New Era University
College of Informatics and Computing Studies
Computer Science Department

TOMATO LEAF HEALTH CLASSIFICATION WITH MANAGEMENT RECOMMENDATIONS USING CNN-BASED DENSENET-121 FOR AGRICULTURAL SUSTAINABILITY

An Undergraduate Thesis Presented to the Faculty of
New Era University, College of Informatics and Computing Studies,
Computer Science Department
New Era, Quezon City

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

JANNAH R. SORIANO
ROSH HASHANA S. TORRES

MAY 2024

APPROVAL SHEET

The thesis entitled **Tomato Leaf Health Classification with Management Recommendations using CNN-based DenseNet-121 for Agricultural Sustainability**, prepared and submitted by **Jannah R. Soriano and Rosh Hashana S. Torres**, in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science, has been examined and recommended for acceptance and approval for the oral examination.

Dr. MARC P. LAURETA
Thesis Adviser

Approved by the Oral Examination with a grade of _____ on _____/_____/_____.

Prof. ERA B. ESPINAS
Committee Chair

Prof. YUMIE M. PASIOL
Member

Prof. EDILBERTO L. SIMBULAN
Member

Accepted as partial requirement for the degree in Bachelor of Science in Computer Science.

Engr. JEREMIAS C. ESPERANZA
Program Head

Prof. AUDREY LYLE D. DIEGO
Dean

ACKNOWLEDGEMENT

The completion of this endeavor could not have been possible without the assistance of numerous people. Their contribution and cooperation were deeply applauded and gladly acknowledged.

We would like to express our deepest gratitude to our family who believed in us from the beginning up until the end of our research process. They consistently provided for our needs and finances, and gave their endless support to us throughout this whole process.

We were grateful for our friends, for they were able to help us in every way they could. We were thankful for them for always reminding us to be strong, and that we could overcome and finish this undertaking.

We would also like to extend our deepest regards and appreciation to our professor in thesis, Dr. Marc P. Laureta, for his utmost dedication and support. He equipped us with adequate knowledge and skills to be able to become hardworking students, not settling for less.

Above all, we thanked our Almighty God, who was the author of our knowledge and skills. We thanked Him for giving us the ability, opportunity, and strength to accomplish this whole study. Without His guidance and blessings, we could not have been able to fulfill this work.

DEDICATION

This research was dedicated to all individuals involved in agriculture, including farmers and agricultural workers, consumers, and plant health authorities and agencies. The researchers recognized the significance of prioritizing the production of healthier tomatoes, emphasizing the need to diagnose and manage tomato leaf health to enhance the quality of the harvested produce. Moreover, they aimed to establish this study as a valuable reference for future researchers exploring similar topics in the agricultural sector.

- *J. R. Soriano*
- *R. H. S. Torres*

ABSTRACT

Title : **Tomato Leaf Health Classification with Management Recommendations using CNN-based DenseNet-121 for Agricultural Sustainability**

Authors : Jannah R. Soriano
Rosh Hashana S. Torres

Adviser : Dr. Marc P. Laureta

School : New Era University

Keywords: Agriculture, Deep Learning, Tomato Leaf, Leaf Health Classification, CNN, DenseNet-121

Technological innovations had transformed agriculture, providing solutions for enhanced sustainability. This study addressed the problem of plant health management in agriculture, by proposing a novel approach using a Convolutional Neural Network (CNN)-based DenseNet-121 model. Using advances in deep learning, particularly in computer vision, the researchers created a web-based application for tomato leaf health classification with management recommendations. The PlantVillage dataset, containing images of tomato leaves depicting both healthy and various diseased states, was used for training and validation. The custom DenseNet-121 model showed remarkable performance metrics after thorough experimentation, obtaining high accuracy, sensitivity, specificity, and F1 scores in training and validation processes. The evaluation of the trained model offered positive results, with an accuracy of 97.37% on the training set and 98.59% on the validation set. Additionally, the model had reasonable specificity (99.77% training, 99.88% validation), sensitivity (97.15% training, 98.59% validation), and F1 scores (97.39% training, 98.67% validation), demonstrating that it was efficient in health classification. By incorporating the developed model into a web-based application, farmers and agricultural workers had access to a valuable tool for plant health management. Furthermore, the addition of management recommendations increased the application's usability by offering actionable information on how to effectively manage diagnosed leaf health. This study significantly contributed to sustainable farming methods by combining modern technology with traditional methods of agriculture, ultimately aiming for the betterment of society.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

Chapter 1

INTRODUCTION

This chapter presented an overview of the study, including its background, objectives, significance, and scope, which was essential for providing the context and rationale behind the research endeavor.

Background of the Study

Technological advancements have fueled innovation, driving humanity to new levels of possibilities. They have encouraged creativity and opened doors to previously unimaginable opportunities. Automation, artificial intelligence (AI), and robotics have streamlined production processes, increasing efficiency and productivity (Murugesan, 2023). These advancements have transformed numerous industries, including the agricultural, influencing the future of work and daily life. According to Moreno-Miranda and Dries (2022), innovations in agriculture production offer a double benefit: they revolutionize farming techniques and improve management of the environmental impact of agriculture. Integrating technologies like Internet of Things (IoT) sensors, drones, satellite imaging, and access to digital platforms and web and mobile applications provides farmers with essential tools for monitoring crop health, soil moisture levels, weather conditions, and other factors.

Agriculture is vital to the world’s economy as it is the primary source of food, income, and employment opportunities. According to the National Economic and

Development Authority (NEDA) and the Department of Finance (DOF), the Philippines, being an agricultural country, is in the best position to have an agriculture-driven economy, which can significantly contribute to rebooting the Philippine economy. Hence, investing in farming technology and infrastructure is essential as it promotes sustainable development. Embracing technology in agriculture has the potential to significantly improve not only the economy but also the whole agricultural industry.

Plant diseases significantly threaten to agricultural productivity, potentially causing financial losses and harming global food security. Traditional disease classification relies on manual examination by experts, which can be subjective, time-consuming, and prone to errors (Dawod & Dobre, 2022). Therefore, early identification and accurate diagnosis of plant health are essential for effective management and minimizing economic losses. Recent advancements in computer vision and artificial intelligence have enabled significant progress in automatic plant health classification and detection. Deep learning techniques, particularly convolutional neural networks (CNNs), have shown promising results in image recognition tasks (Sujatha et al., 2021). Transfer learning, which utilizes pre-trained models trained on large-scale datasets like ImageNet, enhances recognition accuracy and efficiency (Tan et al., 2018; Too et al., 2019).

Tomato (*Solanum lycopersicum*) is a widely cultivated and consumed horticultural crop with South American origins, playing a significant role in global

agriculture and economic development (Grandillo et al., n.d.; Authentica World Cuisine, 2021). Known for their juiciness and versatility, tomatoes are staple ingredients in various cuisines worldwide and are recognized for their healthful benefits, being rich in vitamins and minerals (Rdn, 2023). In the Philippines, tomatoes, or *kamatis*, are a favorite and versatile vegetable, forming a crucial component of Filipino cuisine. It forms one-third of the sauteed holy trinity known as *guisado*, the Filipino equivalent to sofrito. It's the star of the show in countless saucy *ulam* (braises) and fiesta fare, and sours traditional native dishes, from *sawsawan* (dipping sauces) to sinigang (sour soup) (*Why Tomatoes Are Central to Filipino Cuisine*, 2023). The Philippines' climate and soil conditions are favorable for tomato cultivation, making it an important crop for farmers. However, tomato crops are vulnerable to various leaf diseases caused by fungi, bacteria, or viruses, affecting yield, quality, and its overall health (Wiley, 2023). To address this, numerous researchers are exploring computer vision and deep learning techniques for automated classification and detection of leaf health, aiming to improve disease management and sustainable agriculture (Dawod & Dobre, 2022).

The availability of diverse and well-annotated datasets was crucial for training accurate deep learning models. Pennsylvania State University released a plant dataset called PlantVillage (*PlantVillage*, 2015). This publicly available dataset provided a diverse range of plant categories, with each category containing images of both healthy leaves and leaves affected by disease. Among the plants

included was the tomato. The dataset comprised single-leaf images organized into ten categories: one for healthy leaves and nine for various diseased leaves. Since its initial release, numerous studies focused on plant health and disease identification utilized the PlantVillage dataset (Aldhyani et al., 2022; Panigrahi et al., 2020; Kabir et al., 2020; Prodeep et al., 2022).

In the study entitled “Plant Disease Detection using VGG16,” the authors proposed a plant disease classification technique using the VGG-16 deep learning architecture for tomato plant, achieving an accuracy of 88.6%. On the other hand, this study aimed to enhance agricultural sustainability by developing a tomato leaf health classification application using a CNN-based DenseNet-121 model. The researchers aimed to achieve good accuracy, sensitivity, specificity, and F1 score in classifying leaf health by fine-tuning the DenseNet-121 model on the PlantVillage dataset. They created a web-based application that allowed users to upload leaf images for diagnosis. The trained model analyzed the uploaded images and provided health classification results. The researchers integrated a management recommendation into the application to further assist them, suggesting appropriate management strategies based on the diagnosed health information. These recommendations included preventive measures such as organic and chemical control to effectively manage and mitigate identified leaf health. By combining leaf health diagnosis with management recommendations, this study aimed to enhance agricultural sustainability.

Objectives of the Study

The main objective of this study is to develop a tomato leaf health classification application using a CNN-based DenseNet-121 model and providing management recommendations for effective strategies.

The specific objectives of this study are as follows:

1. Apply several preprocessing techniques to improve the quality and diversity of the tomato leaf images dataset for training and validating the model.
2. Develop a customized DenseNet-121 model for tomato leaf health image classification, evaluating its performance through classification accuracy, sensitivity, specificity, and F1 score.
3. Create a web-based application that utilizes the trained model to classify tomato leaf health based on images uploaded by users.
4. Integrate management recommendations into the application, imposing the diagnosed health information to suggest appropriate management strategies.

Significance of the Study

This study aimed to benefit various stakeholders, including farmers and agricultural workers, consumers, plant health authorities and agencies, and future researchers, by providing efficient tomato leaf health identification and management techniques.

The researchers had identified the beneficiaries of the proposed study as follows:

Farmers and Agricultural Workers. They benefited from this study as it helped them quickly identify and diagnose health issues affecting their tomatoes. With the management recommendations provided, they could effectively manage and mitigate the identified tomato leaf health problems, leading to improved crop yield and reduced economic losses.

Consumers. They benefited from this study as it helped ensure the production of healthier tomatoes. By diagnosing and managing tomato leaf health, the quality of the harvested produce could be enhanced, resulting in safer and higher-quality food for consumers in the Philippines.

Plant Health Authorities and Agencies. They benefited from this study as plant health authorities and agencies could utilize the research outcomes, such as the Department of Agriculture (DA) and the Bureau of Plant Industry (BPI). The deep learning-based approach developed in this study enhanced their capabilities in detecting and controlling tomato leaf health effectively, thereby improving their health surveillance and management programs.

Future Researchers. They benefited from this study as it contributed to the development of advanced health classification and management techniques. The CNN-based DenseNet-121 model and the application developed in this study could serve as a valuable reference for further research and innovation in agricultural sustainability and crop protection.

Scope and Limitations

This study aimed to develop a tomato leaf health classification application using a CNN-based DenseNet-121 model. A diverse dataset containing images of healthy tomato leaves and leaves affected by various diseases was acquired. The focus was on implementing a convolutional neural network model based on the DenseNet-121 architecture, ensuring the classification of tomato leaf health. Additionally, a web-based application was created to allow farmers and agricultural workers to upload tomato leaf images for health diagnosis. Furthermore, the application integrated management recommendations to suggest suitable organic and chemical control strategies for managing and mitigating identified tomato leaf health problems. The dataset used in this study, as shown in **Table 1**, consisted of one (1) healthy leaf class and one (1) diseased leaf class, totaling two classes.

Table 1. Description of the Dataset

Plant Type	Number of Classes	Classes Names
Tomato	2	Healthy_Leaf Diseased_Leaf

This study had certain limitations that should be acknowledged. Primarily, it focused on tomato leaf health classification with management recommendations using a CNN-based DenseNet-121 model, overlooking other important aspects of agricultural sustainability. The findings might not have directly applied to different crop species, as this study focused solely on tomato plants. The accuracy of the diagnosis might have been affected by factors like image quality and user errors since the application relied on user-uploaded leaf images. Additionally, this study

only focused on identifying plant health on single leaves and did not extend to identifying health on groups of leaves.

Chapter 2

REVIEW OF RELATED LITERATURE

This chapter reviewed the relevant literature for establishing the foundational knowledge required to build and develop the study. The researchers gathered and utilized a range of literature about Deep Learning in Agriculture, Leaf Disease Classification, and CNN and DenseNet-121.

Deep Learning in Agriculture

Deep learning techniques have gained significant popularity in agricultural applications, particularly in addressing challenges related to crop management and sustainability. Among deep learning architectures, Convolutional Neural Networks (CNNs) stand out for their effectiveness in image-based classification tasks within agriculture. Studies have shown that CNNs offer effective solutions for accurately identifying plant diseases and pests, facilitating timely interventions for crop health management (Fuentes et al., 2019). The integration of deep learning in agriculture not only enhances productivity but also contributes to environmental sustainability through targeted and efficient management strategies.

In the field of plant disease classification, deep learning models have showcased remarkable performance compared to conventional methods. For instance, Sladojevic et al. (2016) demonstrated the superiority of CNN-based approaches in identifying plant diseases from leaf images with higher accuracy and efficiency. This capability is crucial for early disease detection, enabling

farmers to implement timely management practices and mitigate yield losses. Moreover, deep learning models can be tailored to specific crops and diseases, offering customized solutions adaptable to diverse agricultural contexts (Mohanty et al., 2016).

Deep learning algorithms have also been instrumental in developing decision support systems for promoting agricultural sustainability. By combining deep learning models with agronomic knowledge and environmental data, researchers have devised tools capable of providing real-time management recommendations to farmers. For example, Minervini et al. (2015) developed a deep learning-based decision support system for monitoring crop growth and detecting stress factors, enabling precise resource allocation and optimization of inputs. Such systems play a vital role in sustainable agriculture by promoting efficient resource utilization and reducing reliance on chemical inputs.

Despite the promises offered by deep learning in agriculture, several challenges persist, including data availability and model interpretability. Lack of access to annotated datasets and variations in environmental conditions can limit the development and generalization of deep learning models across different regions and crops (Ramcharan et al., 2019). Additionally, the opaque nature of deep learning algorithms poses challenges in understanding the underlying decision-making process, raising concerns about transparency and accountability in automated agricultural systems (Kamilaris et al., 2018). Addressing these

challenges requires concerted efforts in data collection, model validation, and transparency in algorithmic decision-making.

The utilization of deep learning, particularly CNNs, holds immense potential for transforming crop management practices in agriculture. By harnessing the power of image analysis and pattern recognition, deep learning models enable accurate classification of plant diseases, real-time monitoring of crop health, and provision of timely management recommendations for sustainable agriculture. However, addressing challenges related to data availability and model interpretability is critical for realizing the full benefits of deep learning in agriculture and ensuring its responsible deployment in real-world farming scenarios.

Leaf Disease Classification

Leaf diseases present a significant threat to agricultural sustainability, as they can cause substantial losses in crop production and reduce the quality and quantity of food (Alston & Pardey, 2014). Identifying and diagnosing leaf diseases accurately and early on is crucial for effective management and treatment. There are frequently noticeable spots on infected plants' fruits, flowers, stems, or leaves. Each infection and pest condition, in particular, leaves behind unique patterns that can be utilized to diagnose abnormalities. Traditional methods of disease identification rely on manual examination by farmers or experts, which can be subjective, time-consuming, and prone to errors (Dawod & Dobre, 2022). In recent

years, researchers have turned to computer vision and deep learning techniques to automate the detecting and classifying leaf diseases.

Computer vision techniques utilize image processing and machine learning algorithms to analyze visual information from leaf images and identify disease symptoms. Various approaches have been proposed, including hand-crafted feature extraction and segmentation combined with machine learning algorithms. Dubey and Jalal (2013) introduced a K-means clustering algorithm to segment the infected area of leaves. They subsequently employed a multi-class support vector machine (SVM) for the final classification. In a different study, Yun et al. (2015) utilized a probabilistic neural network to extract meteorological and statistical features. These experiments were conducted on cucumber plants affected by cucumber downy mildew, anthracnose, and blights. Various models based on conventional methods have also been suggested for plant disease recognition. For instance, Li et al. (2012) employed SVM and K-means clustering techniques in combination with a backpropagation neural network in their research. These methods have shown promising results but often require extensive manual feature engineering and lack scalability.

Deep learning, specifically convolutional neural networks (CNNs), has revolutionized the field of image classification and object detection (Sujatha et al., 2021). CNNs can automatically learn and extract complex features from images, making them well-suited for leaf disease classification. Kumar et al. (2023)

proposed a novel plant disease detection technique based on deep learning using the VGG16 architecture. The study specifically targeted disease detection in tomato and potato plants. The approach achieved a significant disease detection accuracy of 88.6%. Krishnamoorthy et al. (2021) investigated and compared different CNN architectures for tomato leaf disease prediction using deep learning. The study emphasized the importance of monitoring tomato crops for disease detection to ensure optimal production. The proposed method employed Convolutional Neural Networks (CNNs) to identify tomato leaf diseases using extracted features from images. Two different CNN architectures, namely LeNet and AlexNet, were implemented. LeNet achieved an accuracy of around 90-92%, and AlexNet attained an accuracy of around 92-97%.

Furthermore, Zaki et al. (2020) proposed a computer vision approach using the MobileNet V2 deep learning architecture for tomato leaf disease classification. They fine-tuned the MobileNet V2 model to detect three types of tomato diseases and achieved over 90% accuracy on the PlantVillage dataset. Rangarajan et al. (2018) compared the performance of two deep learning architectures, AlexNet and VGG16 net, for tomato crop disease classification. They analyzed the impact of the number of images and hyperparameters on classification accuracy and execution time. Integrating of deep learning algorithms showcases the potential of these approaches in enhancing agricultural sustainability and crop management.

Table 2 presents a detailed summary of CNN Models used in leaf disease classification in Tomatoes using the PlantVillage dataset.

Table 2. Summary of CNN Models for Leaf Disease Classification in Tomato Using PlantVillage dataset

Study	Year of Publication	Classes Names	Model	Accuracy
Plant Disease Detection using VGG16 (Focal Paper)	2023	**1.** healthy **2.** target_spot **3.** mosaic_virus **4.** yellow_leaf_curl_virus	VGG-16	88.6%
Investigation and comparison of different CNN architectures on tomato leaf disease prediction using deep learning	2021	**1.** bacterial_spot **2.** early_blight **3.** healthy **4.** late_blight **5.** leaf_mold **6.** septoria_leaf_spot **7.** spider_mites_two-spotted_spider_mite **8.** target_spot **9.** mosaic_virus **10.** yellow_leaf_curl_virus	LeNet	90-92%
			AlexNet	92-97%
Classification of tomato leaf diseases using MobileNet V2	2020	**1.** healthy **2.** late_blight **3.** leaf_mold **4.** mosaic_virus	MobileNet V2	94%
Tomato crop disease classification using pre-trained deep learning algorithm	2018	**1.** Tomato_healthy **2.** late_blight **3.** leaf_mold **4.** spider_mites_two-spotted_spider_mite **5.** target_spot **6.** mosaic_virus **7.** yellow_leaf_curl_virus	VGG-16 net	97.29%
			AlexNet	97.49%

The detection and diagnosis of leaf diseases in agriculture have been significantly improved with the adoption of computer vision and deep learning

techniques. Traditional methods have limitations in terms of subjectivity and time-consuming manual processes (Dawod & Dobre, 2022). Deep learning models offer promising solutions by automating the feature extraction and classification tasks (Barbedo, 2018; Huang et al., 2017). These technological advancements enable early identification and accurate diagnosis, leading to more effective management and treatment of leaf diseases, thus enhancing agricultural sustainability.

