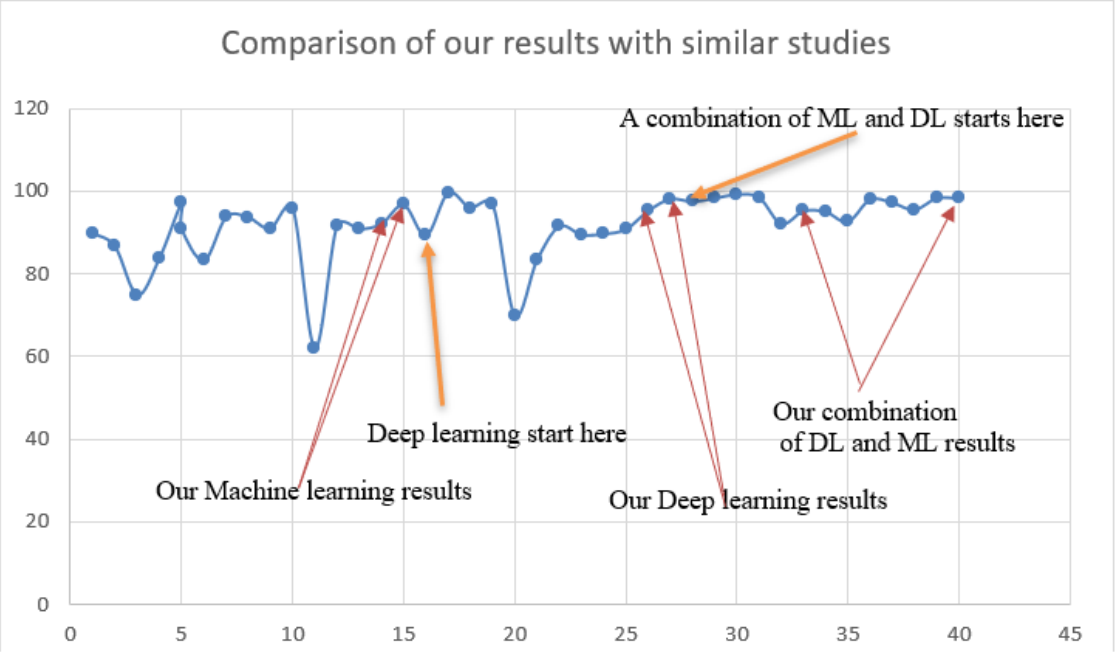

Figure 7.9: Comparison of the results obtained from the various approaches used in similar studies. The vertical axis represents the accuracy percentages, and the horizontal axis the number of research studies reviewed (with traditional machine learning studies results first followed by deep learning then the third approach of using a feature extractor based on deep learning and a classifier based on traditional machine learning). The arrows point to the results obtained by our study.

As presented in Figure 7.9, the results from machine learning are not uniform. Some of the manually designed feature extraction methods produced low results, while some produced high. On the other hand, deep learning outcomes were high and more uniform than machine learning. This demonstrates the capability of deep learning models to learn the important features automatically. The last part of the line graph contains the results of the combined deep learning and machine learning models. These models have the most uniform and high accuracy. As seen in the figure, the results of our study across all three categories of models compare well with the best results obtained from similar studies in terms of accuracy.

7.6 Conclusion

This chapter presented the findings from the study's side-by-side comparison experiments. We examined each model's accuracy, F1-score, precision, recall, and confusion matrices on two distinct datasets. Additionally, we also recorded and analysed the execution times of the tested models for each dataset. By utilizing the performance metrics and the execution time as the basis of comparison, the developed machine learning models competed with the deep learning models in every aspect. The best performing machine learning model (Gabor filters, morphological operators, and RBF SVM) obtained an accuracy of 96.7%, while the best performing transfer learning model (VGG-19) obtained an accuracy of 98.4%. In terms of the other metrics (precision, recall, and F1-score) derived from the confusion matrices, all the models suffered a few false positives and false negative values at the stage 2 class. The third approach investigated in

this study of using deep learning models as a feature extractor and machine learning algorithms such as XGBoost and SVM for classification obtained the best results across the board, with the best and highest results coming from combining VGG-19 and RBF SVM. The overall accuracy of 98.4% is achieved in this model. Besides accuracy, this approach also achieved the highest precision, recall, and F1-score.

In terms of training time, the machine learning models with morphological operators took longer to train due to many pre-processing processes done to the data. Also, four Gabor filters were applied to each image in the dataset. The transfer learning models had the second-longest training time because of the deep neural networks that perform automatic feature extraction plus the fully connected layers that perform the predictions. The machine learning models that do not use morphological operators obtained the fastest execution time. The approach of combining a deep neural network model and a machine learning algorithm required the second least amount of time of 0.05 seconds to classify a single image. However, deep learning models (VGG-19, VGG-16, and ResNet-50) required more testing time than all the models. Therefore, while the deep learning models eliminate the need to handcraft a feature extraction method by automatically learning the key features, deep learning models require high computational power. Also, a vast amount of data should be supplied for deep learning models to flourish. This leads to a longer execution time if no specific hardware is used. Special hardware is needed to speed the training and testing of deep learning models. Machine learning converges quicker and does not need special hardware to perform highly. Combining the two approaches, machine learning and deep learning to develop one approach eliminates the

individual weaknesses such as taking long periods to execute and harnessing the unique strengths such as employing automatic feature extraction. A faster and high-performing model that is not computationally expensive is optimal for real-time operations. Therefore, the model developed due to combining deep learning models and a machine learning algorithm is the best.

Comparing the results obtained in this study with results from other studies showed the effectiveness of the models developed in this study.

Chapter 8

Conclusion

This chapter reviews the key findings and the conclusions drawn from this study. Future research work is also discussed.

8.1 Introduction

As discussed in the first chapter of this dissertation, the primary goal was to develop a machine learning model that can estimate the growth stages of broadleaf plants grown in hydroponics. This chapter presents a detailed examination of how our objectives were met. The key findings are presented together with the implications and future research work is also suggested.