CNN and DenseNet-121

Convolutional Neural Networks (CNNs) have become the cornerstone of many computer vision applications due to their ability to learn hierarchical representations from raw image data. CNNs consist of multiple layers of neurons, including convolutional layers, pooling layers, and fully connected layers, each designed to extract and process features at different levels of abstraction. Convolutional layers are responsible for detecting patterns in the input data through convolution operations, capturing features such as edges, textures, and shapes. Pooling layers down sample the feature maps, reducing their spatial dimensions while retaining important information. Fully connected layers at the end of the network integrate these features to make predictions or classifications. The hierarchical structure of CNNs enables them to learn increasingly complex features from the input data, making them highly effective in tasks such as image classification, object detection, and image segmentation (LeCun et al., 1998).

DenseNet-121, also known as Densely Connected Convolutional Network, is a deep convolutional neural network architecture that has gained popularity in various image classification tasks. The key critical innovation of DenseNet-121 lies in its dense connections, which allow for direct connections between all layers within the network (Huang et al., 2017). Unlike traditional CNNs, where information flows sequentially from one layer to the next, DenseNet-121 enables direct access to the feature maps of all preceding layers. This dense connectivity enhances gradient flow, encourages feature reuse, and allows the model to capture more intricate patterns and details.

The architecture of DenseNet-121, illustrated in **Figure 1**, consists of several dense blocks, each composed of multiple convolutional layers with batch normalization and activation functions (Huang et al., 2017). The feature maps of all preceding layers are concatenated with each dense block, creating a dense connectivity pattern. This design enables feature reuse and facilitates the propagation of gradients, addressing the vanishing gradient problem often encountered in deep neural networks.

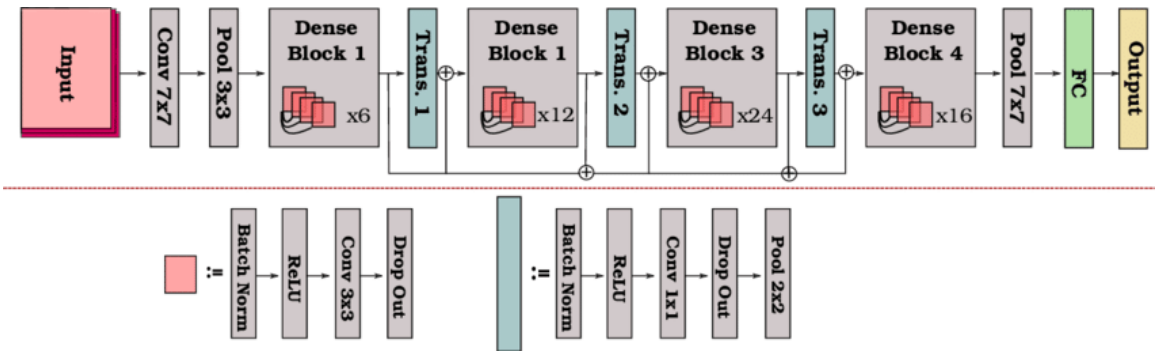


Figure 1. DenseNet-121 Architecture

DenseNet-121 has shown impressive performance in various image classification tasks, including leaf disease classification in agriculture (Barbedo, 2018; Mohanty et al., 2016). Its dense connectivity allows the model to extract and propagate features efficiently, leading to improved accuracy. Moreover, DenseNet-121 can be trained with transfer learning, utilizing pre-trained models on large-scale datasets such as ImageNet (Hussain et al., 2018). This transfer learning approach enables the model to use knowledge learned from a vast amount of data and generalize well to new tasks with limited training data. **Figure 2** presents the basic idea behind transfer learning.

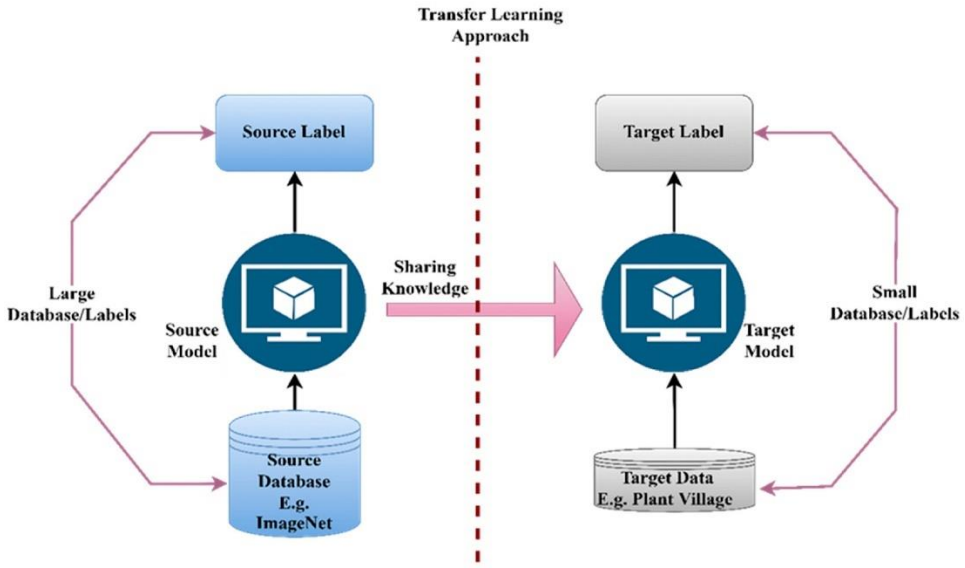


Figure 2. Basic Idea behind Transfer Learning

In their study, Too et al. (2019) compared different deep learning architectures for image-based plant disease classification. They evaluated architectures such as VGG 16, Inception V4, ResNet with 50, 101, and 152 layers, and DenseNet-121. The experiments were conducted on a dataset of 38 different classes of diseased and healthy leaf images from plantVillage. DenseNet-121

showed consistent improvement in accuracy with increasing epochs without signs of overfitting. It achieved a testing accuracy score of 99.75%, outperforming other architectures. The study highlighted the effectiveness of DenseNet-121 in achieving high accuracy with fewer parameters and reasonable computing time. Andrew et al. (2022) also focused on using deep learning techniques for efficient plant disease identification. They fine-tuned pre-trained CNN models, including DenseNet-121, ResNet-50, VGG-16, and Inception V4. The experiments were conducted on the PlantVillage dataset, which consisted of 54,305 image samples of different plant disease species. DenseNet-121 achieved a classification accuracy of 99.81%, surpassing other models. The study emphasized the potential of DenseNet-121 in accurately identifying plant diseases and its superiority over other models in terms of classification accuracy.

Table 3 presents a detailed summary of CNN Models used in the study entitled “Deep Learning-Based Leaf Disease Detection in Crops Using Images for Agricultural Applications,” covering all 38 classes in the PlantVillage dataset.

Table 3. Summary of CNN Models for Deep Learning-Based Leaf Disease Detection in Crops

Study	Year of Publication	Plant Type	Number of Classes	Model	Accuracy
Deep Learning-Based Leaf Disease Detection in Crops Using Images for Agricultural Applications	2022	Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, and Tomato	38	DenseNet-121	99.81%
				ResNet-50	98.73%
				Inception V4	97.59%
				VGG-16	82.75

DenseNet-121 is a powerful deep learning architecture for image classification tasks. Its dense connectivity promotes feature reuse and gradient flow, enabling the model to capture intricate patterns and achieve high accuracy. Using transfer learning further enhances its performance, especially when training data is limited. DenseNet-121 has emerged as a valuable tool for strengthening agricultural sustainability by improving the diagnosis and management of leaf health.

Conceptual Framework

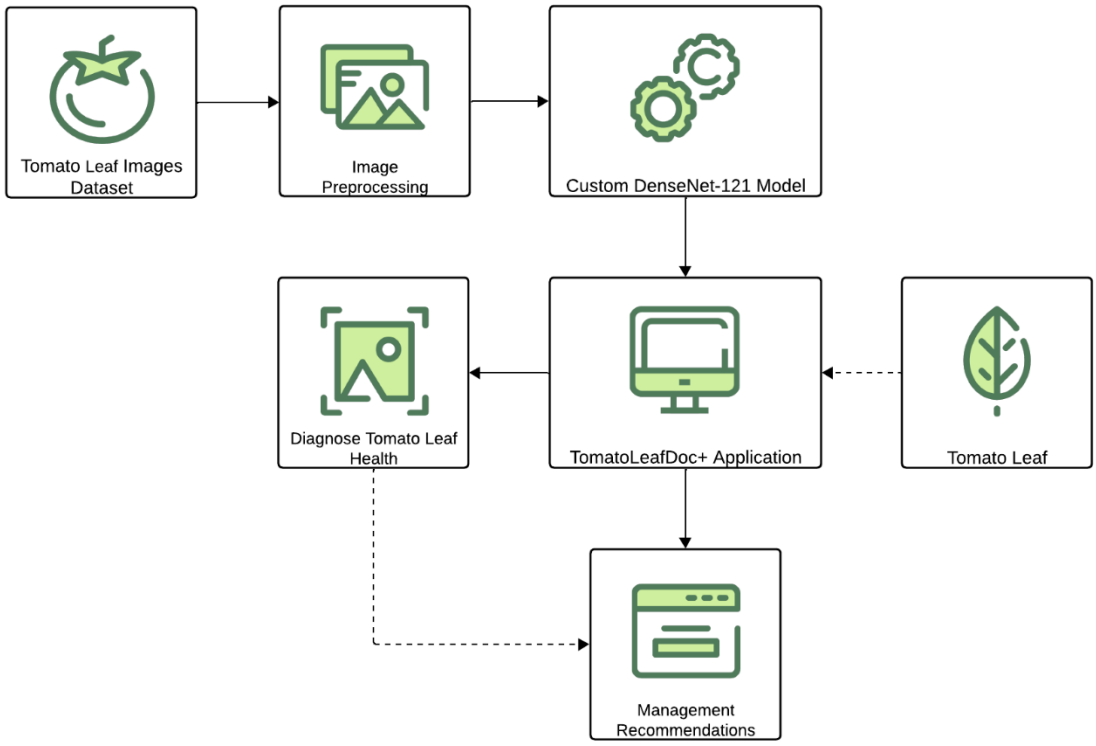


Figure 3. Conceptual Framework of the Study

This conceptual framework outlined a sequential process for this study. The first step was acquiring a dataset of tomato leaf images, including healthy leaves and leaves affected by various diseases. These images served as the basis for training and evaluating the CNN-based DenseNet-121 model.

Next, preprocessing techniques were applied to the dataset to ensure its suitability for analysis. This involved tasks such as resizing the images, dividing them into training, validation, and testing subsets, and augmenting the dataset.

Once the dataset was prepared, the CNN-based DenseNet-121 model was implemented. This deep learning model was trained using the prepared dataset to classify tomato leaf health. After training, the model was integrated into a web-based application developed to provide an interface for uploading leaf images and receiving health diagnoses. The application also incorporated a management recommendation, suggesting appropriate strategies for mitigating the identified tomato leaf health problems.

This process aimed to enhance agricultural sustainability by diagnosing tomato leaf health and guiding farmers or agricultural workers in implementing appropriate management strategies, ultimately contributing to improved crop yield and reduced economic losses.

Definition of Terms

The researchers identified several terms that required concise and precise explanations. The list included:

Convolutional Neural Network (CNN). A type of deep learning model designed specifically for image classification tasks, using convolutional layers to capture patterns and features from images.

Deep Learning. A subset of machine learning that utilizes artificial neural networks with multiple layers to automatically learn and extract features from data, particularly suitable for complex tasks like image recognition.

DenseNet-121. A specific architecture of Convolutional Neural Networks that employs dense connections between layers, facilitating better feature reuse and gradient flow, leading to improved accuracy in image classification tasks.

Image Classification. A computer vision task that categorizes images into predefined classes based on visual content, often using machine learning to extract features automatically.

Leaf Health. The condition of a plant leaf, which can indicate the overall health of the plant, including factors such as disease presence, nutrient status, and environmental stress.

Leaf Health Classification. The process of evaluating and categorizing the condition of plant leaves based on visual features, often employing machine learning algorithms to automate the detection of healthy versus unhealthy leaves.

Tomato. A juicy and tasty fruit widely cultivated and consumed in various cuisines globally, known for its healthful benefits, rich in vitamins and minerals.

Transfer Learning. A technique in deep learning where a pre-trained model is adapted to a new task by using knowledge learned from a different but related task.

Chapter 3

METHODOLOGY

This chapter presented a project design essential for outlining the specific methods and procedures to conduct the study.

A. Project Design

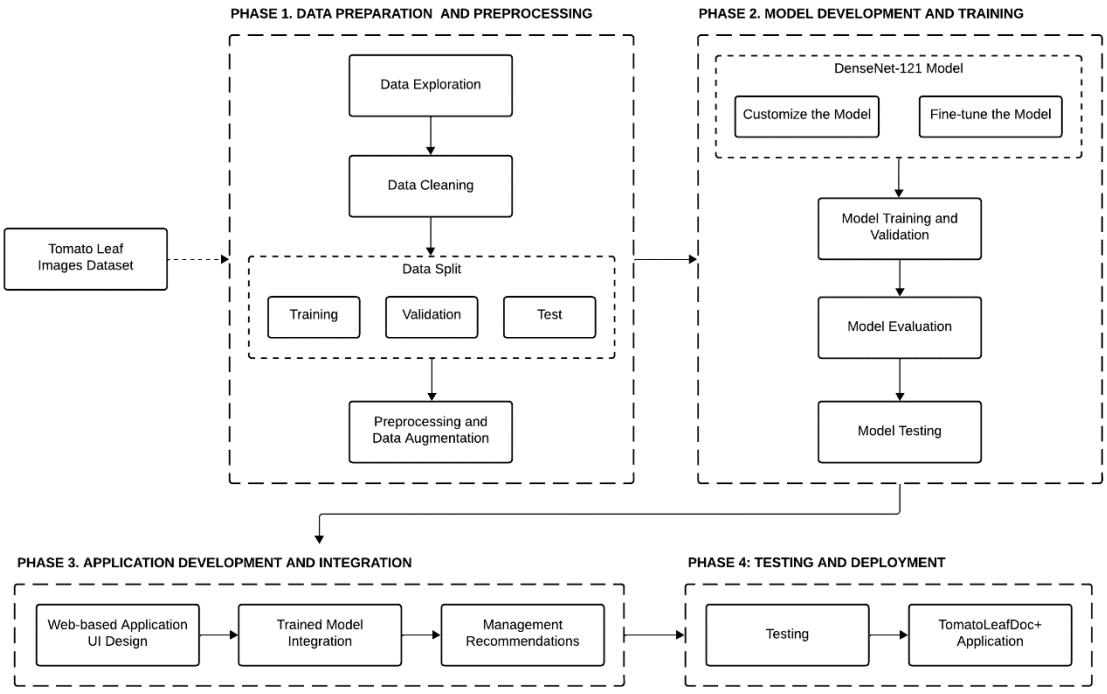


Figure 4. Project Design of the Study

Phase 1: Data Preparation and Preprocessing

The first phase focused on preparing the necessary data for the tomato leaf health classification application. Data acquisition involved gathering relevant images and information of tomato leaf health from different sources, including Kaggle and Mendeley Data. Subsequently, the acquired data underwent exploration to understand its characteristics and intricacies. Data cleaning followed

data exploration to rectify inconsistencies and enhance its quality. Tasks within this phase included splitting the dataset into subsets for training, validation, and testing purposes, resizing leaf images to a uniform dimension, and performing data augmentation to generate diverse instances. By properly preparing the data, it became ready for the succeeding phases of the project.

Phase 2: Model Development and Training

The second phase focused on developing and training the tomato leaf health classification application model. The chosen model architecture for this project was DenseNet-121, a deep learning model known for its effectiveness in image classification tasks. This model was implemented and trained using the prepared dataset. During training, the model's internal parameters were adjusted repeatedly, allowing it to learn and improve its ability to identify plant leaf health accurately. Validation was performed alongside training to assess the model's performance and make necessary adjustments. Once training and validation were complete, the model moved into the evaluation stage, where its performance was measured using appropriate evaluation metrics by comparing its predictions with the correct labels for the leaf health, thus incorporating model testing into the process. This part allowed the researchers to assess how well the model accurately identified leaf health.

Phase 3: Application Development and Integration

The third phase focused on developing the tomato leaf health classification application. This involved creating a user interface (UI) design for a web-based application that provided an intuitive platform for users to interact with the system. The trained DenseNet-121 model was then integrated into the application, allowing it to receive input images and provide health predictions using the integrated model. Additionally, a management recommendation system was developed within the application, utilizing the predicted tomato leaf health to offer strategies, including organic and chemical control, for effectively managing and mitigating the predicted health problems.

Phase 4: Testing and Deployment

The fourth and final phase focused on testing and deploying the developed application. Model testing was conducted to ensure the usability and accuracy of the integrated model within the web-based application. Various input images assessed the model's health prediction and management recommendation capabilities, ensuring its trustworthiness and effectiveness. Once the testing phase was completed, the tomato leaf health classification application was deployed, making it available for users to access and utilize on the web. By reaching this stage, the project was ready to provide farmers and agricultural workers with a useful tool for identifying tomato leaf health and receiving management recommendations, ultimately enhancing agricultural sustainability.

B. Project Development

A comprehensive discussion of the various phases involved in acquiring the project design.

a. Software Installation

This section of the project development outlined the software application and compiler used for this study.

Visual Studio Code. Microsoft's own Visual Studio Code is an effective yet lightweight source code editor with a wide range of extensions and features, including intelligent editing and an integrated terminal for debugging.

Anaconda Python. An extensive Python distribution with many pre-installed libraries and tools designed for data science and machine learning.

b. Package Installation

This section of the project development outlines the necessary dependencies and libraries used for this study.

Jupyter Notebook. Jupyter Notebook is an online application for generating and sharing documents with live code, text, and visualizations.

OpenCV. OpenCV stands out as an exceptional tool for image processing and executing computer vision assignments. As an open-source library, it offers a wide array of functionalities such as face detection, object tracking, landmark detection, and numerous others. Its versatility extends across multiple programming languages, encompassing Python, Java, and C++.

Pandas. Pandas is a Python library for data manipulation and analysis, offering tools for handling structured data efficiently.

Matplotlib and Seaborn. Python users who want flexible 2D charting should use Matplotlib. Seaborn, which is built on top of Matplotlib, provides simplified statistical visualization, which makes creating informative plots and intricate data analysis a breeze.

Numpy. A Python library that provides a wide range of mathematical operations and effective handling of arrays and matrices for quick numerical computation.

TensorFlow and Keras. TensorFlow and Keras are seamlessly connected to provide quick model development and training. Keras is a high-level neural network API aimed for simplicity and fast experimentation.

scikit-learn. Sklearn is a Python module used for statistical modeling and machine learning models. Various machine learning models for regression, classification, clustering, and statistical tools for assessing these models can be implemented with scikit-learn.

Pillow (PIL). Stand as the foundational Python libraries designed for image manipulation. Despite the existence of alternative Python libraries catering to image processing needs, Pillow retains its significance as a fundamental tool for comprehending and managing images.

Flask. Flask is a compact web framework for Python that makes web application development easier by offering a flexible and minimalist method for handling HTTP requests, routing, and template rendering.