Chapter 1 introduces the dissertation, discusses the reason for the study, articulates the problem statement and study objectives, and concludes by outlining the dissertation's structure. Chapter 1 provides an overview of the dissertation, including an explanation of the rationale for the investigation, a definition of the problem, and a list of study objectives. Chapter 2 begins with a brief discussion of leaf anatomy and taxonomy, essential in understanding growth stage classification through leaf recognition. It then investigates some challenges confronting computer vision systems in agricultural applications. Some of those are environmental, and some result from image acquisition

circumstances such as camera shaking. Chapter 2 also presents a comprehensive survey and evaluation of image-based plant segmentation approaches. A literature examination of the current state-of-the-art methods for plant growth stage classification and plant classification, in general, has also been done, demonstrating that current methods have limitations, necessitating the need for an efficient method. Chapter 3 discusses the dataset, the capturing of the images, and all the preprocessing done to it. Chapter 4 proposes the development of a plant growth estimator using traditional machine learning. Chapter 5 discusses the development of a plant growth estimator using deep learning. It also suggests a method of using a pre-trained model as a feature extractor and then concatenating it with a machine learning algorithm for classification. The implementation of the three proposed methods is outlined in Chapter 6. A discussion of the results obtained from the three approaches and the performance of these approaches is given in Chapter 7. Finally, Chapter 8 concludes the dissertation and also suggests future work.

8.2 Achievement of research objectives

The research objectives were:

- Do a literature survey of the existing feature extraction methods.
- To develop a model to estimate four plant growth stages (stage 1, stage 2, stage 3, and stage 4) using the traditional machine learning approach of handcrafting the feature extraction part.
- To develop the growth stage estimation model using the deep learning approach.
- To develop a growth estimation model using a deep learning model as a feature extractor and a machine learning algorithm as a classifier.
- Compare and analyse the three approaches, machine learning, deep learning, and the concatenated model (deep learning as a feature extractor and machine learning as a classifier) based on performance metrics, which are overall classification accuracy, confusion matrices, F1-score, recall, precision, and execution time.

- Evaluate and discuss the results obtained from the two approaches, machine learning and the deep learning approach.

Models using the three approaches were developed. Step-by-step development of the feature extractor manually was presented. Also, the employment of the deep learning approach through transfer learning to do automatic feature extraction is explained. How machine learning and deep learning can be combined to come up with a model to solve our research problem was also presented. The results of these models were recorded and compared using various metrics such as classification accuracy, confusion matrices, F-1 score, recall, and precision which investigates performance at the class level. Execution time as a performance metric was employed in this study. These results were also compared with results obtained in similar studies. The similar studies reviewed did not mention their run time in both training and testing. Therefore, the comparison of the execution time was limited to the results obtained from the approaches investigated in this study. Table 8.1 presents a summary of results obtained in this study together with the running time of the respective models.

Therefore, the objectives of this study were met as the models were able to estimate the growth stages on two datasets. Furthermore, all the models were developed and run on the same hardware and software systems; therefore, the comparison of execution time is valid.

Table 8.1. Summary of the results from all the models in solving the research problem.

Model	Accuracy on canola dataset	Accuracy on radish dataset	Training time canola, (hrs: min:sec)	Training time radish (hrs: min:sec)	Classification time/ image (s) Canola	Classification time/ image (s) Radish
Method 1: Gabor Filter & RBF SVM	90.1%	87.4%	01:10:07	00:59:25	0.04	0.04
Method 1: Gabor Filter & XGBoost	91.1%	89.7%	01:19:26	01:07:19	0.05	0.05
VGG16 & RBF SVM	92.7%	98.1%	00:07:30	00:06:51	0.07	0.07
VGG19 & RBF SVM	**98.4%**	**98.4%**	**00:08:29**	**00:06:03**	**0.08**	**0.06**
VGG16 & XGBoost	95.3%	95.1%	00:07:59	00:06:45	0.08	0.08
ResNet-50& RBF SVM	94.7%	93.1%	00:05:24	00:06:51	0.09	0.08
VGG19 & XGBoost	97.3%	95.6%	00:08:34	00:09:05	0.09	0.09
ResNet-50& XGBoost	88.7%	88%	00:10:48	00:03:27	0.10	0.11
Method 2: Gabor Filter, morphological operators & XGBoost	92.3%	90.1%	02:21:59	02:00:20	0.30	0.31
Method 2: Gabor Filter, morphological operators & SVM	96.8%	91.04%	02:20:10	01:58:47	0.12	0.11

ResNet-50	76.0%	71.6%	01:28:15	01:14:47	1.23	1.22
VGG-16	95.3%	98.2%	01:45:47	01:29:38	1.40	1.39
VGG-19	97.6%	98.4%	02:12:55	01:52:38	1.56	1.55

8.3 Conclusion on the achievement of objectives

In conclusion, the proposed machine learning methods extracted key features representing each growth stage from the images, as shown in Table 8.1. SVM seemed to be the best classification algorithm, as higher accuracy was attained over XGBoost. The machine learning models achieved the fastest classification time per image over the deep learning method despite working on a relatively low power system with no GPU configurations. The best results were obtained with the approach of using a pre-trained deep learning model to extract features and then feed those features into a machine learning classifier. Our hypothesis was to reduce the execution time by removing the fully connected layers of deep learning models then replacing them with light and effective machine learning classification models such as SVM or XGBoost. The fused deep learning and machine learning models obtained state-of-the-art classification accuracy and execution time in training and testing, as shown in Table 8.1. Besides being simple, effective growth stage classification was achieved with this approach. To reduce bias, we compared our results with the results obtained in similar works in Chapter 7, Table 7.11. It is very hard to compare the runtime of different works as different researchers use different hardware setups. With our simple setup, our best approach (VGG-19 and RBF SVM) attained an average classification accuracy of 98.4% and a classification time of 0.06 seconds per image. The classification time is also similar to that obtained using the machine learning approach and the accuracy is also close to the transfer learning accuracy. Therefore, we can conclude that the approach of fusing a pretrained deep neural network model and machine learning algorithm can harness the strengths of each approach resulting in a very effective and efficient method.

8.4 Key findings

This section discusses the key findings and the conclusions drawn from these key findings. We begin with the observations collected while developing a plant growth estimator.