Tomato Leaf Images Dataset. In this study, the researchers utilized the "PlantVillage," a publicly available dataset developed and maintained by researchers at Penn State University particularly, led by Dr. David Hughes. Despite the early use of machine learning for plant health and disease identification in 2007 (Huang, 2007), further studies were hindered by the lack of large public datasets until the publication of PlantVillage in 2015 (Hughes & Salathe, 2015). The dataset contained single-leaf images categorized into ten classes, including a healthy leaf class and nine different diseased leaf classes. The leaves were photographed outdoors against a grey or black background on sunny or cloudy days after being removed from the plant. However, this study exclusively acquired the tomato plant. The tomato plant from this dataset consisted of a healthy leaf class and different diseased leaf classes, such as bacterial spot, early blight, late blight, leaf mold, septoria leaf spot, two-spotted spider mite, target spot, mosaic virus, and yellow leaf curl virus. **Figure 5** displayed sample images for healthy and diseased classes from the dataset on Kaggle and Mendeley Data.

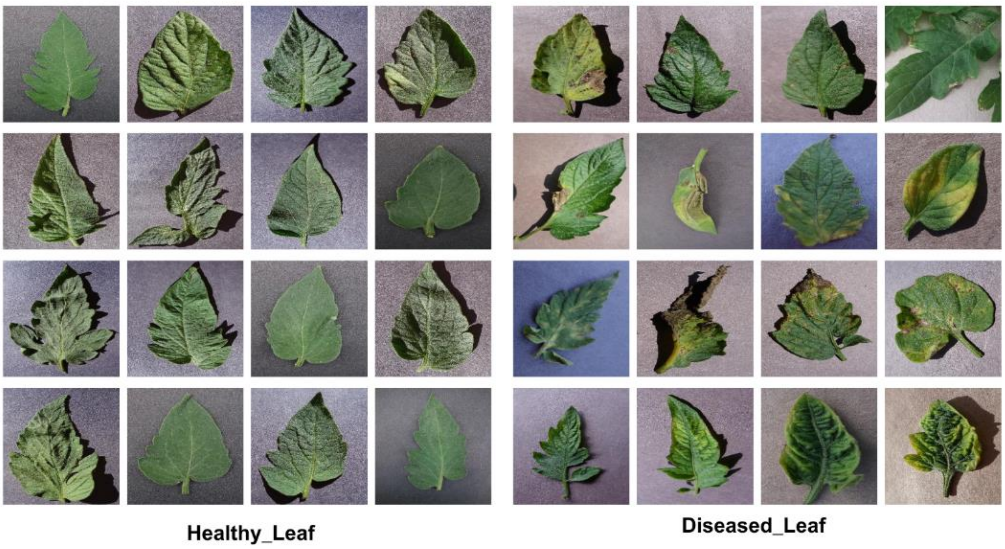


Figure 5. Sample Images from the PlantVillage Dataset for Tomato Leaf

In order to enhance the credibility, diversity, and quality of the dataset, the researchers expanded it by incorporating additional tomato leaf images sourced from tomato plants located in Quezon City, Philippines. These new images were captured using two different smartphone cameras: a 13-megapixel rear camera (Samsung Galaxy J7 2016) with a resolution of 4128 x 3096 pixels and a 12-megapixel rear camera (Samsung Galaxy Note 8 2017) with a resolution of 4032 x 3024 pixels. This addition not only broadened the geographical scope of the dataset but also enriched it with images captured under varying environmental conditions and using different photographic equipment. **Figure 6** showed the location of tomato plants in Quezon Memorial Circle, Quezon City, offering insight into the setting where these additional images were acquired.

Figure 6. Location of Tomato Plants in Quezon City (May 2024)

Phase 1: Data Preparation and Preprocessing

Data Exploration. During this critical step, the researchers dived into the intricate details of their dataset, aiming to uncover insights that would guide them in model development. They investigated the distribution of image counts across classes. They noticed that there was a big difference between the number of images in each class. Some classes had a lot of images, while others had very few. This difference in image counts highlighted that the dataset was imbalanced. Furthermore, they investigated the uniformity of image shapes, discovering a consistent dimensionality across all image features.

Data Cleaning. The researchers looked for any images that were damaged or couldn't be read. Fortunately, none of the images were corrupted or unreadable. Then, they ensured that all the images were in the same format, which helped

maintain consistency and simplify further data processing. This step helped keep everything organized and made it easier to work with the data later on.

Data Split. The dataset held two classes; it consisted of one (1) healthy leaf class and one (1) diseased leaf class. The researchers split this dataset into training samples, validation samples, and testing samples. The DenseNet-121 model was trained with 80% of the dataset; 10% was used for validation, and the other 10% was used for testing, as detailed in **Table 4**.

Table 4. Image Count Distribution by Dataset Type

Dataset Type	Image Count
Training	640
Validation	80
Testing	80
Total	800

Preprocessing and Data Augmentation. For this study, the researchers used the color images from the dataset, as they fit well with the transfer learning model. For DenseNet-121, the input size was 224 × 224 × 3 (height, width, and channel depth), so in order to meet the input requirement of the DenseNet-121 model, all images were rescaled to 224 × 224 pixels. The images matched the real-life images captured by farmers using different image acquisition techniques, such as Kinect sensors, high-definition cameras, and smartphones. To overcome overfitting, regularization techniques, such as data augmentation after preprocessing, were introduced. The augmentation processes used with the preprocessed images included rescaling, clockwise and anticlockwise rotation, shifting, shearing, and zooming, horizontal and vertical flipping, and filling mode adjustment. The images were not duplicated but augmented during the training

process, so the physical copies of the augmented images were not stored but were temporarily used in the process. This augmentation technique not only prevented the model from overfitting and model loss but also increased the strength of the model so that, when the model was used to classify real-life plant health images, it could classify them with better accuracy.

Phase 2: Model Development and Training

DenseNet-121 Model. The pre-trained DenseNet-121 model was chosen based on its applicability to the plant health classification task. The researchers custom-built this deep-learning model to use its capabilities in feature extraction and classification tasks. DenseNet-121 was suitable for this application due to its dense connectivity pattern, which allowed for more efficient information flow throughout the network and better feature reuse, leading to improved performance.

The researchers fine-tuned the pre-trained DenseNet-121 model on their dataset of healthy and diseased leaf images. The transfer learning model had the advantage of learning faster than models developed from scratch, as well as the ability to freeze layers and train the final layers for more accurate categorization. Initially, hyperparameter standardizations for several pre-trained models were carried out. **Table 5** showed the details of the hyperparameter specifications and configurations.

Table 5. Hyperparameter Specifications and Configurations

Attribute	Value
Image Size	224
Channel	3
Batch Size	32
Epoch	30
Regularization	Batch Normalization
Dropout	0.5
Number of hidden layers	2
Number of neurons per layer	1024, 512
Activation	ReLU
Number of Classes	2
Output Layer Activation	softmax
Optimizer	Adam
Learning Rate	0.001
Loss	categorical cross-entropy
Model Checkpoint, CSV Logger, Learning Rate Scheduler, Early Stopping	Yes

These parameters were crucial as they directly impacted the model's learning process and ultimate performance. The image size, set to 224 pixels, ensured consistency in input dimensions, while the three channels corresponded to the standard RGB format. A batch size of 32 dictated the number of samples processed in each iteration, influencing computational efficiency and memory usage.

The regularization technique adopted was Batch Normalization, which helped stabilize and accelerate the training process by normalizing the activations within a layer. A dropout rate of 0.5 was implemented to mitigate overfitting, randomly deactivating neurons during training to promote model generalization. The model architecture consisted of two hidden layers, with 1024 and 512 neurons, respectively, each utilizing the Rectified Linear Unit (ReLU) activation function to introduce non-linearity. This was depicted by the formula in **Eq. 1**.

Eq. 1. ReLU Formula

$$f(x) = max\ (0, x)$$

With 11 classes to predict, the output layer employed softmax activation, generating probabilities for each class. This was expressed by the formula in **Eq. 2**.

Eq. 2. Softmax Formula

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

The Adam optimizer, known for its adaptability and efficiency, was utilized with a learning rate of 0.001 to adjust model parameters during training. Categorical cross-entropy served as the loss function, evaluating the disparity between predicted and actual class distributions. This was represented by the formula in **Eq. 3**.

Eq. 3. Categorical Cross-entropy Formula

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

Additionally, several training enhancements were added, including model checkpointing, CSV logging, a learning rate scheduler, and early stopping, facilitating efficient model monitoring and management throughout the training process.

Model Training and Validation. The model.fit() method began to initiate the training process. The training and testing generators held the improved images, the number of epochs to loop through the model, the number of validation

steps, and the number of steps per epoch among the arguments required by this approach. Callbacks were also used to monitor and record the model's training progress. Based on validation loss, the ModelCheckpoint callback made sure that the best training model was kept in the assigned model path variable. After a certain amount of patience iterations, the EarlyStopping function stopped the training process if there was no improvement in validation loss. Finally, training history data was recorded in CSV format to the designated history path variable using CSVLogger.

In order to determine if the model's adaptation to the training data was optimal or unsatisfactory, validation took place in parallel with training. Callbacks became relevant, and the ModelCheckpoint() function was utilized to guarantee the production of the best possible training result. By carefully examining the accuracy of the validation procedure and storing the best outcome, this function met its goal. The number of epochs was fixed at 30, and the learning rate remained constant at 0.001.

The performance metrics across the specified number of epochs were plotted in the output plots, which facilitated the evaluation of the model's validation and training progress.

Model Evaluation. The performance of the model was evaluated through various metrics, including classification accuracy, sensitivity, specificity, and F1 score. These metrics gave insight into multiple aspects of the model's

performance, ranging from overall correctness to its capacity to accurately detect positive and negative instances, resulting in a thorough assessment of its predictive capabilities.

Accuracy was a measure of the overall correctness of the model's predictions, calculated as the ratio of correct predictions to the total number of predictions made. The total number of two correct predictions (TP + TN) was computed by dividing the complete dataset number (TP + TN + FP + FN) as shown in **Eq. 4**.

Eq. 4. Accuracy Formula

$$\text{accuracy} = \frac{TP + TN}{TP + FN + TN + FP}$$

Specificity indicated the model's ability to correctly identify negative instances, such as unhealthy plants, in this context. It measured the proportion of truly negative cases that were identified as negative, as well as the rate of false positives, where unhealthy plants were incorrectly classified as healthy. The ratio of correctly classified negative instances (TN) to the total number of negative instances (TN + FP) was depicted in **Eq. 5**.

Eq. 5. Specificity Formula

$$\text{specificity} = \frac{\text{True Negative}}{(\text{True Negative} + \text{False Positive})}$$

Sensitivity, also known as recall, indicated the model's capacity to correctly detect positive instances, like healthy plants in this scenario. It calculated the ratio of truly positive cases that were identified as positive to the total number of positive

cases, as well as the rate of false negatives, where healthy plants were wrongly classified as unhealthy. The ratio of correctly classified positive classes (TP) to the total number of positive classes (TP + FN) was described in **Eq. 6**.

Eq. 6. Sensitivity Formula

$$\text{sensitivity} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Negative})}$$

The F1 score is the harmonic mean of precision (TP / (TP + FP)) and recall (TP / (TP + FN)). It provides a balance between the two metrics, giving an overall measure of a model's accuracy, particularly when there's an uneven class distribution. It is calculated as expressed in **Eq. 7**.

Eq. 7. F1 Score Formula

$$\text{F1 Score} = \frac{2 \text{ x Precision x Recall}}{\text{Precision} + \text{Recall}}$$

Model Testing. The testing guaranteed optimum performance, the trained model's best weights were loaded, and a data generator was prepared for the test dataset. This generator processed the test images and batched them for efficient testing. The model then predicted the class labels for the test data, converting the predicted probabilities into class labels.

Predictions

These predictions were compared against the true class labels obtained from the test generator. By evaluating the model's performance on unseen test data, this process assessed its ability to generalize to new images and accurately classify them according to predefined classes.

Tomato Leaf Health Classification

To begin the evaluation procedure, the tomato leaf health classification model's history was loaded. The results obtained during each training epoch were stored in a variable called the model's history. Initially, the researchers plotted the results of the model's history using the plt.plot() function.

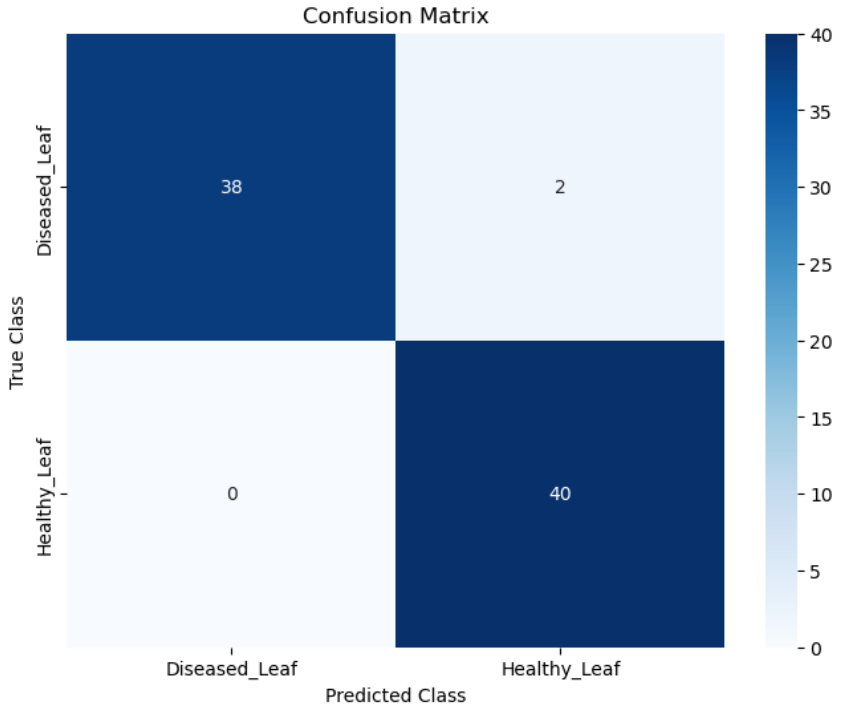


Figure 7. Confusion Matrix

A confusion matrix was employed using scikit-learn's confusion_matrix() function to indicate the standard for the predictions that were produced. To assess the effectiveness of the classification model, it compiled a summary of accurate and wrong predictions. **Figure 7** displayed the confusion matrix the researchers were able to get.

Following the generation of the confusion matrix, a classification report was also employed using scikit-learn's classification_report() function to produce a complete report on the model's performance, including metrics such as precision, recall, and F1-score for each class. After that, a custom function calculated the specificity (true negative rate) for each class, and the results were printed out. Additionally, the macro and the weighted average for all metrics across all classes were calculated and displayed, providing further insights into the model's overall performance.

Phase 3: Application Development and Integration

Web-based Application UI Design. A web application was developed using Visual Studio Code, a popular code editor known for its versatility and ease of use. This application enabled simple navigation and interaction, allowing users to upload an image of a tomato leaf for diagnosis and receive treatment recommendations promptly. The UI design ensured that users could easily access the information they needed. Using HTML, CSS, and JavaScript, a cohesive design with a layout that supported usability and engagement was developed, incorporating interactive elements and navigation menus. The application featured a responsive design using Bootstrap and utilized Flask for backend functionality, ensuring it effectively met the needs of its users. The application's UI design is detailed in *Appendix G, User's Manual*.

Trained Model Integration. The integration demonstrated an innovative combination of web-based technologies and machine learning to address

significant issues in tomato farming. Using uploaded leaf images by the user and the ability of the DenseNet-121 architecture, which is based on Convolutional Neural Networks (CNNs), the system accurately classified tomato plant health. Farmers could easily engage with the model by uploading photos for immediate analysis using the Flask web framework. After processing an image, the model offered a precise diagnosis, identifying symptoms that were important for initiating a response to control plant's health.

The system provided farmers with comprehensive information on symptoms and management recommendations, including chemical and natural alternatives, getting further basic diagnoses to provide useful findings. With the help of a thorough approach, farmers could minimize crop losses, promote environmentally friendly farming methods, and efficiently address diseases that had been diagnosed. This solution not only increased the effectiveness of health management in tomato growing but also offered farmers the tools they needed to maximize profitability and encouraged sustainable agriculture by integrating modern technology with user-friendly web interfaces.

Management Recommendations. In this study, the researchers utilized Plantix's library of management recommendations for crop health encompassing both organic and chemical methods. This library served as a reliable resource for farmers and agriculturists as it was developed in collaboration with leading agronomists and backed by extensive research. The data for management recommendations used in the study is stored in *Appendix H, Dataset*. These

management recommendations were integrated into the Flask web application. The application received an image file through a POST request and processed it to make a prediction. Based on the predicted health label, the application retrieved the corresponding management recommendations from a predefined dictionary “management.” For each health label, the management recommendations were structured into three components: Symptoms, Organic Control, and Chemical Control. These components were structured as a list within the management dictionary entry for each health label. The Flask application then checked if the predicted health label corresponded to a category that had all three management components available. If so, it retrieved the symptoms, organic control, and chemical control recommendations from the management dictionary. The Flask application rendered an HTML template with the predicted health label, the symptoms description, and the organic and chemical control recommendations. Additionally, it displayed the uploaded image along with the result. This integration allowed users to upload an image of a tomato plant and receive management recommendations based on the predicted health.

Phase 4: Testing and Deployment

Testing. A thorough evaluation of the developed application was carried out to ensure its functionality and reliability. This involved testing the application using a variety of input images representing tomato leaf healthy and diseased. Through these tests, the integrated model's health prediction and management recommendations capabilities were comprehensively evaluated to guarantee their

efficacy in real-world scenarios. This testing helped identify and rectify any issues before deployment, ensuring the application met the desired standards of usability and effectiveness. **Figure 8** showcased the actual tomato leaves sourced directly from the tomato plants, which were utilized as input during the system testing.

Figure 8. Actual Tomato Leaves Utilized for System Testing

These leaves served as crucial inputs during system testing, contributing to the validation and refinement of the application's performance, thereby enhancing its accuracy and reliability in diagnosing tomato leaf diseases.

TomatoLeafDoc+ Application. TomatoLeafDoc+ was a web-based application designed to assist farmers and agricultural workers in classifying tomato leaf health and receiving management recommendations. The application integrated a model trained on a diverse dataset of tomato leaf images to classify health accurately and provide tailored recommendations for effective management. The interface of TomatoLeafDoc+ enabled easy uploading of

images and seamless navigation through the diagnosis and management process. TomatoLeafDoc+ aimed to empower users with a valuable tool for enhancing agricultural sustainability.

Chapter 4

RESULTS AND DISCUSSIONS

This chapter presented the findings of the study and discussed their implications in relation to the research objectives.

The researchers aimed to achieve the study's **first specific objective**, "*Applied several preprocessing techniques to improve the quality and diversity of the tomato leaf images dataset for training and validating the model*," by gathering and looking over the needed tomato leaf images. Various methods of preprocessing were then used to guarantee that the model collected accurate data. The training, testing, and validation data were separated into different folders as part of the data splitting procedure. Images in each class in the dataset were first randomized to guarantee randomness; training (80%), testing (10%), and validation (10%) were created based on predetermined ratios. In accordance with the specified splits, images were transferred from the original class directory to the appropriate split directories. The allocation of the dataset across the splits was subsequently analyzed by the code by counting the number of samples for each class in each split. The implementation of a systematic strategy guaranteed that the model was trained on a wide variety of examples and assessed on unique, unseen data to have an efficient model creation and evaluation for the classification of tomato leaf health. After completing this data split, **Table 6** displayed the number of images per class.

Table 6. Details of Dataset Split for Training, Validation, and Testing

Class Name	Total Samples	Training Samples	Validation Samples	Test Samples
Healthy_Leaf	400	320	40	40
Diseased_Leaf	400	320	40	40
Overall Total Samples	800	640	80	80

After gathering training, validation, and test samples, the researchers created an image data generator that generated batches of images dynamically during training rather than loading all of the images into memory at once. This helped to conserve memory and enabled larger datasets to be used in training. It also helped to enhance the dataset and improve model performance.