- The problem of plant growth is complex, as a plant at different growth stages still poses the same features.
- The conversion of the dataset to grayscale reduces the computation required in all the approaches as the number of channels is reduced from 3 (RGB) to 1 (grayscale) channel.
- Key features can be targeted and extracted using the traditional handcrafting approach, a feature extraction method with tools such as the Gabor filter, Sobel filter, and morphological operators.
- Adding two branches to the proposed method, one where morphological operations of opening and closing are done to highlight the shape and edges of the plant leaves, improves the feature extraction process.
- Gabor filter is a low-level feature extraction method that can extract leaf texture.
- The automatic feature extraction using the dense layers of a convolutional network requires more time as the data has to pass through all the dense layers. Therefore, for this method to be more effective, more data is needed, requiring more computational power.
- The use of randomized search to tune the hyper-parameters (C and gamma) of the SVM kernel can increase the classification accuracy.
- Using the transfer learning approach and preserving the pre-trained models' original weights, higher classification accuracy can be obtained.
- The deeper the neural network is, the more data is required to learn the features that can bring about classification. This was observed as ResNet-50, the deepest neural network from the tested networks in this work, struggled to distinguish growth using our dataset.

- As a result, the hypothesis that deep learning models flourish when more data is available and that the machine learning approach (the handcrafted approach) does not require a lot of data to learn the targeted features was supported.

8.5 Contributions

This study has introduced two computer vision approaches for estimating plant growth stages. One is the traditional machine learning approach of using Gabor filters and morphological filters as the main feature extraction method. The second approach uses a pre-trained deep learning model as a feature extractor and a machine learning model as a classification algorithm.

Using two different plant datasets, we have demonstrated that:

- Handcrafting a feature extraction method can increase the classification accuracy and execution as they target the most important features when designing the feature extraction method.
- The manually developed method does not require big datasets, as they can learn the targeted features with a few samples. We can see this from the results obtained in this study.
- Adding morphological operations (opening and closing) to the method improves the classification accuracy and performance of the method, as they accurately highlight the shape and edges of an object.
- Using a pre-trained model for feature extraction and a machine learning algorithm for classification lowered the execution time and increased the classification.
- We obtained state-of-the-art results with both approaches.

8.6 Source code

A repository with the source code for training and testing all the presented models is published online on GitHub[1] together with the dataset.

8.7 Future Work

Further development of this method will concentrate on improving the texture feature extraction method. With the addition of morphological operators, we believe morphological features have been extracted excellently but not the texture features. From the results, we have seen that the transfer learning methods got higher accuracies, and to improve the accuracies of the machine learning proposed approach, the following can be done:

- Using colour images for feature extraction. With grayscale images, we ignore some important information, such as colour, which can enhance the classification. By using colour images, we consider this information. However, we did not use colour images in this study because of their complexity and computational requirement.
- The addition of another feature extraction method to the feature extraction function, such as local binary pattern (LBP), which is well known for feature extraction in facial recognition problems, can be investigated.
- This method can be assessed with images captured from other environments that are not ideal, such as fields with a complex background, which can improve the model's robustness.
- One shortcoming of the current method is limiting it to only broadleaf plants in indoor farming situations. A more exploratory work would be to evaluate this method in conditions that are not so ideal, situations with complex backgrounds,

[1] https://github.com/SimbaAI/plant-growth-estimator-for-hydroponics

field situations where factors like humidity, lighting, and temperature affect the segmentation and identification of plants.

8.6 Conclusion

With the increase in hydroponics for farming broadleaf plants, it is crucial to improve the management of this farming technique by incorporating computer vision technology. In this work, we have presented a method for estimating the plant growth of two common broadleaf hydroponics crops that is faster, computationally inexpensive, and accurate, with the primary benefit of reducing resource wastage such as nutrient volume, water, and other environmental parameters that can be controlled in automated hydroponics. When the growth stage is known, appropriate decisions to control parameters can be made. We hope researchers can further develop this work to accommodate other farming methods which are not hydroponics.

References

[1] D. Touliatos, I. C. Dodd, and M. McAinsh, "Vertical farming increases lettuce yield per unit area compared to conventional horizontal hydroponics," *Food and energy security*, vol. 5, no. 3, pp. 184–191, 2016.

[2] G. Marques, D. Aleixo, and R. Pitarma, "Enhanced hydroponic agriculture environmental monitoring: An internet of things approach," in *International Conference on Computational Science*, 2019, pp. 658–669.

[3] M. I. Alipio, A. E. M. dela Cruz, J. D. A. Doria, and R. M. S. Fruto, "A smart hydroponics farming system using exact inference in Bayesian network," in *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, 2017, pp. 1–5.

[4] H. K. Srinidhi, H. S. Shreenidhi, and G. S. Vishnu, "Smart Hydroponics system integrating with IoT and Machine learning algorithm," in *2020 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, 2020, pp. 261–264.

[5] O. Tackenberg, "A new method for non-destructive measurement of biomass, growth rates, vertical biomass distribution and dry matter content based on digital image analysis," *Annals of botany*, vol. 99, no. 4, pp. 777–783, 2007.

[6] M. L. Tenzer and N. C. Clifford, "A Digital Green Thumb: Neural Networks to Monitor Hydroponic Plant Growth," in *2020 Systems and Information Engineering Design Symposium (SIEDS)*, 2020, pp. 1–6.

[7] X. He, Y. Chen, and P. Ghamisi, "Heterogeneous transfer learning for hyperspectral image classification based on convolutional neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 5, pp. 3246–3263, 2019.

[8] A. Bauer, A. B. Frank, and A. L. Black, "Estimation of Spring Wheat Leaf Growth Rates and Anthesis from Air Temperature 1," *Agronomy Journal*, vol. 76, no. 5, pp. 829–835, 1984.

[9] K.-J. Lee and B.-W. Lee, "Estimation of rice growth and nitrogen nutrition status using color digital camera image analysis," *European Journal of Agronomy*, vol. 48, pp. 57–65, 2013.

[10] H. Yalcin and S. Razavi, "Plant classification using convolutional neural networks," in *2016 Fifth International Conference on Agro-Geoinformatics (Agro-Geoinformatics)*, 2016, pp. 1–5.

[11] M. Hess *et al.*, "Use of the extended BBCH scale—general for the descriptions of the growth stages of mono; and dicotyledonous weed species," *Weed Research*, vol. 37, no. 6, pp. 433–441, 1997.

[12] V. N. T. Le, B. Apopei, and K. Alameh, "Effective plant discrimination based on the combination of local binary pattern operators and multiclass support vector machine methods," *Information processing in agriculture*, vol. 6, no. 1, pp. 116–131, 2019.

[13] A. Bhargava and A. Bansal, "Fruits and vegetables quality evaluation using computer vision: A review," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 3, pp. 243–257, 2021.

[14] P. Moallem, A. Serajoddin, and H. Pourghassem, “Computer vision-based apple grading for golden delicious apples based on surface features,” *Information processing in agriculture*, vol. 4, no. 1, pp. 33–40, 2017.
[15] A. H. Kulkarni, H. M. Rai, K. A. Jahagirdar, and R. J Kadkol, “A leaf recognition technique for plant classification using RBPNN and pseudo Zernike moments,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 1, pp. 984–988, 2013.
[16] “Plant Leaves and Leaf Anatomy.” https://www.thoughtco.com/plant-leaves-and-leaf-anatomy-373618 (accessed Oct. 18, 2021).
[17] “Anatomy of a Dicot and Monocot Leaves.” https://www.brainkart.com/article/Anatomy-of-a-Dicot-and-Monocot-Leaves_33043/ (accessed Dec. 07, 2021).
[18] A. Derakhshan, A. Bakhshandeh, S. A. allah Siadat, M. R. Moradi-Telavat, and S. B. Andarzian, “Quantifying the germination response of spring canola (Brassica napus L.) to temperature,” *Industrial Crops and Products*, vol. 122, pp. 195–201, Oct. 2018, doi: 10.1016/J.INDCROP.2018.05.075.
[19] “Types of leaves – Botanical online.” https://www.botanical-online.com/en/botany/leaves-types (accessed Oct. 18, 2021).
[20] J. Wäldchen and P. Mäder, “Plant species identification using computer vision techniques: A systematic literature review,” *Archives of Computational Methods in Engineering*, vol. 25, no. 2, pp. 507–543, 2018.
[21] J. Chaki and R. Parekh, “Plant leaf recognition using shape based features and neural network classifiers,” *International Journal of Advanced Computer Science and Applications*, vol. 2, no. 10, 2011.
[22] A. Aakif and M. F. Khan, “Automatic classification of plants based on their leaves,” *Biosystems Engineering*, vol. 139, pp. 66–75, 2015.
[23] D. Zhang, M. M. Islam, and G. Lu, “A review on automatic image annotation techniques,” *Pattern Recognition*, vol. 45, no. 1, pp. 346–362, 2012.
[24] M. Seeland *et al.*, “Description of flower colors for image based plant species classification,” 2016.
[25] M.-E. Nilsback and A. Zisserman, “A visual vocabulary for flower classification,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, 2006, vol. 2, pp. 1447–1454.
[26] A. H. Schistad and A. K. Jain, “Texture Analysis in the Presence of Speckle Noise,” *[Proceedings] IGARSS ’92 International Geoscience and Remote Sensing Symposium*, vol. 2, pp. 884–886, 1992.
[27] J. S. Cope, P. Remagnino, S. Barman, and P. Wilkin, “Plant texture classification using gabor co-occurrences,” in *International Symposium on Visual Computing*, 2010, pp. 669–677.
[28] J. Chaki, R. Parekh, and S. Bhattacharya, “Plant leaf recognition using texture and shape features with neural classifiers,” *Pattern Recognition Letters*, vol. 58, pp. 61–68, 2015.
[29] J. Wäldchen and P. Mäder, “Plant species identification using computer vision techniques: A systematic literature review,” *Archives of Computational Methods in Engineering*, vol. 25, no. 2, pp. 507–543, 2018.

[30] Z. Wang, B. Lu, Z. Chi, and D. Feng, “Leaf image classification with shape context and sift descriptors,” in *2011 International Conference on Digital Image Computing: Techniques and Applications*, 2011, pp. 650–654.
[31] O. M. Bruno, R. de Oliveira Plotze, M. Falvo, and M. de Castro, “Fractal dimension applied to plant identification,” *Information Sciences*, vol. 178, no. 12, pp. 2722–2733, 2008.
[32] C. Gwo and C. Wei, “Plant identification through images: Using feature extraction of key points on leaf contours1,” *Applications in plant sciences*, vol. 1, no. 11, p. 1200005, 2013.
[33] M. P. Arakeri, “Computer vision based fruit grading system for quality evaluation of tomato in agriculture industry,” *Procedia Computer Science*, vol. 79, pp. 426–433, 2016.
[34] J. Jhawar, “Orange sorting by applying pattern recognition on colour image,” *Procedia computer science*, vol. 78, pp. 691–697, 2016.
[35] M. K. Tripathi and D. D. Maktedar, “A role of computer vision in fruits and vegetables among various horticulture products of agriculture fields: A survey,” *Information Processing in Agriculture*, vol. 7, no. 2, pp. 183–203, 2020.
[36] M. P. Arakeri, B. P. V. Kumar, S. Barsaiya, and H. v Sairam, “Computer vision based robotic weed control system for precision agriculture,” in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1201–1205.
[37] L. Hong, Y. Wan, and A. Jain, “Fingerprint image enhancement: algorithm and performance evaluation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 777–789, 1998.
[38] Z. Huang *et al.*, “Optical remote sensing image enhancement with weak structure preservation via spatially adaptive gamma correction,” *Infrared Physics & Technology*, vol. 94, pp. 38–47, 2018.
[39] R. Firoz, M. S. Ali, M. N. U. Khan, M. K. Hossain, M. K. Islam, and M. Shahinuzzaman, “Medical image enhancement using morphological transformation,” *Journal of Data Analysis and Information Processing*, vol. 4, no. 1, pp. 1–12, 2016.
[40] Z. Wang, K. Wang, F. Yang, S. Pan, Y. Han, and X. Zhao, “Image enhancement for crop trait information acquisition system,” *Information Processing in Agriculture*, vol. 5, no. 4, pp. 433–442, 2018.
[41] L. F. Tian and D. C. Slaughter, “Environmentally adaptive segmentation algorithm for outdoor image segmentation,” *Computers and electronics in agriculture*, vol. 21, no. 3, pp. 153–168, 1998.
[42] L. Li, Y. Si, and Z. Jia, “Remote sensing image enhancement based on adaptive thresholding in NSCT domain,” in *2017 2nd International Conference on Image, Vision and Computing (ICIVC)*, 2017, pp. 319–322.
[43] L. Spirkovska, *A summary of image segmentation techniques*, vol. 104022. Ames Research Center, 1993.
[44] P. Sharma, Y. P. S. Berwal, and W. Ghai, “Performance analysis of deep learning CNN models for disease detection in plants using image segmentation,” *Information Processing in Agriculture*, vol. 7, no. 4, pp. 566–574, 2020.