Table 7. Parameters of the Data Augmentation Techniques

Parameter	Value
Scaling	1./255
Rotation	40
Shearing	0.2
Zoom	0.2
Flipping	Vertical/Horizontal
Shifting	Width/Height with range 0.2
Filling	nearest

Table 7 listed parameters for data augmentation methods that were frequently applied in machine learning tasks. Pixel values were divided by 255 to scale images for normalization purposes. They could be rotated up to 40 degrees; images were randomly sheared by 2%, zoomed by 2%, then flipped both vertically and horizontally, and shifted up to 2% of their original proportions. Following transformations, the "nearest" filling technique was used to fill in any vacant spaces. By increasing the range of datasets, these augmentation strategies helped to train reliable and generalized models.

After applying the various preprocessing techniques, some images for each class are displayed in **Figure 9** below.

Figure 9. Sample Preprocessed Images

To achieve the study's **second specific objective**, "*Develop a customized DenseNet-121 model for tomato leaf health image classification, evaluating its performance through classification accuracy, sensitivity, specificity, and F1 score,*" the researchers built, trained, and evaluated a convolutional neural network (CNN) for image classification using the DenseNet121 architecture. They started by loading the DenseNet121 model, which was imported from the Tensorflow Keras library without its classification layer, as the goal was to customize the classification layers for a specific task rather than using the pre-trained classifier. This was done by setting include_top to False while loading the model. The model was loaded with weights pre-trained on the ImageNet dataset.

Next, custom classification layers were added on top of the base model. These custom layers included a global average pooling layer, which aggregated

the spatial information across the feature maps generated by the base model. Subsequently, batch normalization was applied to normalize the activations, stabilize the training process, and accelerate convergence. Following batch normalization, a dropout layer was introduced with a dropout rate of 0.5, randomly deactivating 50% of the neurons during training to prevent overfitting and promote model generalization. Two dense layers were also added, each with 1024 and 512 units, respectively, activated by rectified linear unit (ReLU) activation functions. These dense layers enabled feature extraction and higher-level representation learning from the pooled features. Another batch normalization layer followed, further enhancing stability, followed by another dropout layer with a dropout rate of 0.5. Finally, the output layer was added with the specified number of classes, employing a softmax activation function to produce class probabilities. After defining the custom classification layers, the final model was created by specifying the optimizer and assembling the model architecture. The Adam optimizer, an optimization algorithm that adapts learning rates for each parameter individually, was implemented with a learning rate set to the predefined value of 0.001.

Following this, the model was fine-tuned by unfreezing some layers of the base model. All layers except the last ten layers were frozen to prevent them from being updated during training. The last ten layers were frozen, as these deeper layers often capture more abstract and generalizable features, which are beneficial for transfer learning.

After the model definition, functions for calculating evaluation metrics such as precision, specificity, sensitivity (recall), and F1 score were defined using Keras backend functions. These metrics provided insights into the performance of the model beyond simple accuracy. The model was then compiled with an Adam optimizer, categorical cross-entropy loss function, and the defined evaluation metrics.

Paths for model checkpointing and logging were set up, along with callback functions for ModelCheckpoint, CSVLogger, LearningRateScheduler, and EarlyStopping. These callbacks enabled monitoring of the model's performance during training and took actions such as saving the best model, adjusting learning rates, and stopping training early to prevent overfitting.

Finally, the model was trained using the fit() method, which iteratively updated the model parameters based on the training data. The train_generator provided batches of training data, where the number of steps per epoch was determined by the total number of training samples divided by the batch size. The training process was carried out for a total of 30 epochs. Additionally, validation data from validation_generator was used to evaluate the model's performance after each epoch, with the number of validation steps similarly determined by the total number of validation samples divided by the batch size. During training, several callbacks were also employed.

Figure 10 showed the first three epochs of the training process for the tomato leaf health classification using the custom DenseNet-121 model. It also showed that the model was automatically saved once the validation accuracy improved.

```
Epoch 1/30
WARNING:tensorflow:From D:\anacondaa\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragge
d.RaggedTensorValue instead.

WARNING:tensorflow:From D:\anacondaa\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

481/481 [==============================] - ETA: 0s - loss: 0.7207 - accuracy: 0.7751 - precision: 0.8184 - specificity: 0.9841 - recall: 0.7400 - f1: 0.7758
Epoch 1: val_accuracy improved from -inf to 0.87865, saving model to model.hdf5
481/481 [==============================] - 1102s 2s/step - loss: 0.7207 - accuracy: 0.7751 - precision: 0.8184 - specificity: 0.9841 - recall: 0.7400 - f1: 0.7758 - val_l
oss: 0.3107 - val_accuracy: 0.8786 - val_precision: 0.8970 - val_specificity: 0.9901 - val_recall: 0.8615 - val_f1: 0.8784 - lr: 0.0010
Epoch 2/30
481/481 [==============================] - ETA: 0s - loss: 0.3798 - accuracy: 0.8715 - precision: 0.8922 - specificity: 0.9897 - recall: 0.8510 - f1: 0.8707
Epoch 2: val_accuracy improved from 0.87865 to 0.94948, saving model to model.hdf5
481/481 [==============================] - 1014s 2s/step - loss: 0.3798 - accuracy: 0.8715 - precision: 0.8922 - specificity: 0.9897 - recall: 0.8510 - f1: 0.8707 - val_l
oss: 0.1384 - val_accuracy: 0.9495 - val_precision: 0.9589 - val_specificity: 0.9959 - val_recall: 0.9464 - val_f1: 0.9525 - lr: 0.0010
Epoch 3/30
481/481 [==============================] - ETA: 0s - loss: 0.2875 - accuracy: 0.9025 - precision: 0.9173 - specificity: 0.9920 - recall: 0.8877 - f1: 0.9020
Epoch 3: val_accuracy improved from 0.94948 to 0.95156, saving model to model.hdf5
481/481 [==============================] - 1019s 2s/step - loss: 0.2875 - accuracy: 0.9025 - precision: 0.9173 - specificity: 0.9920 - recall: 0.8877 - f1: 0.9020 - val_l
oss: 0.1362 - val_accuracy: 0.9516 - val_precision: 0.9564 - val_specificity: 0.9957 - val_recall: 0.9479 - val_f1: 0.9521 - lr: 0.0010
```

Figure 10. A Snippet of the Custom DenseNet-121 Model Training Process

Model Evaluation

Subsequent to training, the researchers visualized the training and validation performance of the custom DenseNet-121 model over multiple epochs, as illustrated in **Figure 11**.

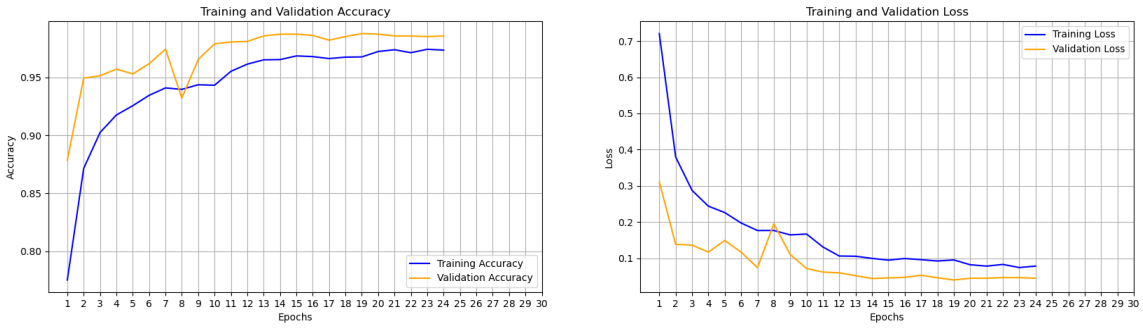


Figure 11. Learning Curve of the Custom DenseNet-121

The training process was carried out using a number of epochs set to 30. However, due to early stopping, the training was discontinued after epoch 24. Each epoch involved iterating over the full dataset in batches and changing the model's weights to minimize the specified loss function. Several measures were tracked throughout the training process, including accuracy, precision, specificity, recall, and F1-score, all of which indicated the model's performance.

The model's performance rapidly increased during the initial epochs, as indicated by increasing values of metric scores on both the training and validation sets. Notably, the validation accuracy peaked at around 98.8%, demonstrating that the model successfully generalized to unseen data. Also, the validation loss steadily decreased throughout epochs. However, at epoch 19, the validation accuracy plateaued while the validation loss began to fluctuate, indicating that the model's performance was stabilizing. As a result, the early stopping was activated, preventing overfitting by terminating the training process at epoch 24 when the model's performance on the validation set failed to improve.

This training process showed the model's ability to learn and improve over time, resulting in high performance on the validation dataset, as presented in **Table 8.** The early stopping strategy efficiently prevented overfitting and guaranteed that the model was terminated at an optimal point.

Table 8. Training and Validation Evaluation

	Train	Validation
Accuracy	0.9737 or 97.37%	0.9859 or 98.59%
Loss	0.0782 or 7.82%	0.0448 or 4.48%

During the researchers' data exploration, they found out that their data was imbalanced, so they just couldn't rely on accuracy. This was not the best metric for the model's performance as it might give vague results. That's why they used other metric scores, such as specificity, sensitivity, and F1 Score. **Table 9** showcases the metric scores obtained at the end of the training process. This suggested that the model performed well in distinguishing between classes and generalized effectively to unseen data.

Table 9. Specificity, Sensitivity, and F1 Score

	Train	Validation
Specificity	0.9977 or 99.77%	0.9988 or 99.88%
Sensitivity	0.9715 or 97.15%	0.9859 98.59%
F1 Score	0.9739 or 97.39%	0.9867 or 98.67%

Model Testing

Table 10. Classification Report

Accuracy	0.97		
Class Name	**Specificity**	**Sensitivity**	**F1 Score**
Healthy_Leaf	0.95	1.00	0.98
Diseased_Leaf	1.00	0.95	0.97
Macro Average	0.97	0.97	0.97
Weighted Average	0.97	0.97	0.97

The model testing results, as indicated in **Table 10**, demonstrate the classifier's performance across several classes. The overall accuracy of the model stands high at 99%. Each class is evaluated based on specificity, sensitivity, and F1 score, providing detailed insights into the model's capability. The model shows high specificity across all classes, indicating its ability to correctly identify true negative cases. Sensitivity differs slightly across classes but remains high overall, with a macro and weighted of 99%. The F1 score indicates the model's balanced

performance, with a macro and weighted average of 99%. This demonstrates good performance across multiple classes, implying that it can reliably classify tomato leaf health.

The confusion matrix analysis highlighted that there's almost a perfect diagonal line from left to right. This pattern indicates that the binary classification model performed exceptionally well in its predictions. Its ability to differentiate between healthy and diseased leaves is critical to the agricultural industry, highlighting its role in promoting crop health and productivity.

To achieve the study's **third specific objective**, "*Create a web-based application that utilizes the trained model to classify* tomato *leaf health based on images uploaded by users,*" the researchers developed a web-based application using a Flask framework to allow users to upload images from their gallery. The backend prepared this image after the user uploaded an image. Similar objective methods of preparation were applied afterward to the backend. Diagnoses were produced using the model.predict() function. Once the user successfully uploaded, a sweet alert prompted; after clicking the "ok" button, the result showed, as illustrated in **Figure 12**.

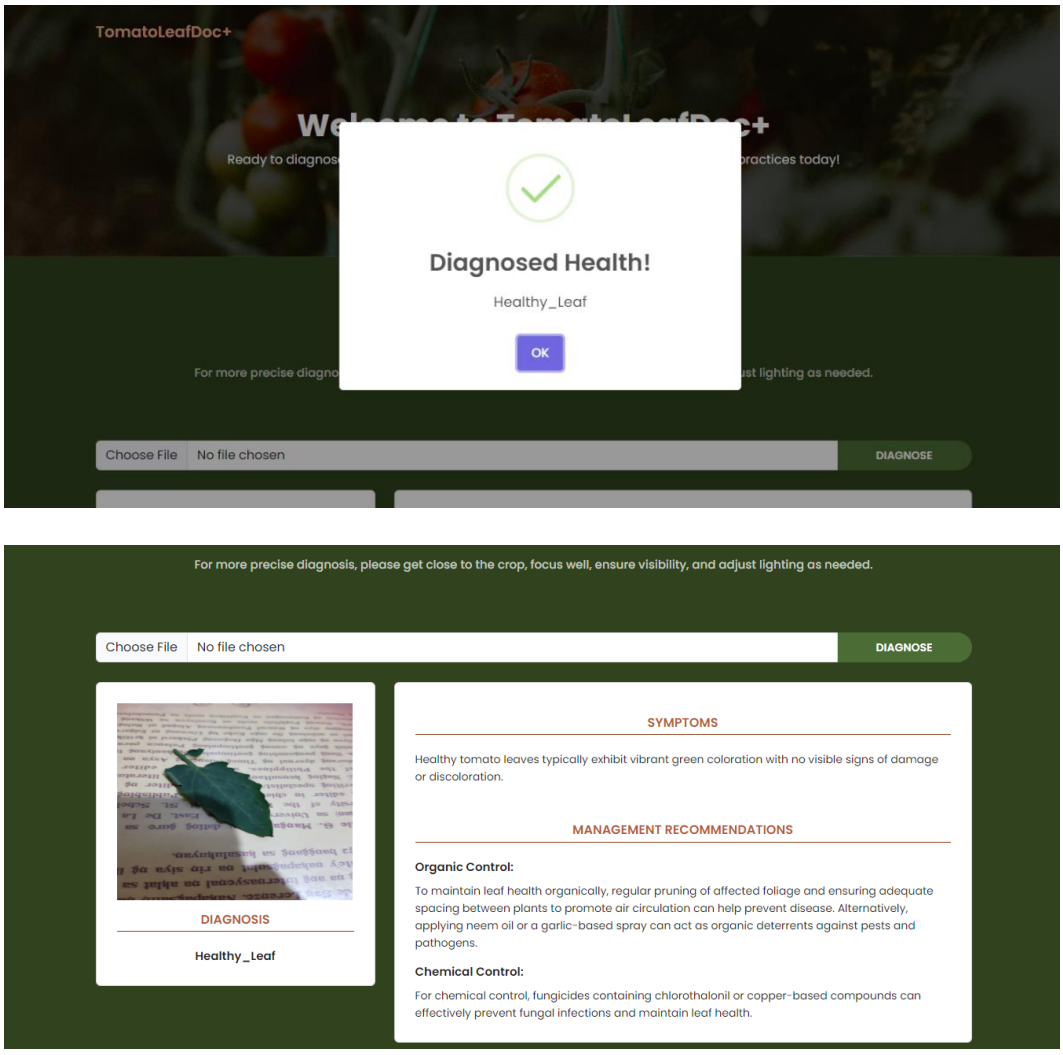


Figure 12. Application User Interface with Diagnosis Result

There were two buttons on the application's user interface: choose file and diagnose. The choose file button opened the device's file location to select an image for upload. The diagnose button initiated the analysis of the chosen image, providing a diagnosis once processed.

Testing

A comprehensive evaluation of the application was conducted to ensure its functionality and reliability in diagnosing tomato leaf health. This testing involved

using a variety of input images representing different classes of tomato leaf conditions, specifically healthy and diseased leaf. The tomato leaves shown in **Figure 13** were utilized as input for system testing, contributing to the evaluation of the application's performance.

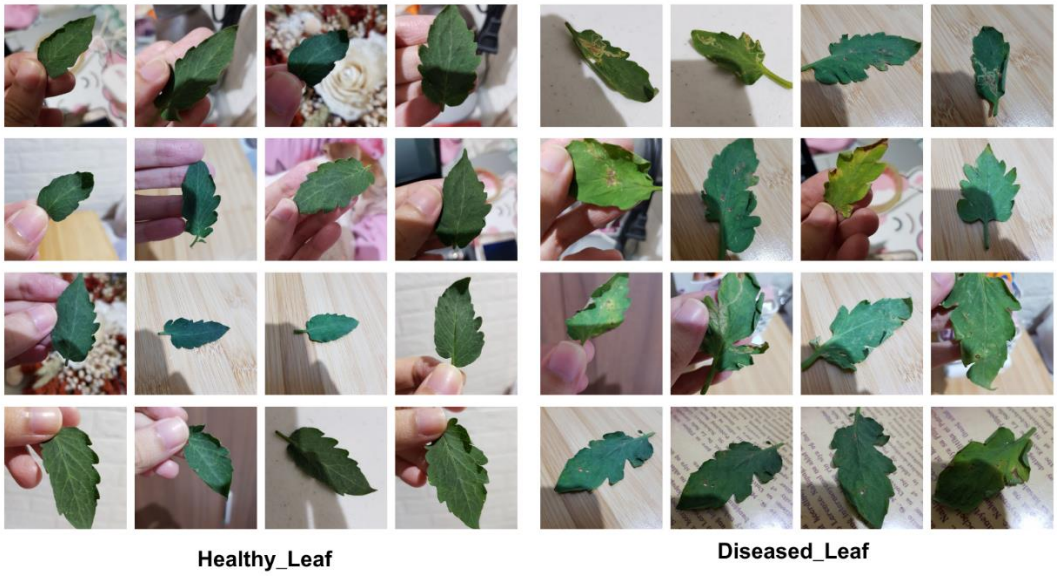


Figure 13. Sample Images Utilized for System Testing

During testing, fifty (50) images were used for each class to thoroughly evaluate the application's performance. The results are presented in **Table 11**.

Table 11. Application Performance Results

Classes Names	Correctly Identified	Total Tested	Accuracy
Healthy_Leaf	45	50	90%
Diseased_Leaf	47	50	94%

These results demonstrate the integrated model's reliable disease prediction capabilities. The overall accuracy rates of 90% for Healthy_Leaf and 94% for Diseased_Leaf indicate that the application is highly effective in real-world

scenarios. By thoroughly evaluating and refining the model based on these test outcomes, the researchers ensured that the application meets the desired standards of usability and effectiveness.

Tomato Leaf Health Detection

Aside from developing a web-based application that utilizes the trained model to classify tomato leaf health based on images uploaded by users, the researchers also implemented a real-time Tomato Leaf Health Detection system using a webcam. The process begins by loading necessary libraries (Keras for model handling, OpenCV for image capture, and NumPy for numerical operations) and the model along with class labels. The webcam captures images in real-time, which are then resized to 224 x 224 pixels, converted to a NumPy array, reshaped, and normalized. The model predicts the class of the tomato leaf, displaying the class name and confidence score, as depicted in **Figure 14**.

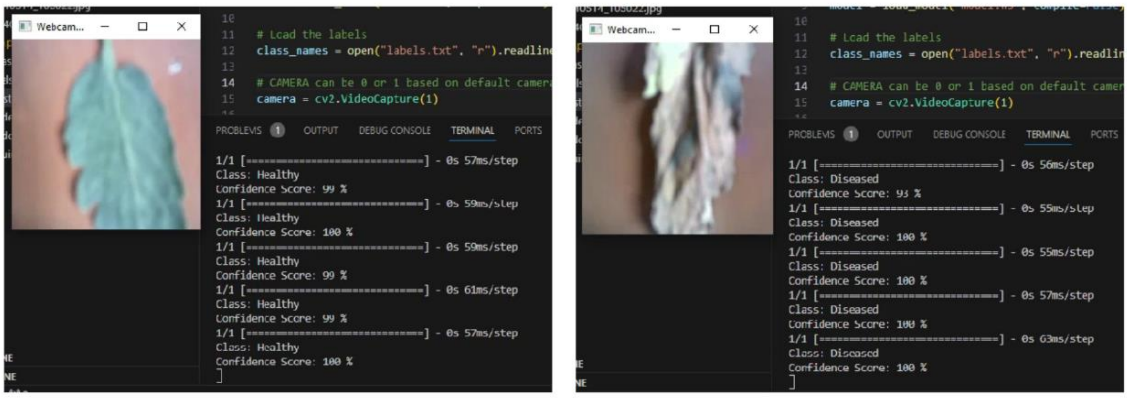


Figure 14. Tomato Leaf Health Detection

This provides a practical solution for real-time tomato leaf health detection, complementing the web-based application and enhancing the overall utility of the model in diverse scenarios.