[45] D. M. Woebbecke, G. E. Meyer, K. von Bargen, and D. A. Mortensen, “Plant species identification, size, and enumeration using machine vision techniques on near-binary images,” in *Optics in Agriculture and Forestry*, 1993, vol. 1836, pp. 208–219.
[46] X. P. Burgos-Artizzu, A. Ribeiro, M. Guijarro, and G. Pajares, “Real-time image processing for crop/weed discrimination in maize fields,” *Computers and Electronics in Agriculture*, vol. 75, no. 2, pp. 337–346, 2011.
[47] M. Reyniers, E. Vrindts, and J. de Baerdemaeker, “Optical measurement of crop cover for yield prediction of wheat,” *Biosystems engineering*, vol. 89, no. 4, pp. 383–394, 2004.
[48] E. Hamuda, B. Mc Ginley, M. Glavin, and E. Jones, “Automatic crop detection under field conditions using the HSV colour space and morphological operations,” *Computers and electronics in agriculture*, vol. 133, pp. 97–107, 2017.
[49] H. Zhang, X. Wang, L. Jiang, Y. Xu, and G. Jiang, “Near color recognition based on residual vector and SVM,” *Multimedia Tools and Applications*, vol. 78, no. 24, pp. 35313–35328, 2019.
[50] E. Hamuda, M. Glavin, and E. Jones, “A survey of image processing techniques for plant extraction and segmentation in the field,” *Computers and Electronics in Agriculture*, vol. 125, pp. 184–199, 2016.
[51] N. Liu *et al.*, “Growth stages classification of potato crop based on analysis of spectral response and variables optimization,” *Sensors*, vol. 20, no. 14, p. 3995, 2020.
[52] F. Gatius, J. Lloveras, J. Ferran, and J. Puy, “Prediction of crude protein and classification of the growth stage of wheat plant samples from NIR spectra,” *The Journal of Agricultural Science*, vol. 142, no. 5, pp. 517–524, 2004.
[53] K. Murata, A. Ito, Y. Takahashi, and H. Hatano, “A Study on Growth Stage Classification of Paddy Rice by CNN using NDVI Images,” in *2019 Cybersecurity and Cyberforensics Conference (CCC)*, 2019, pp. 85–90.
[54] R. Kusumaningrum, W. Satriaji, S. N. Endah, Y. Prasetyo, and A. Sukmono, “Classification of rice growth stage based on convolutional neural network,” in *Journal of Physics: Conference Series*, 2020, vol. 1524, no. 1, p. 012114.
[55] F. Ramadhani, R. Pullanagari, G. Kereszturi, and J. Procter, “Automatic mapping of rice growth stages using the integration of SENTINEL-2, MOD13Q1, and SENTINEL-1,” *Remote Sensing*, vol. 12, no. 21, p. 3613, 2020.
[56] D. M. Woebbecke, G. E. Meyer, K. von Bargen, and D. A. Mortensen, “Color indices for weed identification under various soil, residue, and lighting conditions,” *Transactions of the ASAE*, vol. 38, no. 1, pp. 259–269, 1995.
[57] T. Hague, N. D. Tillett, and H. Wheeler, “Automated crop and weed monitoring in widely spaced cereals,” *Precision Agriculture*, vol. 7, no. 1, pp. 21–32, 2006.
[58] G. Ruiz-Ruiz, J. Gómez-Gil, and L. M. Navas-Gracia, “Testing different color spaces based on hue for the environmentally adaptive segmentation algorithm (EASA),” *Computers and Electronics in Agriculture*, vol. 68, no. 1, pp. 88–96, 2009.
[59] N. Teimouri, M. Dyrmann, P. R. Nielsen, S. K. Mathiassen, G. J. Somerville, and R. N. Jørgensen, “Weed growth stage estimator using deep convolutional neural networks,” *Sensors*, vol. 18, no. 5, p. 1580, 2018.
[60] M. V. Giuffrida, M. Minervini, and S. A. Tsaftaris, “Learning to count leaves in rosette plants,” 2016.