The researchers aimed to achieve the study's **last specific objective**, "*Integrate management recommendations into the application, imposing the diagnosed health information to suggest appropriate management strategies,*" the researchers implemented a system within the Flask web application to integrate management recommendations for diagnosed plant health. Upon receiving an image of a tomato leaf, the application analyzed the image and predicted its health. Once the health was diagnosed, the application retrieved corresponding management recommendations from its predefined dictionary. These recommendations, which were divided into three components—Symptoms, Organic Control, and Chemical Control—were presented to the user in an organized way. The web-based application that was developed not only displayed the predicted health label but also provided a detailed description of the symptoms observed on the tomato plant. This ensured that users had a complete understanding of the diagnosed health and the corresponding management strategies recommended.

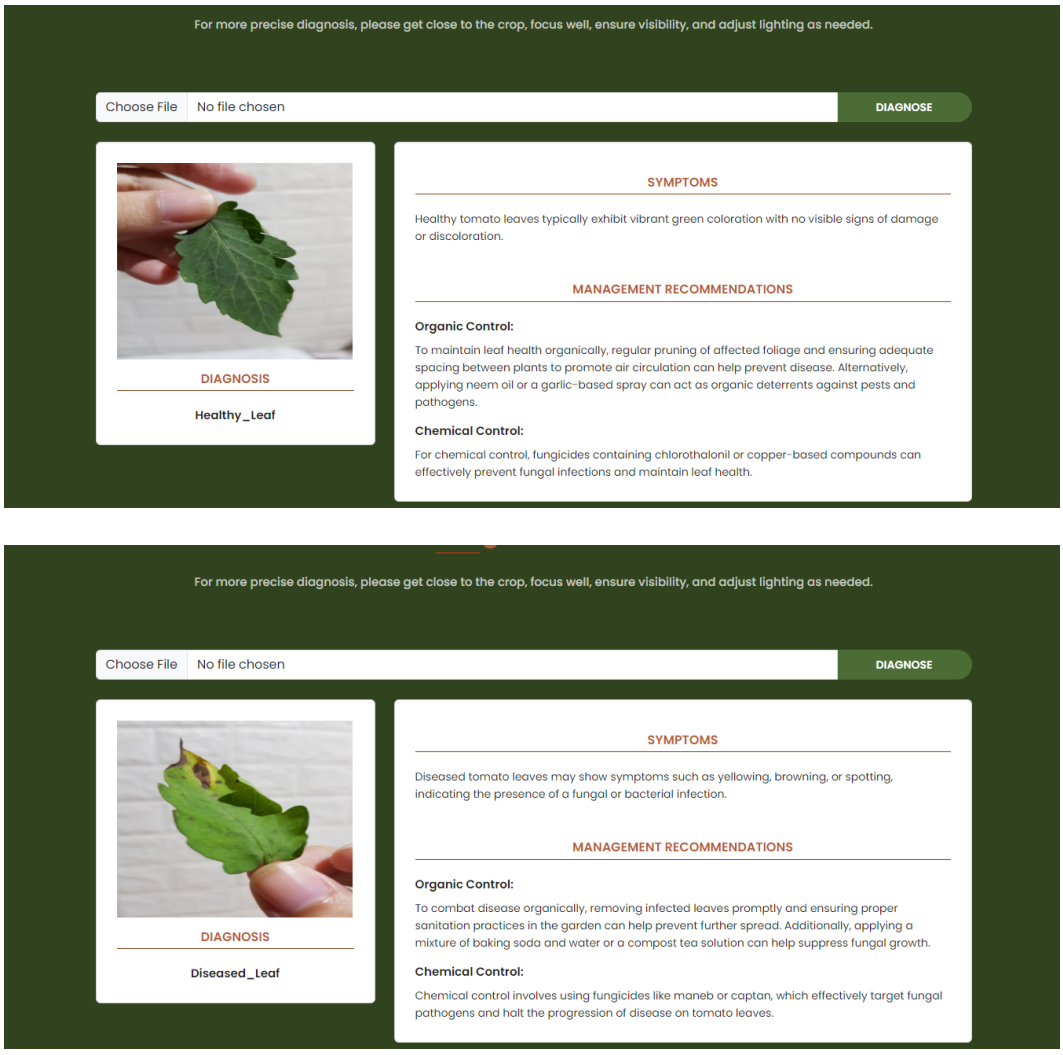


Figure 15. Application User Interface with Management Recommendations

The top section of **Figure 15** illustrates the outcome when a user submits an image of a healthy tomato leaf, while the lower section demonstrates the result from an image of a diseased tomato leaf. In the interface, the application not only identifies the presence of disease but also provides recommendations to maintain or improve leaf health. These suggestions encompass both organic and chemical approaches, offering users options to address issues such as pests, fungal, or bacterial infections, and promote overall leaf vitality.

Chapter 5

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

This chapter provides a summary of the key findings, conclusions drawn from the study, and recommendations for future research.

Summary

Despite tomatoes being economically and nutritionally valuable, they were prone to various leaf diseases, impacting yield, quality, and overall plant health, highlighting the need for early disease diagnosis to enable effective management. This study aimed to contribute to agricultural sustainability by creating a web-based application using a customized Convolutional Neural Network (CNN)-based DenseNet-121 model for tomato leaf health classification and management recommendations. The findings held potential benefits for a range of agriculture stakeholders, such as farmers and agricultural workers, consumers, and plant health authorities and agencies.

The process began with data preparation, involving acquiring images from the PlantVillage dataset, exploring the data to understand its characteristics, cleaning it to enhance quality, and performing tasks such as splitting, resizing, and augmenting the dataset. After that, the DenseNet-121 model was developed and trained, with a focus on standardizing hyperparameters and fine-tuning to achieve the best performance. It underwent iterative adjustments during training, validated alongside to assess performance, and then evaluated using classification

accuracy, specificity, sensitivity, and F1 Score to ensure accurate disease classification. Then, a web-based application was created, Visual Studio Code was utilized for the user interface, and Flask handled the backend functionality. The application integrated the trained DenseNet-121 model for predicting health from input images and included a management recommendation system based on predicted health to offer effective management strategies. Management recommendations, which encompassed both organic and chemical control, were sourced from Plantix's library. Lastly, the developed application underwent rigorous testing to verify its usability and accuracy before it was deployed, including assessments of health prediction and management recommendations using diverse input images.

This study concluded that the customized DenseNet-121 model, trained on a preprocessed dataset of tomato leaf images and integrated into a web-based application, effectively classified tomato leaf health. The developed application was a useful tool for all individuals involved in agriculture to manage tomato leaf health because of its capabilities. Its innovative technology not only simplified health classification but also allowed for immediate management techniques, addressing the pressing difficulties of treating tomato leaf health. This application empowered farmers by providing an advanced platform for health classification, allowing them to protect their crops, increase productivity, and promote agricultural sustainability.

Conclusion

The study concludes with the successful development of tomato leaf health classification application. Utilizing a customized Convolutional Neural Network (CNN)-based DenseNet-121 model, the application accurately classifies tomato leaf health and offers tailored management recommendations. Through data preparation, model development, and application integration, the technology demonstrates its effectiveness in empowering farmers, agricultural workers, consumers, and plant health authorities to make informed decisions to protect tomato crops, enhance productivity, and foster agricultural sustainability.

By focusing on plant health management in agriculture, this study introduces a CNN-based DenseNet-121 model as a promising solution. The integration of deep learning techniques into agricultural practices holds significant potential for improving sustainability. The evaluation of the custom DenseNet-121 model showcases exceptional performance in accuracy, specificity, sensitivity, and F1 scores. This comprehensive approach ensures reliability in health classification and represents a substantial step forward in providing farmers and agricultural workers with a valuable tool for early disease detection and effective management strategies, ultimately contributing to agricultural sustainability and economic prosperity.

Recommendations

The researchers recommend the following for future research:

Mobile Application Development. Building a mobile application can significantly benefit farmers and the agricultural sector. Such an application could transform farming practices by offering real-time health classification and management recommendations. This would empower farmers with on-the-spot health diagnosis capabilities, allowing for timely responses to plant health issues directly in the field.

Multi-leaf Disease Classification. Extending the classification framework to diagnose health on groups of leaves would offer a more comprehensive approach. Future research could explore methods for multi-leaf health diagnosis. This expanded scope would provide farmers with a more comprehensive picture of plant health, facilitating more effective health management strategies.

Integration with Agricultural IoT Devices. Integrating the application with agricultural Internet of Things (IoT) devices, such as sensors and drones, can provide additional data inputs for improved disease detection and management. These devices can capture real-time environmental data and images, which can be used to augment the application's capabilities and provide more timely and accurate recommendations to farmers.

Environmental Factors in Disease Classification. Integrating environmental factors into the classification can further improve health diagnosis and management recommendations. Factors like climate, soil conditions, and geographic location play crucial roles in disease development. By incorporating

these, its relevance across various agricultural settings can be enhanced. This ensures that management recommendations are designed for specific environmental conditions.

REFERENCES

Aldhyani, T. H. H., Alkahtani, H., Eunice, R. J., & Hemanth, D. J. (2022). Leaf Pathology Detection in Potato and Pepper Bell Plant using Convolutional Neural Networks. In *2022 7th International Conference on Communication and Electronics Systems (ICCES)*. https://doi.org/10.1109/icces54183.2022.9835735

Alston, J. M., & Pardey, P. G. (2014). Agriculture in the Global Economy. *Journal of Economic Perspectives*, *28*(1), 121–146. https://doi.org/10.1257/jep.28.1.121

Andrew, J., R, J. E., Popescu, D., Chowdary, M. K., & Hemanth, D. J. (2022). Deep Learning-Based Leaf Disease Detection in Crops Using Images for Agricultural Applications. *Agronomy*, *12*(10), 2395. https://doi.org/10.3390/agronomy12102395

Barbedo, J. G. A. (2018). Factors influencing the use of deep learning for plant disease recognition. *Biosystems Engineering*, *172*, 84–91. https://doi.org/10.1016/j.biosystemseng.2018.05.013

blinkAdmin. (2021, June 4). How Cultures Around the World Use Tomatoes in their Cuisines | Authentica World Cuisine. *Authentica World Cuisine*. https://authenticaworldcuisine.com/how-cultures-around-the-world-use-tomatoes-in-their-cuisines/

Dawod, R. G., & Dobre, C. (2022). Upper and Lower Leaf Side Detection with Machine Learning Methods. *Sensors*, *22*(7), 2696. https://doi.org/10.3390/s22072696

De Luna, R. G., Dadios, E. P., & Bandala, A. A. (2018). *Automated image capturing system for deep learning-based tomato plant leaf disease detection and recognition*. https://doi.org/10.1109/tencon.2018.8650088

Dubey, S. R., & Jalal, A. S. (2013). Adapted Approach for Fruit Disease Identification using Images. In *IGI Global eBooks* (pp. 1395–1409). https://doi.org/10.4018/978-1-4666-3994-2.ch069

Fuentes, A., Yoon, S., Kim, S. C., & Park, D. S. (2017). A robust Deep-Learning-Based detector for Real-Time tomato plant diseases and pests recognition. *Sensors*, *17*(9), 2022. https://doi.org/10.3390/s17092022

GmbH, P. (n.d.). *Pests & Diseases | Plantix*. Plantix. https://plantix.net/en/library/plant-diseases/

Grandillo, S., Zamir, D., & Tanksley, S. D. (1999). Genetic improvement of processing tomatoes: A 20 years perspective. *Euphytica*, *110*(2), 85–97. https://doi.org/10.1023/a:1003760015485

Huang, G. (2017). *Densely Connected Convolutional Networks*. https://openaccess.thecvf.com/content_cvpr_2017/html/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.html

Huang, K. (2007). Application of artificial neural network for detecting Phalaenopsis seedling diseases using color and texture features. *Computers and Electronics in Agriculture*, *57*(1), 3–11. https://doi.org/10.1016/j.compag.2007.01.015

Hughes, D. P., & Salathe, M. (2015, November 25). *An open access repository of images on plant health to enable the development of mobile disease diagnostics*. arXiv.org. https://arxiv.org/abs/1511.08060

Hussain, M., Bird, J. T., & Faria, D. R. (2018). A Study on CNN Transfer Learning for Image Classification. In *Advances in intelligent systems and computing* (pp. 191–202). Springer Nature. https://doi.org/10.1007/978-3-319-97982-3_16

J, A. P., & Gopal, G. (2019). Data for: Identification of plant leaf diseases using a 9-layer deep convolutional neural network. *Mendeley Data*, *1*. https://doi.org/10.17632/tywbtsjrjv.1

Kabir, M. M., Ohi, A. Q., & Mridha, M. F. (2020). A Multi-Plant Disease Diagnosis Method using Convolutional Neural Network. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2011.05151

Krishnamoorthy, N., Nirmaladevi, K., Shanth, S., & Karthikeyan, N. (2021). Investigation and comparison of different CNN architectures on tomato leaf disease prediction using deep learning. In *Nucleation and Atmospheric Aerosols*. American Institute of Physics. https://doi.org/10.1063/5.0068638

Kumar, A., & Kumar, A. (2023). Plant Disease Detection using VGG16. *International Journal of Creative Research Thought*, *11*(1). https://ijcrt.org/papers/IJCRT2301347.pdf

Li, G., Ma, Z., & Wang, H. (2012). Image Recognition of Grape Downy Mildew and Grape Powdery Mildew Based on Support Vector Machine. In *IFIP advances in information and communication technology* (pp. 151–162). Springer Science+Business Media. https://doi.org/10.1007/978-3-642-27275-2_17

Minervini, M., Fischbach, A., Scharr, H., & Tsaftaris, S. A. (2016). Finely-grained annotated datasets for image-based plant phenotyping. *Pattern Recognition Letters*, *81*, 80–89. https://doi.org/10.1016/j.patrec.2015.10.013

Mohanty, S. P., Hughes, D. L., & Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, *7*. https://doi.org/10.3389/fpls.2016.01419

Moreno-Miranda, C., & Dries, L. (2022). Assessing the sustainability of agricultural production - a cross-sectoral comparison of the blackberry, tomato and tree tomato sectors in Ecuador. *International Journal of Agricultural Sustainability*, *20*(7), 1373–1396. https://doi.org/10.1080/14735903.2022.2082764

Murugesan, H. (2023, June 30). *The crucial role of technology in our modern world:* https://www.linkedin.com/pulse/crucial-role-technology-our-modern-world-hema-murugasen/

Panigrahi, K. P., Das, H., Sahoo, A. K., & Moharana, S. C. (2020). Maize leaf disease detection and classification using machine learning algorithms. In *Advances in intelligent systems and computing* (pp. 659–669). https://doi.org/10.1007/978-981-15-2414-1_66

PlantVillage. (n.d.). https://plantvillage.psu.edu/

Prajwala, T. M., Pranathi, A., SaiAshritha, K., Chittaragi, N. B., & Koolagudi, S. G. (2018). *Tomato leaf disease detection using convolutional neural networks*. https://doi.org/10.1109/ic3.2018.8530532

Prodeep, A. R., Hoque, A. S. M. M., Kabir, M. M., Rahman, S., & Mridha, M. F. (2022). Plant Disease Identification from Leaf Images using Deep CNN's EfficientNet. In *2022 International Conference on Decision Aid Sciences and Applications (DASA)*. https://doi.org/10.1109/dasa54658.2022.9765063

Quinet, M., Angosto, T., Yuste-Lisbona, F. J., Blanchard-Gros, R., Bigot, S., Martinez, J. A., & Lutts, S. (2019). Tomato fruit development and metabolism. *Frontiers in Plant Science*, *10*. https://doi.org/10.3389/fpls.2019.01554

Rangarajan, A. K., Purushothaman, R., & Ramesh, A. (2018). Tomato crop disease classification using pre-trained deep learning algorithm. *Procedia Computer Science*, *133*, 1040–1047. https://doi.org/10.1016/j.procs.2018.07.070

Rdn, A. B. M. (2023, February 3). *Tomatoes 101: Nutrition Facts and health Benefits*. Healthline. https://www.healthline.com/nutrition/foods/tomatoes#vitamins-and-minerals

Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., & Stefanovic, D. (2016). Deep neural networks based recognition of plant diseases by leaf image classification. *Computational Intelligence and Neuroscience*, *2016*, 1–11. https://doi.org/10.1155/2016/3289801

Sujatha, R., Chatterjee, J. M., Zaman, N., & Brohi, S. N. (2021). Performance of deep learning vs machine learning in plant leaf disease detection. *Microprocessors and Microsystems*, *80*, 103615. https://doi.org/10.1016/j.micpro.2020.103615

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A Survey on Deep Transfer Learning. In *Lecture Notes in Computer Science* (pp. 270–279). Springer Science+Business Media. https://doi.org/10.1007/978-3-030-01424-7_27

Too, E. C., Yujian, L., Njuki, S., & Yingchun, L. (2019a). A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*, *161*, 272–279. https://doi.org/10.1016/j.compag.2018.03.032

Why tomatoes are central to Filipino cuisine. (2023, May 31). Food. https://www.sbs.com.au/food/article/2023/05/31/why-tomatoes-are-central-filipino-cuisine

Wiley, D. (2023). 10 common tomato plant diseases that can wreck your crop. *Better Homes & Gardens*. https://www.bhg.com/gardening/vegetable/vegetables/tomato-plant-diseases/

Yun, S., Xianfeng, W., Shanwen, Z., & Chuanlei, Z. (2015). PNN based crop disease recognition with leaf image features and meteorological data. *International Journal of Agricultural and Biological Engineering*, *8*(4), 60–68. https://doi.org/10.25165/ijabe.v8i4.1719

Zaki, S. K. M., Zulkifley, M. A., Stofa, M. M., Kamari, N. a. M., & Mohamed, N. F. F. (2020). Classification of tomato leaf diseases using MobileNet v2. *IAES International Journal of Artificial Intelligence*, *9*(2), 290. https://doi.org/10.11591/ijai.v9.i2.pp290-296

APPENDICES

APPENDIX A
Approved
Thesis Proposal

NEW ERA UNIVERSITY
College of Computer Studies
Computer Science Department

CCS431-18 THESIS 1
THESIS 1 PROPOSAL FORM

Researchers	Soriano, Jannah R. Torres, Rosh Hashana S.
Proposed Title	Enhancing Agricultural Sustainability: Leaf Disease Diagnosis with Treatment Recommendations using CNN-based DenseNet-121 Model
Field of Study	Deep Learning

Introduction	
Background of the Study	Agriculture plays a vital role in the world's economy as it serves as the primary source of food, income, and employment opportunities. According to the National Economic and Development Authority (NEDA) and the Department of Finance (DOF), the Philippines, being an agricultural country, is in the best position to have an agriculture-driven economy, which can greatly contribute to rebooting the Philippine economy. However, plant diseases and pest infections present a significant threat to agricultural productivity. Early identification and accurate diagnosis of plant diseases is essential for effective disease management and minimizing economic losses (Dawod & Dobre, 2022). Traditional methods of disease identification rely on manual examination by experts, which can be subjective, time-consuming, and prone to errors (Dawod & Dobre, 2022). Recent advancements in computer vision and artificial intelligence have enabled significant progress in automatic plant disease detection and classification. Deep learning techniques, particularly convolutional

	neural networks (CNNs), have shown promising results in image recognition tasks (Sujatha et al., 2021). Transfer learning, which utilizes pre-trained models like DenseNet-121, trained on large-scale datasets like ImageNet, enhances recognition accuracy and efficiency (Tan et al., 2018; Too et al., 2019). The availability of diverse and well-annotated datasets is crucial for training accurate deep learning models. The PlantVillage dataset, developed by Pennsylvania State University, contains a large collection of high-quality images representing 38 classes of plant diseases, making it a valuable resource for plant disease identification studies (*PlantVillage*, n.d.). The main objective in this study is to enhance agricultural sustainability by developing a leaf disease diagnosis application using a CNN-based DenseNet-121 model. The researchers aim to achieve high accuracy, sensitivity, and specificity in diagnosing leaf diseases by fine-tuning the DenseNet-121 model on the PlantVillage dataset. They will create a user-friendly web-based application that allows users to upload leaf images for diagnosis. The trained CNN-based DenseNet-121 model will analyze the uploaded images and provide accurate disease identification results. To further assist them, the researchers will integrate a recommendation system into the application, suggesting appropriate treatment strategies based on the diagnosed disease information. These recommendations may include specific products, cultural practices, or preventive measures to effectively manage and mitigate identified leaf diseases. By combining leaf disease diagnosis with treatment recommendations, this study aims to contribute in enhancing agricultural sustainability.
Main Objective	The main objective of this study is to enhance agricultural sustainability by developing a leaf disease diagnosis application using a CNN-based DenseNet-121 model, and providing treatment recommendations for effective disease management.