[61] P. J. M. Loresco, I. C. Valenzuela, and E. P. Dadios, “Color space analysis using KNN for lettuce crop stages identification in smart farm setup,” in *TENCON 2018-2018 IEEE Region 10 Conference*, 2018, pp. 2040–2044.
[62] R. S. Concepcion II *et al.*, “Lettuce growth stage identification based on phytomorphological variations using coupled color superpixels and multifold watershed transformation,” *International Journal of Advances in Intelligent Informatics*, vol. 6, no. 3, pp. 261–277, 2020.
[63] H. Chandel and S. Vatta, “Occlusion detection and handling: a review,” *International Journal of Computer Applications*, vol. 120, no. 10, 2015.
[64] W. S. Lee and D. C. Slaughter, “Recognition of partially occluded plant leaves using a modified watershed algorithm,” *Transactions of the ASAE*, vol. 47, no. 4, p. 1269, 2004.
[65] C. Niu, H. Li, Y. Niu, Z. Zhou, Y. Bu, and W. Zheng, “Segmentation of cotton leaves based on improved watershed algorithm,” in *International Conference on Computer and Computing Technologies in Agriculture*, 2015, pp. 425–436.
[66] M. H. Malik, T. Zhang, H. Li, M. Zhang, S. Shabbir, and A. Saeed, “Mature tomato fruit detection algorithm based on improved HSV and watershed algorithm,” *IFAC-PapersOnLine*, vol. 51, no. 17, pp. 431–436, 2018.
[67] Y. Tian, G. Yang, Z. Wang, H. Wang, E. Li, and Z. Liang, “Apple detection during different growth stages in orchards using the improved YOLO-V3 model,” *Computers and electronics in agriculture*, vol. 157, pp. 417–426, 2019.
[68] G. Liu, J. C. Nouaze, P. L. Touko Mbouembe, and J. H. Kim, “YOLO-tomato: A robust algorithm for tomato detection based on YOLOv3,” *Sensors*, vol. 20, no. 7, p. 2145, 2020.
[69] J. Alejandrino *et al.*, “Visual classification of lettuce growth stage based on morphological attributes using unsupervised machine learning models,” in *2020 IEEE REGION 10 CONFERENCE (TENCON)*, 2020, pp. 438–443.
[70] A. D. A. Aldabbagh, C. Hairu, and M. Hanafi, “Classification of chili plant growth using deep learning,” *2020 IEEE 10th International Conference on System Engineering and Technology, ICSET 2020 - Proceedings*, pp. 213–217, Nov. 2020, doi: 10.1109/ICSET51301.2020.9265351.
[71] B. T. Borille, M. C. A. Marcelo, R. S. Ortiz, K. de Cássia Mariotti, M. F. Ferrão, and R. P. Limberger, “Near infrared spectroscopy combined with chemometrics for growth stage classification of cannabis cultivated in a greenhouse from seized seeds,” *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy*, vol. 173, pp. 318–323, 2017.
[72] S. Rasti, C. J. Bleakley, G. C. M. Silvestre, N. M. Holden, D. Langton, and G. M. P. O’Hare, “Crop growth stage estimation prior to canopy closure using deep learning algorithms,” *Neural Computing and Applications*, vol. 33, no. 5, pp. 1733–1743, 2021.
[73] F. Mostajer Kheirkhah and H. Asghari, “Plant leaf classification using GIST texture features,” *IET Computer Vision*, vol. 13, no. 4, pp. 369–375, 2019.
[74] H. J. Vala and A. Baxi, “A review on Otsu image segmentation algorithm,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 2, no. 2, pp. 387–389, 2013.
[75] X. XIE, “Principal component analysis,” 2019.
[76] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE.,” *Journal of machine learning research*, vol. 9, no. 11, 2008.

[77] M. Salman, S. Mathavan, K. Kamal, and M. Rahman, “Pavement crack detection using the Gabor filter,” in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, 2013, pp. 2039–2044. doi: 10.1109/ITSC.2013.6728529.

[78] A. Daamouche, D. Fares, I. Maalem, and K. Zemmouri, “Unsupervised method for building detection using Gabor Filters,” *Acta Physica Polonica A*, vol. 130, no. 1, pp. 28–29, 2016.

[79] A. G. Ramakrishnan, S. Kumar Raja, and H. V. Raghu Ram, “Neural network-based segmentation of textures using Gabor features,” *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, vol. 2002-January, pp. 365–374, 2002, doi: 10.1109/NNSP.2002.1030048.

[80] “3D surface tracking and approximation using Gabor filters - CoViL.” https://www.yumpu.com/en/document/read/44234347/3d-surface-tracking-and-approximation-using-gabor-filters-covil (accessed Nov. 04, 2021).

[81] M. Rivera, O. Dalmau, A. Gonzalez, and F. Hernandez-Lopez, “Two-step fringe pattern analysis with a Gabor filter bank,” *Optics and Lasers in Engineering*, vol. 85, pp. 29–37, Oct. 2016, doi: 10.1016/J.OPTLASENG.2016.04.014.

[82] F. Alonso-Fernandez, J. Fierrez-Aguilar, and J. Ortega-Garcia, “An enhanced Gabor filter-based segmentation algorithm for fingerprint recognition systems,” in *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.*, 2005, pp. 239–244. doi: 10.1109/ISPA.2005.195416.

[83] Z. Zhang, W. Wang, and K. Lu, “Video Text Extraction Using the Fusion of Color Gradient and Log-Gabor Filter,” in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 2938–2943. doi: 10.1109/ICPR.2014.506.

[84] J. G. Daugman, “Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters,” *J. Opt. Soc. Am. A*, vol. 2, no. 7, pp. 1160–1169, Jul. 1985, doi: 10.1364/JOSAA.2.001160.

[85] D. Gabor, “Theory of communication. Part 1: The analysis of information,” *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, 1946.

[86] G. H. Granlund, “In search of a general picture processing operator,” *Computer Graphics and Image Processing*, vol. 8, no. 2, pp. 155–173, Oct. 1978, doi: 10.1016/0146-664X(78)90047-3.

[87] W. Gao, L. Yang, X. Zhang, and H. Liu, “An improved Sobel edge detection,” *Proceedings - 2010 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2010*, vol. 5, pp. 67–71, 2010, doi: 10.1109/ICCSIT.2010.5563693.

[88] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Ijcai*, 1995, vol. 14, no. 2, pp. 1137–1145.

[89] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152.

[90] E. Owusu, Y. Zhan, and Q. R. Mao, “An SVM-AdaBoost facial expression recognition system,” *Applied intelligence*, vol. 40, no. 3, pp. 536–545, 2014.