Specific Objectives	The researchers have identified the beneficiaries of the proposed study as follows: **a.** To collect and organize a diverse and extensive dataset comprising high quality images of healthy leaves, as well as leaves affected by various diseases, to train and validate CNN-based DenseNet-121 model. **b.** To implement a convolutional neural network model based on DenseNet-121 architecture to accurately classify leaf diseases from input images, achieving high accuracy, sensitivity, and specificity in disease diagnosis. **c.** To create a user-friendly web-based application that utilizes the trained CNN-based DenseNet-121 model to diagnose leaf diseases based on images uploaded by farmers or agricultural workers. **d.** To integrate a recommendation system into the leaf disease diagnosis application, imposing the diagnosed disease information to suggest appropriate treatment strategies, including specific products, cultural practices, or other preventive measures to manage and mitigate the identified leaf diseases effectively.
Significance of the Study	The researchers have identified the beneficiaries of the proposed study as follows: **a. Farmers and Agricultural Workers.** They will benefit from this study as this will help them identify and diagnose diseases affecting their crops more accurately and quickly. With the treatment recommendations provided, they can effectively manage and mitigate the identified leaf diseases, leading to improved crop yield and reduced economic losses. **b. Consumers.** They will benefit from this study as this will help them in ensuring the production of healthier crops. By accurately diagnosing and managing leaf diseases, the quality of the harvested produce can be enhanced, resulting in safer and higher-quality food for consumers in the Philippines. **c. Plant Health Authorities and Agencies.** They will benefit from this study as the research outcomes can be utilized by plant health authorities and agencies, such as the

	Department of Agriculture (DA) and the Bureau of Plant Industry (BPI). The deep learning-based approach developed in this study will enhance their capabilities in detecting and controlling plant leaf diseases effectively, thereby improving their disease surveillance and management programs. **d. Future Researchers.** They will benefit from this study as it contributes to the development of advanced techniques for disease diagnosis and management. The DenseNet-121-based CNN model and the application developed in this study can serve as a valuable reference for further research and innovation in the field of agricultural sustainability and crop protection.
Scope and Limitations	**Scope** This study aims to develop a leaf disease diagnosis application by utilizing CNN-based DenseNet-121 model. To achieve this, a diverse dataset containing high-quality images of healthy leaves and leaves affected by various diseases will be collected and organized. The focus will be on implementing a convolutional neural network model based on the DenseNet-121 architecture, ensuring accurate classification of leaf diseases. Additionally, a user-friendly web-based application will be created to allow farmers and agricultural workers to upload leaf images for accurate disease diagnosis. Furthermore, the application will integrate a recommendation system to suggest suitable treatment strategies for managing and mitigating identified leaf diseases. The study will cover eight (8) crop species, including apple, corn, grape, mango, pepper, potato, strawberry, and tomato. The dataset used in the study will consist of 8 healthy leaf classes and 27 diseased leaf classes, totaling 35 classes. **Limitations** This study has certain limitations that should be acknowledged. Primarily, it focuses on leaf disease diagnosis and treatment recommendations using a CNN-based DenseNet-121 model, overlooking other important aspects of

	agricultural sustainability. The findings may not directly apply to crop species beyond the seven species examined. The accuracy of the diagnosis may be affected by factors like image quality and user errors since the application relies on user-uploaded leaf images. It is worth noting that the study does not involve developing physical treatment products or cultural practices; it simply provides recommendations based on diagnosed diseases. The economic feasibility and cost-effectiveness of the recommended treatment strategies are not addressed. Additionally, it is important to mention that this study only focuses on detecting plant diseases on single leaves and does not extend to detecting diseases on groups of leaves.

Review of Related Literature

This chapter presents a review of the relevant literature essential for establishing the foundational knowledge required to build and develop the proposed study. The researchers gathered and utilized a range of literature pertaining to Leaf Disease Classification and DenseNet-121.

Leaf Disease Classification

Leaf diseases present a significant threat to agricultural sustainability, as they can cause substantial losses in crop production and reduce the quality and quantity of food (Alston & Pardey, 2014). Identifying and diagnosing leaf diseases accurately and early on is crucial for effective management and treatment. Traditional methods of disease identification rely on manual examination by experts, which can be subjective, time-consuming, and prone to errors (Dawod & Dobre, 2022). In recent years, researchers have turned to computer vision and deep learning techniques to automate the detection and classification of leaf diseases.

Computer vision techniques leverage image processing and machine learning algorithms to analyze visual information from leaf images and identify disease symptoms. Various approaches have been proposed, including hand-crafted feature extraction and segmentation combined with machine learning algorithms (Scientist et al., 2020; Dubey & Jalal, 2013; Li et al., 2012). These methods have shown promising results but often require extensive manual feature engineering and lack scalability.

Deep learning, specifically convolutional neural networks (CNNs), has revolutionized the field of image classification and object detection (Sujatha et al., 2021). CNNs can automatically learn and extract complex features from

images, making them well-suited for leaf disease classification. One popular CNN model used in leaf disease diagnosis is DenseNet-121 (Barbedo, 2018). DenseNet-121 is a deep CNN model that utilizes dense connections between layers, allowing information to flow more efficiently and improving the model's ability to capture fine-grained details (Huang et al., 2017). This architecture has been shown to achieve high accuracy in plant disease recognition tasks (Mohanty et al., 2016).

The classification and diagnosis of leaf diseases in agriculture have been significantly improved with the adoption of computer vision and deep learning techniques. Traditional methods have limitations in terms of subjectivity and time-consuming manual processes (Dawod & Dobre, 2022). Deep learning models like DenseNet-121 offer promising solutions by automating the feature extraction and classification tasks (Barbedo, 2018; Huang et al., 2017). These advancements in technology enable early classification and accurate diagnosis, leading to more effective management and treatment of leaf diseases, thus enhancing agricultural sustainability.

DenseNet-121

DenseNet-121, also known as Densely Connected Convolutional Network, is a deep convolutional neural network architecture that has gained popularity in various image classification tasks, including leaf disease classification in agriculture. The key innovation of DenseNet-121 lies in its dense connections, which allow for direct connections between all layers within the network (Huang et al., 2017). Unlike traditional CNNs, where information flows sequentially from one layer to the next, DenseNet-121 enables direct access to the feature maps of all preceding layers. This dense connectivity enhances gradient flow, encourages feature reuse, and enables the model to capture more intricate patterns and details.

The architecture of DenseNet-121 consists of several dense blocks, each composed of multiple convolutional layers with batch normalization and activation functions (Huang et al., 2017). Within each dense block, the feature maps of all preceding layers are concatenated, creating a dense connectivity pattern. This design enables feature reuse and facilitates the propagation of gradients, addressing the vanishing gradient problem often encountered in deep neural networks.

DenseNet-121 has shown impressive performance in various image classification tasks, including leaf disease classification in agriculture (Barbedo, 2018; Mohanty et al., 2016). Its dense connectivity allows the model to efficiently extract and propagate features, leading to improved accuracy and robustness. Moreover, DenseNet-121 can be trained with transfer learning, leveraging pre-trained models on large-scale datasets such as ImageNet (Hussain et al., 2018). This transfer learning approach enables the model to leverage knowledge

learned from a vast amount of data and generalize well to new tasks with limited training data.
DenseNet-121 is a powerful deep learning architecture for image classification tasks, including leaf disease classification in agriculture. Its dense connectivity promotes feature reuse and gradient flow, enabling the model to capture intricate patterns and achieve high accuracy. Leveraging transfer learning further enhances its performance, especially when training data is limited. DenseNet-121 has emerged as a valuable tool in enhancing agricultural sustainability by improving the diagnosis and management of leaf diseases.

Conceptual Framework & Project Design

This chapter provides a conceptual framework essential for understanding the relationships and variables of the proposes study, and a project design essential for outlining the specific methods and procedures to be employed in conducting the study.

Conceptual Framework

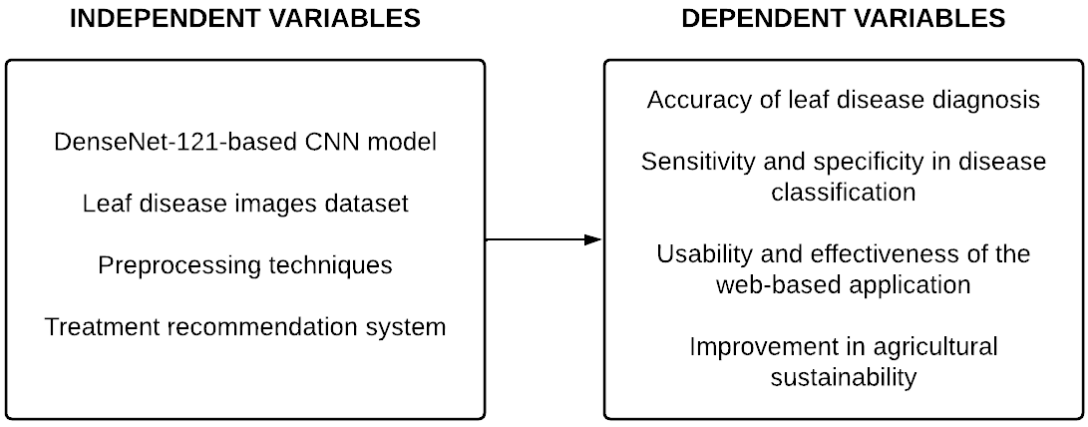


Figure 1: Conceptual Framework

The collection and organization of a diverse dataset comprising healthy and diseased leaf images serve as the foundation. This dataset is used to train and optimize a CNN-based DenseNet-121 model, a deep learning architecture known for its effectiveness in image classification. The trained model is integrated into a user-friendly web-based application that allows users to upload leaf images for disease diagnosis and provides treatment recommendations for effective management and mitigation of leaf diseases. This study aims to enhance agricultural sustainability by enabling accurate disease diagnosis and guiding those people who work in agriculture in implementing appropriate treatment strategies.

Project Design

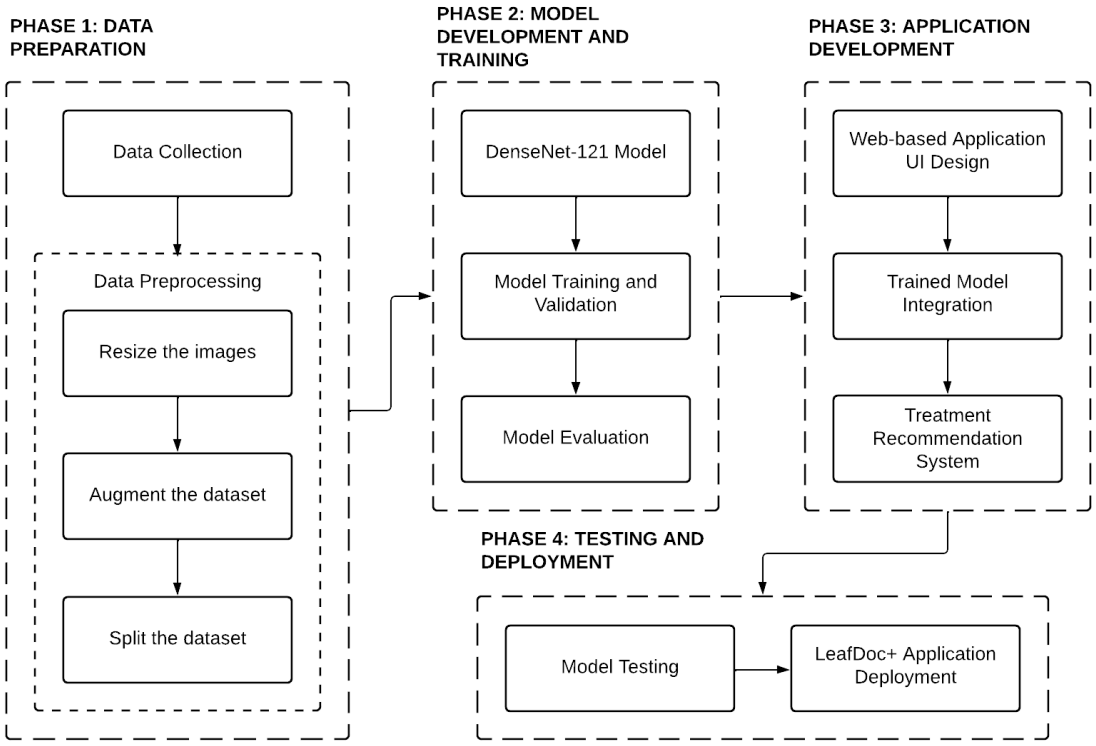


Figure 2: Project Design

Phase 1: Data Preparation – In the first phase, it focuses on preparing the necessary data for the leaf disease diagnosis application. Data collection involves gathering relevant images and information of leaf diseases from different sources, including Kaggle and Mendeley. Once the data is collected, it undergoes preprocessing to ensure its suitability for analysis. This preprocessing includes tasks such as resizing the leaf images to a consistent size, augmenting the dataset by creating variations of the original images, and splitting the dataset into subsets for training, validation, and testing purposes. By properly preparing the data, it becomes ready for the succeeding phases of the project.

Phase 2: Model Development and Training – In the second phase, it focuses on developing and training the leaf disease diagnosis application model. The chosen model architecture for this project is DenseNet-121, a deep learning model known for its effectiveness in image classification tasks. This model is implemented and trained using the prepared dataset. During training, the model's internal parameters are adjusted repeatedly, allowing it to learn and improve its ability to identify leaf diseases accurately. Validation is performed alongside training to assess the model's performance and make any necessary adjustments. Once training and validation are complete, the model moves into

the evaluation stage, where its performance is measured using appropriate evaluation metrics by comparing its predictions with the actual correct labels for the leaf diseases, known as the ground truth labels. This comparison allows us to assess how well the model is performing in accurately identifying the diseases.

Phase 3: Application Development – In the third phase, it focuses on developing the leaf disease diagnosis application. This involves creating a user interface (UI) design for a web-based application that provides an intuitive and user-friendly platform for users to interact with the system. The trained DenseNet-121 model is then integrated into the application, allowing it to receive input images and provide disease predictions using the integrated model. Additionally, a treatment recommendation system is developed within the application, utilizing the predicted leaf diseases to offer strategies, including specific products, cultural practices, or other preventive measures, for managing and mitigating the predicted disease effectively.

Phase 4: Testing and Deployment – In the fourth and final phase, it focuses on testing and deploying the developed application. Model testing is conducted to ensure the usability and accuracy of the integrated model within the web-based application. Various input images are used to assess the model's disease prediction and treatment recommendation capabilities, ensuring its trustworthiness and effectiveness. Once the testing phase is successfully completed, the leaf disease diagnosis application is deployed, making it available for users to access and utilize on the web. By reaching this stage, the project is ready to provide farmers and agricultural workers with a useful tool for diagnosing leaf diseases and receiving treatment recommendations, ultimately contributing to the enhancement of agricultural sustainability.

References

Alston, J. M., & Pardey, P. G. (2014). Agriculture in the Global Economy. *Journal of Economic Perspectives*, *28*(1), 121–146. https://doi.org/10.1257/jep.28.1.121

Andrew, J., R, J. E., Popescu, D., Chowdary, M. K., & Hemanth, D. J. (2022). Deep Learning-Based Leaf Disease Detection in Crops Using Images for Agricultural Applications. Agronomy, 12(10), 2395. https://doi.org/10.3390/agronomy12102395

Barbedo, J. G. A. (2018). Factors influencing the use of deep learning for plant disease recognition. *Biosystems Engineering*, *172*, 84–91. https://doi.org/10.1016/j.biosystemseng.2018.05.013

Dawod, R. G., & Dobre, C. (2022). Upper and Lower Leaf Side Detection with Machine Learning Methods. *Sensors*, *22*(7), 2696. https://doi.org/10.3390/s22072696

Dubey, S. R., & Jalal, A. S. (2013). Adapted Approach for Fruit Disease Identification using Images. In *IGI Global eBooks* (pp. 1395–1409). https://doi.org/10.4018/978-1-4666-3994-2.ch069

Huang, G., Liu, Z., Maaten L., & Weinberger, K.Q. (2017). *Densely Connected Convolutional Networks*. https://openaccess.thecvf.com/content_cvpr_2017/html/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.html

Hussain, M., Bird, J. T., & Faria, D. R. (2018). A Study on CNN Transfer Learning for Image Classification. In *Advances in intelligent systems and computing* (pp. 191–202). Springer Nature. https://doi.org/10.1007/978-3-319-97982-3_16

Li, G., Ma, Z., & Wang, H. (2012). Image Recognition of Grape Downy Mildew and Grape Powdery Mildew Based on Support Vector Machine. In *IFIP advances in information and communication technology* (pp. 151–162). Springer Science+Business Media. https://doi.org/10.1007/978-3-642-27275-2_17

Mohanty, S. P., Hughes, D. L., & Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, *7*. https://doi.org/10.3389/fpls.2016.01419

PlantVillage. (n.d.). https://plantvillage.psu.edu/

Sujatha, R., Chatterjee, J. M., Zaman, N., & Brohi, S. N. (2021). Performance of deep learning vs machine learning in plant leaf disease detection. *Microprocessors and Microsystems*, *80*, 103615. https://doi.org/10.1016/j.micpro.2020.103615

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A Survey on Deep Transfer Learning. In *Lecture Notes in Computer Science* (pp. 270–279). Springer Science+Business Media. https://doi.org/10.1007/978-3-030-01424-7_27

Too, E. C., Yujian, L., Njuki, S., & Yingchun, L. (2019). A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*, *161*, 272–279. https://doi.org/10.1016/j.compag.2018.03.032

Approving Committee

Prof. AUDREY LYLE D. DIEGO
Dean

Dr. MARC P. LAURETA
Thesis Adviser

Engr. JEREMIAS C. ESPERANZA
Program Head

APPENDIX B
Grammarly

Tomato Leaf Health Classification with Management Recommendations using CNN-based DenseNet-121 for Agricultural Sustainabi

by Ryven

General metrics

73,033	10,257	618	41 min 1 sec	1 hr 18 min
characters	words	sentences	reading time	speaking time

Score

This text scores better than 98% of all texts checked by Grammarly

Writing Issues

138	50	88
Issues left	Critical	Advanced

Plagiarism

44 sources

4% of your text matches 44 sources on the web or in archives of academic publications

Writing Issues

62	**Correctness**	
7	Mixed dialects of english	
1	Citation style options	
6	Ungrammatical sentence	
1	Commonly confused words	
2	Misplaced words or phrases	
4	Unknown words	
4	Punctuation in compound/complex sentences	
2	Pronoun use	
9	Misspelled words	
3	Incorrect noun number	
7	Incorrect verb forms	
5	Incorrect phrasing	
3	Confused words	
2	Wrong or missing prepositions	
1	Conjunction use	
3	Improper formatting	
2	Determiner use (a/an/the/this, etc.)	
42	**Engagement**	
42	Word choice	
14	**Delivery**	
4	Inappropriate colloquialisms	
8	Incomplete sentences	
2	Tone suggestions	

20	**Clarity**	
14	Intricate text	
4	Passive voice misuse	
2	Hard-to-read text	

Unique Words

17%

Measures vocabulary diversity by calculating the percentage of words used only once in your document

unique words

Rare Words

50%

Measures depth of vocabulary by identifying words that are not among the 5,000 most common English words.

rare words

Word Length

5.8

Measures average word length

characters per word

Sentence Length

16.6

Measures average sentence length

words per sentence

APPENDIX C
Turnitin

Tomato Leaf Health Classification with Management Recommendations using CNN-based DenseNet-121 for Agricultural Sustainability.docx

ORIGINALITY REPORT

8%	5%	5%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.mdpi.com Internet Source	3%
2	R. Karthik, J. Joshua Alfred, J. Joel Kennedy. "Inception-based global context attention network for the classification of coffee leaf diseases", Ecological Informatics, 2023 Publication	<1%
3	Submitted to University of Sunderland Student Paper	<1%
4	"Pan-African Conference on Artificial Intelligence", Springer Science and Business Media LLC, 2024 Publication	<1%
5	www.ijraset.com Internet Source	<1%
6	www.irjmets.com Internet Source	<1%
7	Submitted to University of North Texas	

	Student Paper	<1%
8	jivp-eurasipjournals.springeropen.com Internet Source	<1%
9	www2.mdpi.com Internet Source	<1%
10	www.jsr.org Internet Source	<1%
11	www.biorxiv.org Internet Source	<1%
12	Submitted to University of Sheffield Student Paper	<1%
13	boris.unibe.ch Internet Source	<1%
14	Submitted to srmap Student Paper	<1%
15	www.theivacompany.co.uk Internet Source	<1%
16	Submitted to Asian Institute of Technology Student Paper	<1%
17	Leixian Qiao, Xue Li, Shuqiang Jiang. "RGB-D Object Recognition from Hand-Held Object Teaching", Proceedings of the International	<1%

	Conference on Internet Multimedia Computing and Service - ICIMCS'16, 2016 Publication	
18	Submitted to University of Hertfordshire Student Paper	<1 %
19	Submitted to University of Nottingham Student Paper	<1 %
20	Submitted to Arts, Sciences & Technology University In Lebanon Student Paper	<1 %
21	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	<1 %
22	Submitted to Berlin School of Business and Innovation Student Paper	<1 %
23	Eric Hitimana, Omar Janvier Sinayobye, J. Chrisostome Ufitinema, Jane Mukamugema et al. "An Intelligent System-Based Coffee Plant Leaf Disease Recognition Using Deep Learning Techniques on Rwandan Arabica Dataset", Technologies, 2023 Publication	<1 %
24	Rajasekaran Thangaraj, P. Pandiyan, S. Anandamurugan, Sivaramakrishnan Rajendar. "A deep convolution neural network model based on feature concatenation approach for	<1 %

classification of tomato leaf disease", Multimedia Tools and Applications, 2023
Publication

25	Sue Han Lee, Hervé Goëau, Pierre Bonnet, Alexis Joly. "New perspectives on plant disease characterization based on deep learning", Computers and Electronics in Agriculture, 2020 Publication	<1 %
26	Zhiyun Xue, Sivaramakrishnan Rajaraman, Rodney Long, Sameer Antani, George Thoma. "Gender Detection from Spine X-Ray Images Using Deep Learning", 2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS), 2018 Publication	<1 %
27	www.frontiersin.org Internet Source	<1 %
28	Bryan He, Ludvig Bergenstråhle, Linnea Stenbeck, Abubakar Abid et al. "Integrating spatial gene expression and breast tumour morphology via deep learning", Nature Biomedical Engineering, 2020 Publication	<1 %
29	Ekta Kumari, Mahendra K. Shukla, Om Jee Pandey, Suneel Yadav. "NeuroAid: Emotion-Based EEG Analysis for Parkinson's Disease Identification", IEEE Sensors Letters, 2023	<1 %

	Publication	
30	www.nature.com Internet Source	<1 %
31	Andrew J., Jennifer Eunice R., Daniela Elena Popescu, M. Kalpana Chowdary, Jude Hemanth D.. "Deep Learning-Based Leaf Disease Detection in Crops Using Images for Agricultural Applications", Agronomy, 2022 Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

APPENDIX D
Proofreading Certificate

DON MARIANO MARCOS MEMORIAL STATE UNIVERSITY
South La Union Campus
COLLEGE OF EDUCATION
Agoo, La Union, Philippines
Telephone No. (072)- 682-0480
Email Address: ce.sluc@dmmmsu.edu.ph

OFFICE OF THE PROGRAM CHAIR

CERTIFICATE OF PROOFREADING

This is to certify that I have edited this thesis manuscript entitled **TOMATO LEAF HEALTH CLASSIFICATION WITH MANAGEMENT RECOMMENDATIONS USING CNN-BASED DENSENET-121 FOR AGRICULTURAL SUSTAINABILITY** prepared by JANNAH R. SORIANO and ROSH HASHANA S. TORRES and have found it through and acceptable with respect to grammar and composition.

Issued this 18th of May 2024.

ALEXANDER S. LOPEZ
Program Chair, BECEd
DMMMSU-SLUC

APPENDIX E
Source Code

Loading the Dataset

```
# Get the current working directory
current_dir = os.getcwd()

# Define dataset directory
dataset_dir = os.path.join(current_dir, "dataset")

img_rows, img_cols = 224, 224
arr = os.listdir(dataset_dir)
print(arr)

count = []
for i in os.listdir(dataset_dir):
    l = len(os.listdir(os.path.join(dataset_dir, i)))
    count.append(l)

class_count = {key:value for key,value in enumerate(count)}

class_count.keys()
```

Data Exploration

```
Plotting the Image Count

plt.bar(class_count.keys(), class_count.values(), width = .5);
plt.title("Number of Images by Class");
plt.xlabel('Class Name');
plt.ylabel('Images_Count');

# Save the image
images_dir = os.path.join(current_dir, "images")
os.makedirs(images_dir, exist_ok=True)
image_path = os.path.join(images_dir, "image_count.png")
plt.savefig(image_path)

Plotting the Image Shape

def image_shape(path):
    index = [i for i in range(len(os.listdir(os.path.join(dataset_dir,
path))))]
    df = pd.DataFrame(index=index, columns=['height', 'width'])
    for i, j in enumerate(os.listdir(os.path.join(dataset_dir, path))):
        image = cv2.imread(os.path.join(dataset_dir, path, j))
        h, w, d = image.shape
        df['height'][i] = h
        df['width'][i] = w
    return df

s = 0
plt.figure(figsize=(15,65))
for path in arr:
    s += 1
    size = image_shape(path)
```

```
    plt.subplot(13, 3, s)

    plt.scatter(size['height'],size['width'])

    plt.xlabel('Width')
    plt.ylabel('Height')

    plt.title('{}'.format(path))

# Save the image
images_dir = os.path.join(current_dir, "images")
os.makedirs(images_dir, exist_ok=True)
image_path = os.path.join(images_dir, "image_shape.png")
plt.savefig(image_path)

Visualizing the Images

def load_images(class_path, num_samples=3):
    return [cv2.resize(cv2.cvtColor(cv2.imread(os.path.join(class_path, img_name)), cv2.COLOR_BGR2RGB), (224, 224)) for img_name in os.listdir(class_path)[:num_samples]]

num_samples = 3
num_classes = len(os.listdir(dataset_dir))
plt.figure(figsize=(15, 6*num_classes))

for i, class_name in enumerate(sorted(os.listdir(dataset_dir))):
    class_path = os.path.join(dataset_dir, class_name)
    images = load_images(class_path, num_samples)

    plt.subplot(num_classes, num_samples+1, i*(num_samples+1)+1)
    plt.imshow(np.ones((224, 224, 3), dtype=np.uint8) * 255)
    plt.axis('off')
    plt.title(class_name, fontsize=10)

    for j, img in enumerate(images):
        plt.subplot(num_classes, num_samples+1, i*(num_samples+1)+j+2)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'Shape: {img.shape}', fontsize=10)

plt.tight_layout()
plt.show()
```

Data Cleaning

```
Check for corrupted or unreadable images

# Function to check if an image file is readable
def is_image_readable(file_path):
    try:
        img = cv2.imread(file_path)
        if img is None:
            return False
```

```
        else:
            return True
    except Exception as e:
        print(f"Error reading {file_path}: {str(e)}")
        return False

# Iterate over all folders in the dataset directory
for folder in os.listdir(dataset_dir):
    folder_path = os.path.join(dataset_dir, folder)
    num_corrupted_images = 0

    # Iterate over all files in the folder
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file)
        if not is_image_readable(file_path):
            num_corrupted_images += 1
            os.remove(file_path)

    print(f"{folder}: {num_corrupted_images} corrupted/unreadable images")
```

Standardize image formats

```
# Function to get the image format
def get_image_format(file_path):
    try:
        img = Image.open(file_path)
        return img.format
    except Exception as e:
        print(f"Error getting image format for {file_path}: {str(e)}")
        return None

# Count occurrences of each image format
image_formats = {}

# Iterate over all folders in the dataset directory
for folder in os.listdir(dataset_dir):
    folder_path = os.path.join(dataset_dir, folder)

    # Iterate over all files in the folder
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file)
        image_format = get_image_format(file_path)
        if image_format:
            image_formats.setdefault(image_format, 0)
            image_formats[image_format] += 1

# Print the results
print("Image Format Counts:")
for format_name, count in image_formats.items():
    print(f"{format_name.upper()}: {count}")
```

Data Split

```
# Define paths
```

```
train_dir = os.path.join(current_dir, "dataset_split", "train") # train_ratio = 0.8
val_dir = os.path.join(current_dir, "dataset_split", "val") # val_ratio = 0.1
test_dir = os.path.join(current_dir, "dataset_split", "test")  # test_ratio = 0.1
```

Constants

```
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 30
LEARNING_RATE = 0.001
NUM_CLASSES = 6
```

Preprocessing and Data Augmentation

```
# ImageDataGenerator for preprocessing and data augmentation
train_datagen = ImageDataGenerator(
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    val_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

# Get class indices
class_indices = train_generator.class_indices

# Count the number of images per class in the training set
train_image_count_per_class = {class_name: len(os.listdir(os.path.join(train_dir, class_name))) for class_name in class_indices}

# Count the number of images per class in the validation set
val_image_count_per_class = {class_name: len(os.listdir(os.path.join(val_dir, class_name))) for class_name in class_indices}

print("Number of images per class in the training set:")
print(train_image_count_per_class)
print("\nNumber of images per class in the validation set:")
print(val_image_count_per_class)
```

Customizing and Fine-tuning the DenseNet-121 Model

```
# Load pre-trained DenseNet121 model without the top classification layer
```

```
base_model = DenseNet121(include_top=False, weights='imagenet',
input_shape=(IMG_SIZE, IMG_SIZE, 3))

# Add custom classification layers

# Create final model
optimizer = Adam(learning_rate=LEARNING_RATE)
model = Model(inputs=base_model.input, outputs=predictions)

# Fine-tuning: Unfreeze some layers of the base model

# Precision, Specificity, Sensitivity (Recall), and F1 Score

# Compile the model
optimizer = Adam(learning_rate=LEARNING_RATE)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy', precision, specificity, recall, f1])

model.summary()
```

Define Training Callbacks

```
# Define paths
checkpoint_path = "model.hdf5"
log_path = "training_log.csv"

# ModelCheckpoint
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
verbose=1, save_best_only=True, mode='auto')

# CSVLogger
csv_logger = CSVLogger(log_path)

# LearningRateScheduler
def lr_schedule(epoch):
    if epoch < 10:
        return LEARNING_RATE
    elif epoch < 20:
        return LEARNING_RATE * 0.1
    else:
        return LEARNING_RATE * 0.01

lr_scheduler = LearningRateScheduler(lr_schedule)

# Early Stopping
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, verbose=1,
restore_best_weights=True)
```

Model Training

```
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.n // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
```

```
    validation_steps=validation_generator.n // BATCH_SIZE,
    callbacks=[checkpoint_callback, csv_logger, lr_scheduler, early_stopping]
)
```

Model Evaluation

```
Plotting Training and Validation Accuracy and Loss per Epoch

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)

# Plotting
plt.figure(figsize=(20, 5))

# Plot training and validation accuracy per epoch
plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, 'blue', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'orange', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.xticks(range(1, EPOCHS + 1))

# Plot training and validation loss per epoch
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'blue', label='Training Loss')
plt.plot(epochs, val_loss, 'orange', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.xticks(range(1, EPOCHS + 1))

# Save the image
images_dir = os.path.join(current_dir, "images")
os.makedirs(images_dir, exist_ok=True)
image_path = os.path.join(images_dir, "training_validation_accuracy-
loss_plot.png")
plt.savefig(image_path)

plt.show()

Model Performance Summary: Accuracy, Precision, Specificity, Sensitivity
(Recall), and F1 Score

# Extract metrics from history
final_accuracy = history.history['accuracy'][-1]
final_precision = history.history['precision'][-1]
```

```
final_specificity = history.history['specificity'][-1]
final_recall = history.history['recall'][-1]
final_f1_score = history.history['f1'][-1]

# Convert scores to percentages
final_accuracy_percentage = final_accuracy * 100
final_precision_percentage = final_precision * 100
final_specificity_percentage = final_specificity * 100
final_recall_percentage = final_recall * 100
final_f1_score_percentage = final_f1_score * 100

# Print the model performance summary
print("Model Performance Summary:")
print("Accuracy: {:.4f} or {:.2f}%".format(final_accuracy,
final_accuracy_percentage))
print("Precision: {:.4f} or {:.2f}%".format(final_precision,
final_precision_percentage))
print("Specificity: {:.4f} or {:.2f}%".format(final_specificity,
final_specificity_percentage))
print("Sensitivity (Recall): {:.4f} or {:.2f}%".format(final_recall,
final_recall_percentage))
print("F1 Score: {:.4f} or {:.2f}%".format(final_f1_score,
final_f1_score_percentage))
```

Model Testing

```
# Load the best weights
model.load_weights(checkpoint_path)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

# Make predictions on test data
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Get true labels
true_classes = test_generator.classes

# Get class labels
class_labels = list(test_generator.class_indices.keys())

# Generate confusion matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=class_labels, yticklabels=class_labels)
```

```
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion Matrix')

# Save the image
images_dir = os.path.join(current_dir, "images")
os.makedirs(images_dir, exist_ok=True)
image_path = os.path.join(images_dir, "confusion_matrix.png")
plt.savefig(image_path)

plt.show()

# Generate classification report
report = classification_report(true_classes, predicted_classes,
target_names=class_labels)

# Print the classification report
print(report)

# Calculate specificity for each class
def specificity_per_class(conf_matrix):
    spec_per_class = {}
    for i, class_label in enumerate(class_labels):
        true_negatives = np.sum(np.delete(np.delete(conf_matrix, i, axis=0),
i, axis=1))
        false_positives = np.sum(conf_matrix[:, i]) - conf_matrix[i, i]
        possible_negatives = np.sum(np.delete(conf_matrix, i, axis=0)) -
np.sum(conf_matrix[i, :])
        specificity = true_negatives / (true_negatives + false_positives +
np.finfo(float).eps)
        spec_per_class[class_label] = specificity
    return spec_per_class

# Calculate specificity for each class
spec_per_class = specificity_per_class(conf_matrix)

# Print specificity for each class
print("{: <40}{}".format("Class", "Specificity"))
for class_name, spec in spec_per_class.items():
    print("{: <40}{:.2f}".format(class_name, spec))

# Print macro and weighted average specificity
macro_avg_spec = np.mean(list(spec_per_class.values()))
weighted_avg_spec = np.average(list(spec_per_class.values()),
weights=np.bincount(true_classes))
print("\n{: <40}{:.2f}".format("Macro Avg", macro_avg_spec))
print("{: <40}{:.2f}".format("Weighted Avg", weighted_avg_spec))
```

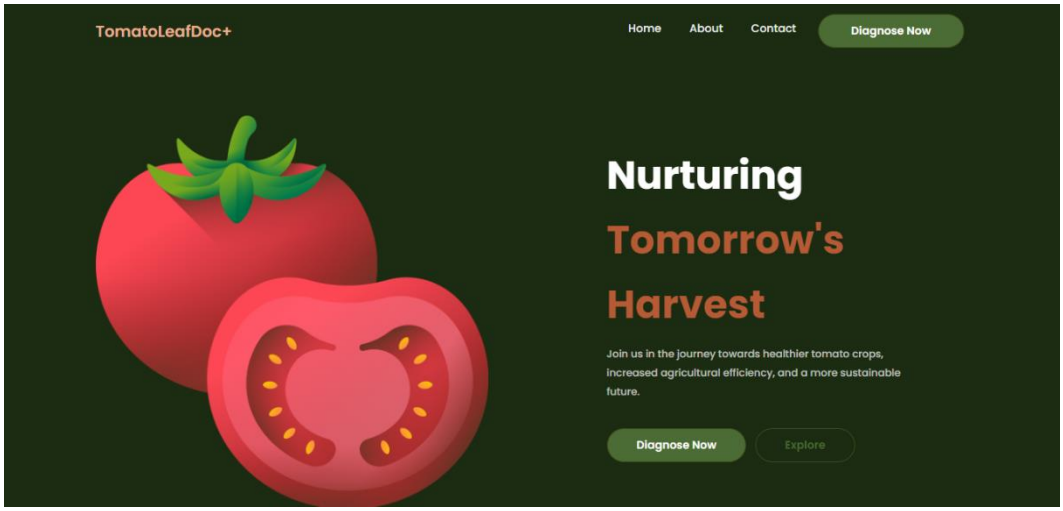
APPENDIX F
ACM

APPENDIX G
User's Manual

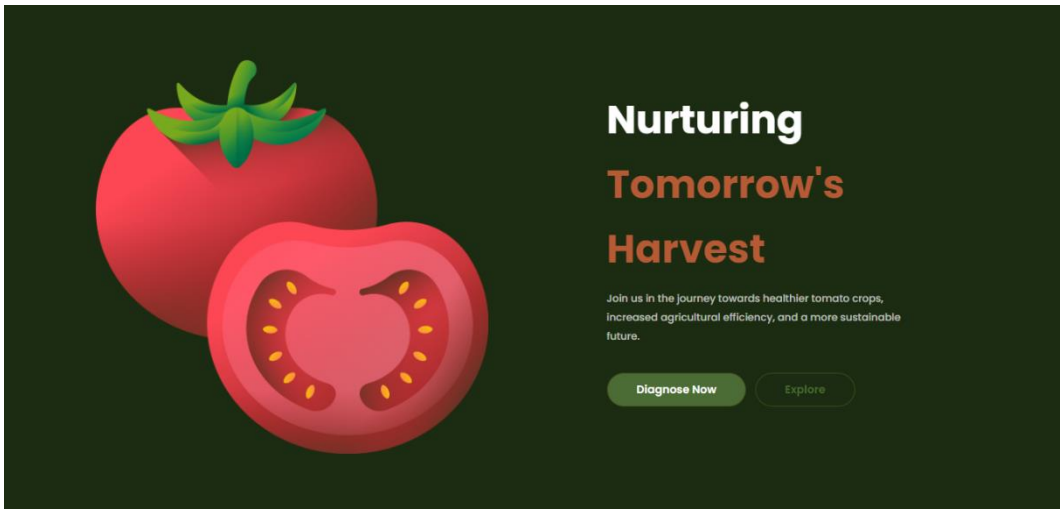
This guide will help you navigate and utilize all features of the application to enhance your tomato crop health and agricultural efficiency.