[91] N. Kasabov and S. Pang, “Transductive support vector machines and applications in bioinformatics for promoter recognition,” in *International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003*, 2003, vol. 1, pp. 1–6.
[92] A. Ozcift, “SVM feature selection based rotation forest ensemble classifiers to improve computer-aided diagnosis of Parkinson disease,” *Journal of medical systems*, vol. 36, no. 4, pp. 2141–2147, 2012.
[93] J. Li, H. Zhou, P. Xie, and Y. Zhang, “Improving the Generalization Performance of Multi-class SVM via Angular Regularization.,” in *IJCAI*, 2017, pp. 2131–2137.
[94] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, “A Practical Guide to Support Vector Classification.” [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin
[95] “SVM and Kernel SVM. Learn about SVM or Support Vector… | by Czako Zoltan | Towards Data Science.” https://towardsdatascience.com/svm-and-kernel-svm-fed02bef1200 (accessed Nov. 07, 2021).
[96] M. L. Comer and E. J. D. III, “Morphological operations for color image processing,” *https://doi.org/10.1117/1.482677*, vol. 8, no. 3, pp. 279–289, Jul. 1999, doi: 10.1117/1.482677.
[97] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors),” *https://doi.org/10.1214/aos/1016218223*, vol. 28, no. 2, pp. 337–407, Apr. 2000, doi: 10.1214/AOS/1016218223.
[98] D. Zhang and Y. Gong, “The Comparison of LightGBM and XGBoost Coupling Factor Analysis and Prediagnosis of Acute Liver Failure,” *IEEE Access*, 2020, doi: 10.1109/ACCESS.2020.3042848.
[99] X. Dairu and Z. Shilong, “Machine Learning Model for Sales Forecasting by Using XGBoost,” *2021 IEEE International Conference on Consumer Electronics and Computer Engineering, ICCECE 2021*, pp. 480–483, Jan. 2021, doi: 10.1109/ICCECE51280.2021.9342304.
[100] M. Gumus and M. S. Kiran, “Crude oil price forecasting using XGBoost,” *2nd International Conference on Computer Science and Engineering, UBMK 2017*, pp. 1100–1103, Oct. 2017, doi: 10.1109/UBMK.2017.8093500.
[101] D. Zhang, L. Qian, B. Mao, C. Huang, B. Huang, and Y. Si, “A Data-Driven Design for Fault Detection of Wind Turbines Using Random Forests and XGboost,” *IEEE Access*, vol. 6, pp. 21020–21031, Mar. 2018, doi: 10.1109/ACCESS.2018.2818678.
[102] A. Ogunleye and Q. G. Wang, “XGBoost Model for Chronic Kidney Disease Diagnosis,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 17, no. 6, pp. 2131–2140, Nov. 2020, doi: 10.1109/TCBB.2019.2911071.
[103] “Deep Learning | The MIT Press.” https://mitpress.mit.edu/books/deep-learning (accessed Nov. 08, 2021).
[104] “Introducing Deep Learning with MATLAB - MATLAB & Simulink.” https://www.mathworks.com/campaigns/offers/next/deep-learning-ebook.html (accessed Nov. 08, 2021).
[105] J. D. Bodapati and N. Veeranjaneyulu, “Feature Extraction and Classification Using Deep Convolutional Neural Networks,” *Journal of Cyber Security and Mobility*, vol. 8, no. 2, pp. 261–276, Apr. 2019, doi: 10.13052/JCSM2245-1439.825.

[106] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” *Neural Networks*, vol. 6, no. 6, pp. 861–867, Jan. 1993, doi: 10.1016/S0893-6080(05)80131-5.
[107] A. M. Fred Agarap, “Deep Learning using Rectified Linear Units (ReLU),” Mar. 2018, Accessed: Nov. 09, 2021. [Online]. Available: https://arxiv.org/abs/1803.08375v2
[108] “Machine learning fundamentals (I): Cost functions and gradient descent | by Conor Mc. | Towards Data Science.” https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220 (accessed Nov. 09, 2021).
[109] “A Beginner’s Guide To Understanding Convolutional Neural Networks Part 2 – Adit Deshpande – Engineering at Forward | UCLA CS ’19.” https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/ (accessed Nov. 09, 2021).
[110] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.
[111] “Everything you need to know about Neural Networks | Hacker Noon.” https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491 (accessed Nov. 09, 2021).
[112] R. Rojas, “Neural Networks: Backpropagation Algorithm,” *Behaviorism: a Conceptual Reconstruction*, p. 502, 1996, Accessed: Nov. 09, 2021. [Online]. Available: http://link.springer.com/10.1007/978-3-642-61068-4
[113] R. Rojas, “The Backpropagation Algorithm,” *Neural Networks*, pp. 149–182, 1996, doi: 10.1007/978-3-642-61068-4_7.
[114] Y. H. Zweiri, J. F. Whidborne, and L. D. Seneviratne, “A three-term backpropagation algorithm,” *Neurocomputing*, vol. 50, pp. 305–318, Jan. 2003, doi: 10.1016/S0925-2312(02)00569-6.
[115] “Machine Learning: A Probabilistic Perspective - Kevin P. Murphy - Google Books.” https://books.google.co.za/books?hl=en&lr=&id=RC43AgAAQBAJ&oi=fnd&pg=PR7&dq=Machine+Learning+A+Probabilistic+Perspective+Kevin+P.+Murphy&ots=umiAdARr0c&sig=kBe7Y-h_M2G8hgz-_oafz4NLAKU&redir_esc=y#v=onepage&q=Machine%20Learning%20A%20Probabilistic%20Perspective%20Kevin%20P.%20Murphy&f=false (accessed Nov. 09, 2021).
[116] “Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der ... - Chollet, François - Google Books.” https://books.google.co.za/books?hl=en&lr=&id=ouVcDwAAQBAJ&oi=fnd&pg=PT2&dq=Chollet,+F.,+2018.+Deep+Learning+with+Python.+1st+ed.+Shelter+Island:+Manning+Publications+Co.&ots=3Zn1Z1YFD4&sig=Em9Vt0ABHgrfQyKf5uFh48g3Wok&redir_esc=y#v=onepage&q&f=false (accessed Nov. 09, 2021).
[117] I. Z. Mukti and D. Biswas, “Transfer learning based plant diseases detection using ResNet50,” in *2019 4th International Conference on Electrical Information and Communication Technology (EICT)*, 2019, pp. 1–6.
[118] S. CS231n, “Convolutional neural networks for visual recognition,” *URL: https://cs231n. github. io/neural-networks-3/# baby (дата обращения 01.09. 2020)*, 2017.

[119] V. N. T. Le, B. Apopei, and K. Alameh, “Effective plant discrimination based on the combination of local binary pattern operators and multiclass support vector machine methods,” *Information processing in agriculture*, vol. 6, no. 1, pp. 116–131, 2019.
[120] K. Weiss, T. M. Khoshgoftaar, and D. D. Wang, “A survey of transfer learning,” *Journal of Big Data*, vol. 3, no. 1, pp. 1–40, Dec. 2016, doi: 10.1186/S40537-016-0043-6/TABLES/6.
[121] A. Kaya, A. S. Keceli, C. Catal, H. Y. Yalic, H. Temucin, and B. Tekinerdogan, “Analysis of transfer learning for deep neural network based plant classification models,” *Computers and Electronics in Agriculture*, vol. 158, pp. 20–29, Mar. 2019, doi: 10.1016/J.COMPAG.2019.01.041.
[122] M. Mehdipour Ghazi, B. Yanikoglu, and E. Aptoula, “Plant identification using deep neural networks via optimization of transfer learning parameters,” *Neurocomputing*, vol. 235, pp. 228–235, Apr. 2017, doi: 10.1016/J.NEUCOM.2017.01.018.
[123] J. Champ, T. Lorieul, M. Servajean, and A. Joly, “A comparative study of fine-grained classification methods in the context of the LifeCLEF plant identification challenge 2015,” no. 1391, Sep. 2015, Accessed: Nov. 09, 2021. [Online]. Available: https://hal.inria.fr/hal-01182788
[124] K. Thenmozhi and U. Srinivasulu Reddy, “Crop pest classification based on deep convolutional neural network and transfer learning,” *Computers and Electronics in Agriculture*, vol. 164, p. 104906, Sep. 2019, doi: 10.1016/J.COMPAG.2019.104906.
[125] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Sep. 2014, Accessed: Nov. 08, 2021. [Online]. Available: https://arxiv.org/abs/1409.1556v6
[126] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 770–778, Dec. 2015, doi: 10.1109/CVPR.2016.90.
[127] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, 2009, pp. 248–255.
[128] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
[129] E. Rezende *et al.*, “Malicious Software Classification Using VGG16 Deep Neural Network’s Bottleneck Features,” *Advances in Intelligent Systems and Computing*, vol. 738, pp. 51–59, 2018, doi: 10.1007/978-3-319-77028-4_9.
[130] A. S. B. Reddy and D. S. Juliet, “Transfer learning with ResNet-50 for malaria cell-image classification,” in *2019 International Conference on Communication and Signal Processing (ICCSP)*, 2019, pp. 945–949.
[131] S. Vesal, N. Ravikumar, A. Davari, S. Ellmann, and A. Maier, “Classification of breast cancer histology images using transfer learning,” in *International conference image analysis and recognition*, 2018, pp. 812–819.
[132] P. Muralidharan and C. S. Kumar, “Fusion of Bottleneck Features Derived from CNNs to Enhance the Performance of Multi-Parameter Patient Monitors,” *Proceedings of the 2nd*

International Conference on Inventive Research in Computing Applications, ICIRCA 2020, pp. 601–605, Jul. 2020, doi: 10.1109/ICIRCA48905.2020.9183077.
[133] "Home — Spyder IDE." https://www.spyder-ide.org/ (accessed Nov. 10, 2021).
[134] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
[135] J. v Dillon *et al.*, "Tensorflow distributions," *arXiv preprint arXiv:1711.10604*, 2017.
[136] S. Purushotham and B. K. Tripathy, "Evaluation of Classifier Models Using Stratified Tenfold Cross Validation Techniques," *Communications in Computer and Information Science*, vol. 270 CCIS, no. PART II, pp. 680–690, 2011, doi: 10.1007/978-3-642-29216-3_74.
[137] R. Vyas, T. Kanumuri, and G. Sheoran, "Iris recognition using 2-D Gabor filter and XOR-SUM code," in *2016 1st India International Conference on Information Processing (IICIP)*, 2016, pp. 1–5. doi: 10.1109/IICIP.2016.7975369.
[138] A. Kumar, "An Efficient Approach for Text Extraction in Images and Video Frames Using Gabor Filter," *International Journal of Computer and Electrical Engineering*, vol. 6, no. 4, pp. 316–320, 2014, doi: 10.7763/ijcee.2014.v6.845.
[139] S. R. Zhou, J. P. Yin, and J. M. Zhang, "Local binary pattern (LBP) and local phase quantization (LBQ) based on Gabor filter for face representation," *Neurocomputing*, vol. 116, pp. 260–264, Sep. 2013, doi: 10.1016/J.NEUCOM.2012.05.036.
[140] D. Gabor, "Theory of communication. Part 1: The analysis of information," *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, 1946.
[141] I. S. Al-Mejibli, J. K. Alwan, and H. Abd Dhafar, "The effect of gamma value on support vector machine performance with different kernels," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 5, p. 5497, 2020.
[142] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," Sep. 2013, Accessed: Dec. 06, 2021. [Online]. Available: https://arxiv.org/abs/1309.0238v1
[143] A. Kadir, L. E. Nugroho, A. Susanto, and P. I. Santosa, "Leaf classification using shape, color, and texture features," *arXiv preprint arXiv:1401.4447*, 2013.
[144] "(PDF) Plant Classification Based on Leaf Recognition." https://www.researchgate.net/publication/45638498_Plant_Classification_Based_on_Leaf_Recognition (accessed Nov. 27, 2021).
[145] B. Espejo-Garcia, N. Mylonas, L. Athanasakos, S. Fountas, and I. Vasilakoglou, "Towards weeds identification assistance through transfer learning," *Computers and Electronics in Agriculture*, vol. 171, p. 105306, Apr. 2020, doi: 10.1016/J.COMPAG.2020.105306.
[146] S. Ramaneswaran, K. Srinivasan, P. M. D. R. Vincent, and C. Y. Chang, "Hybrid Inception v3 XGBoost Model for Acute Lymphoblastic Leukemia Classification," *Computational and Mathematical Methods in Medicine*, vol. 2021, 2021, doi: 10.1155/2021/2577375.
[147] C. Sitaula and S. Aryal, "Fusion of whole and part features for the classification of histopathological image of breast tissue," *Health Information Science and Systems 2020 8:1*, vol. 8, no. 1, pp. 1–12, Nov. 2020, doi: 10.1007/S13755-020-00131-7.

[148] V. N. T. Le, S. Ahderom, and K. Alameh, “Performances of the LBP Based Algorithm over CNN Models for Detecting Crops and Weeds with Similar Morphologies,” *Sensors 2020, Vol. 20, Page 2193*, vol. 20, no. 8, p. 2193, Apr. 2020, doi: 10.3390/S20082193.

[149] S. G. Wu, F. S. Bao, E. Y. Xu, Y. Wang, Y. Chang and Q. Xiang, "A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network," 2007 IEEE International Symposium on Signal Processing and Information Technology, 2007, pp. 11-16, doi: 10.1109/ISSPIT.2007.4458016.