LANDING PAGE

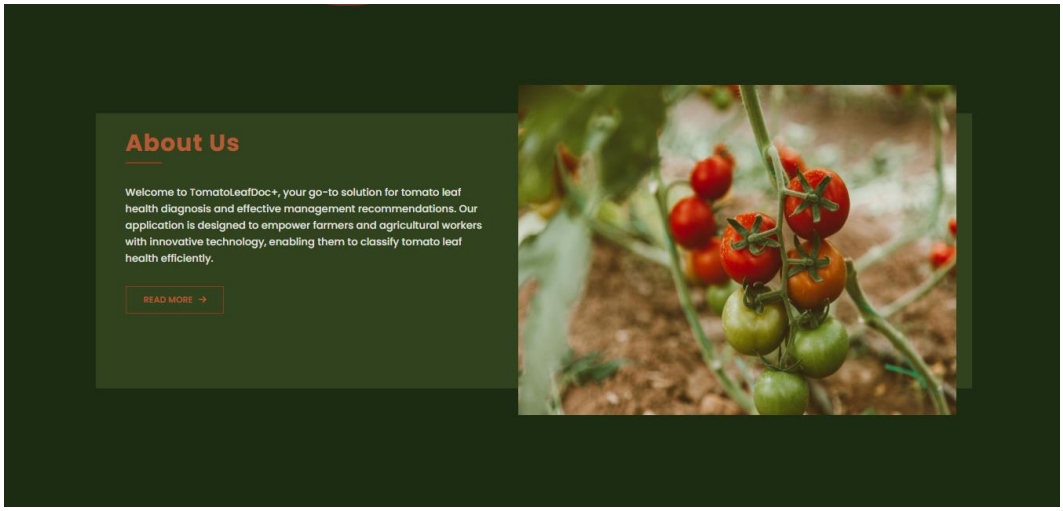
The landing page is your gateway to discovering the application. It provides an overview of the application's mission, features, and benefits.



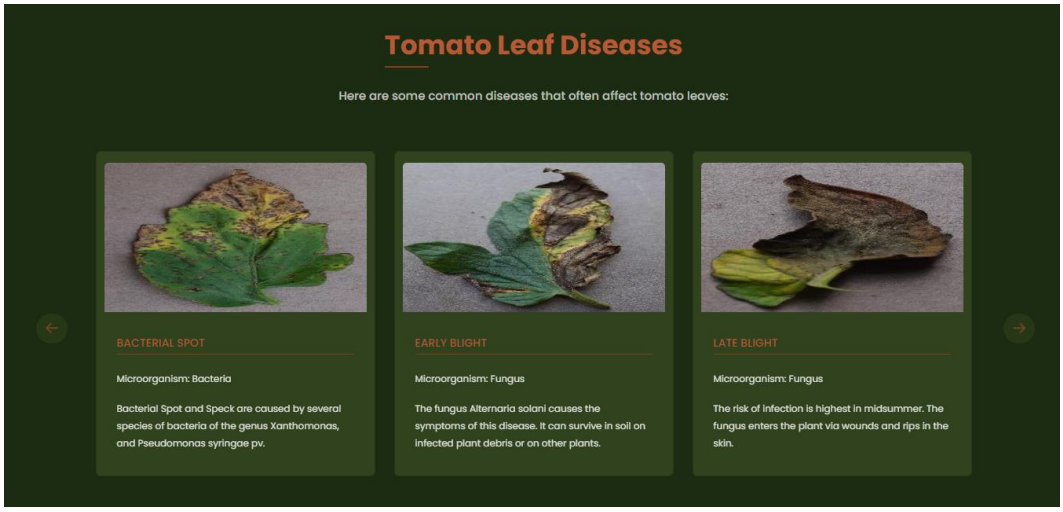
Home – This section welcomes you to the application. Join us in the journey towards healthier tomato crops, improved agricultural efficiency, and a more sustainable future.



About Us – Learn about the application, its purpose, and how it serves as a solution for diagnosing tomato leaf health and offering management recommendations.



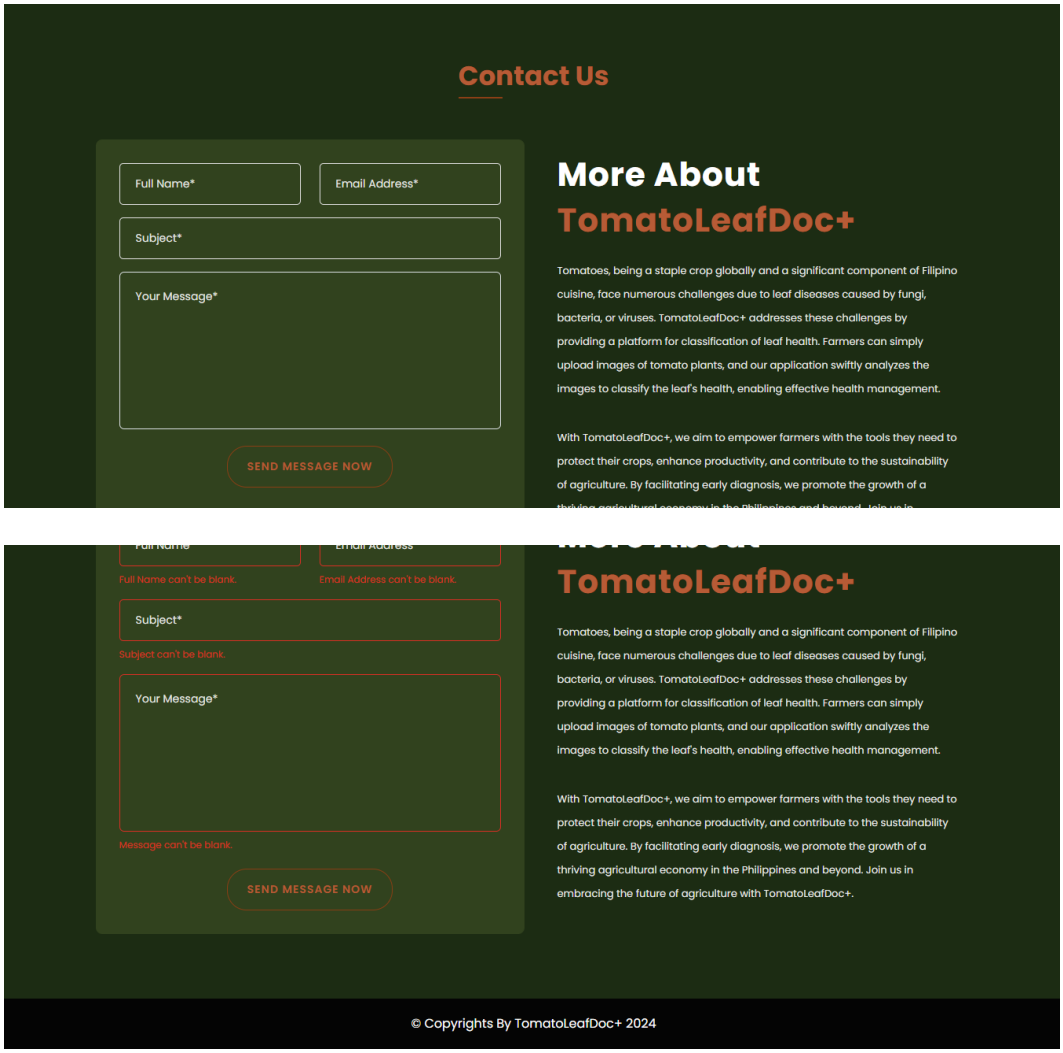
Tomato Leaf Diseases – This section provides detailed information about the common diseases that often affect tomato leaves.



How it Works – Follow these steps to use the application effectively:

1. Upload an image of a diseased tomato leaf.
2. Receive an instant analysis.
3. Get specific recommendations for disease management.

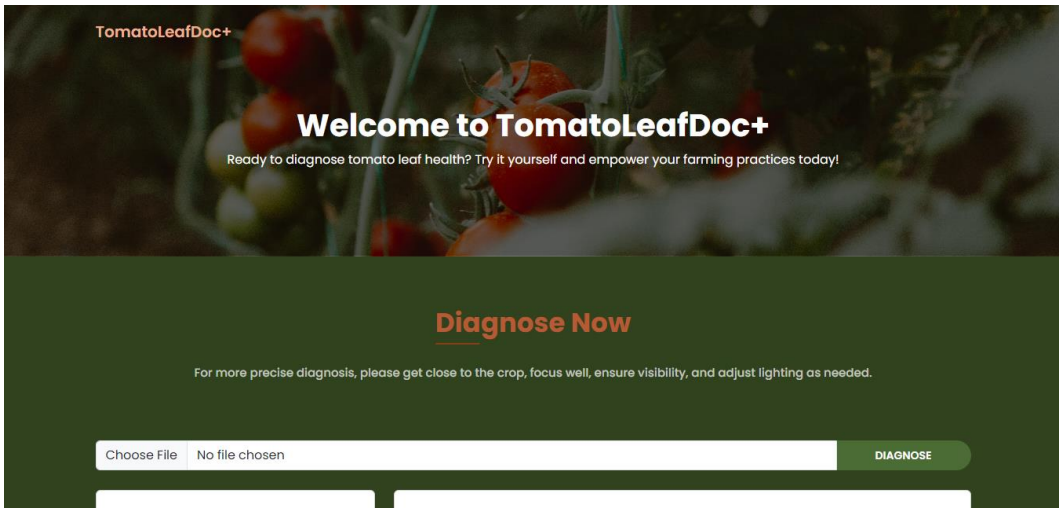
Contact Us – Need assistance or have questions? Use the contact form in this section to get in touch for support, inquiries, or feedback.

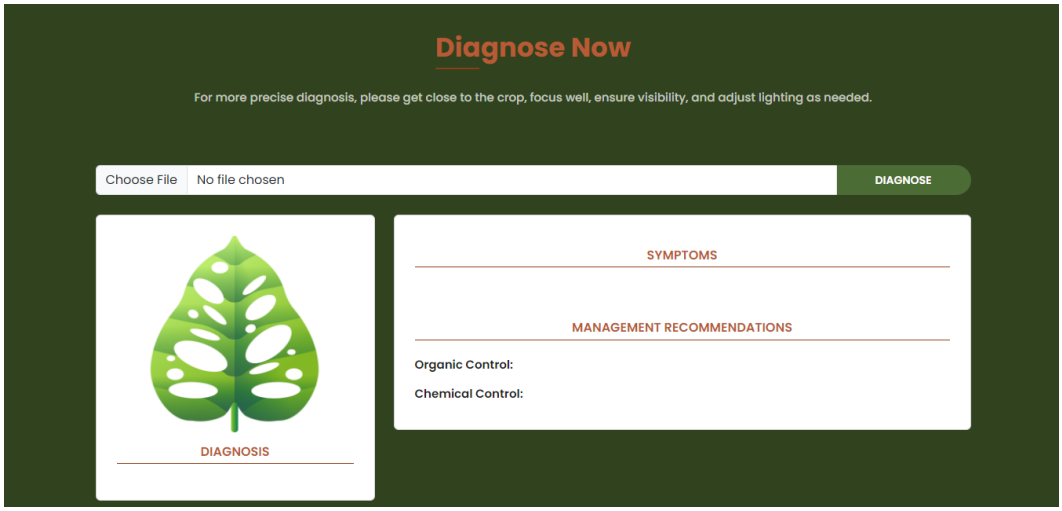


CLASSIFICATION PAGE

The Classification Page is where you can upload images of your tomato leaves to receive a health diagnosis and management recommendations.

1. **Upload Image:** Select and upload an image of a tomato leaf.
2. **Receive Diagnosis:** The application will analyze the image and determine if the leaf is healthy or identify the specific disease.
3. **Get Recommendations:** Based on the diagnosis, you will receive management recommendations to address the identified issue.

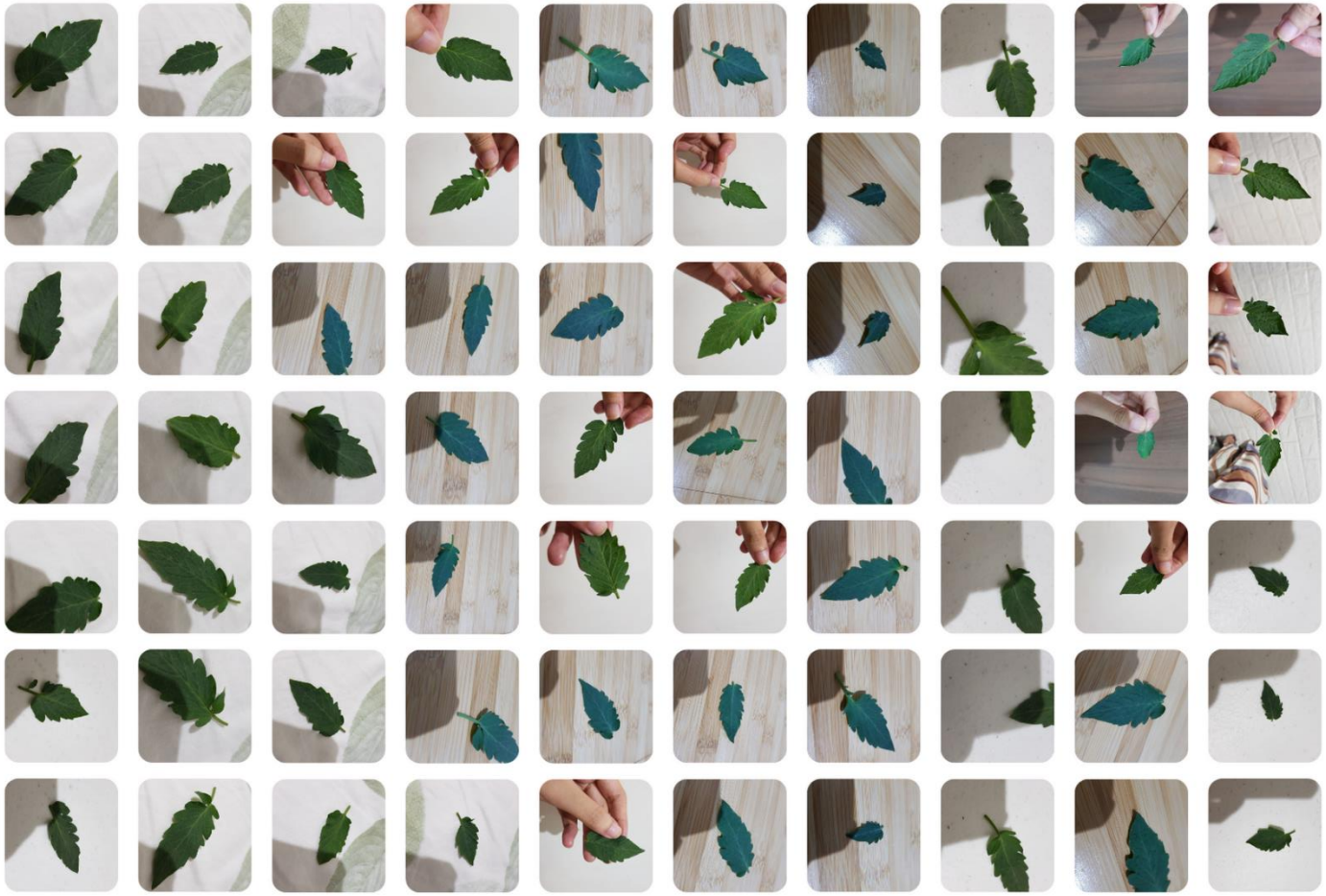




APPENDIX H
Dataset

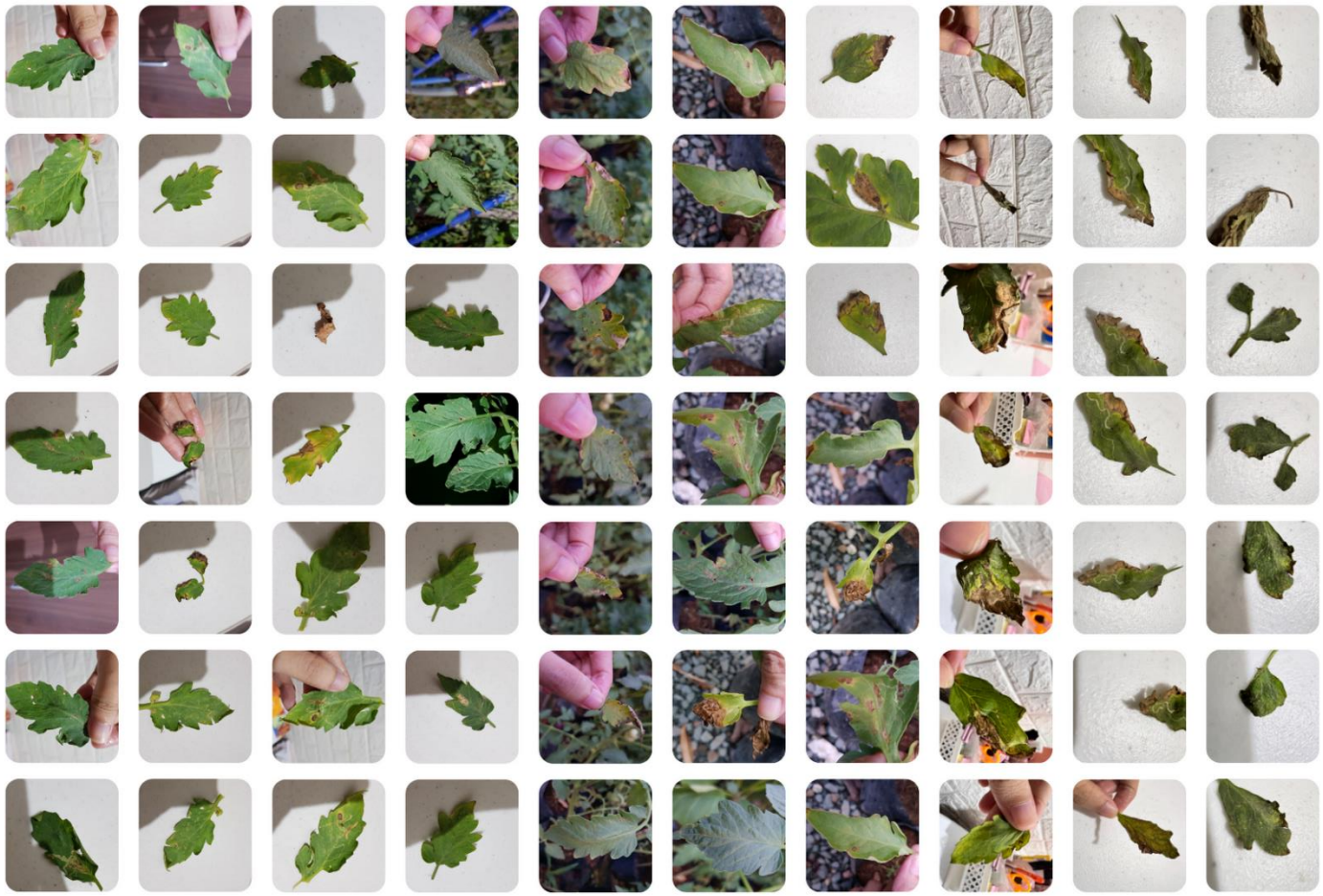
HEALTHY LEAF

HEALTHY LEAF



DISEASED LEAF

DISEASED LEAF



Management Recommendations

HEALTHY LEAF

Symptoms:

Healthy tomato leaves typically exhibit vibrant green coloration with no visible signs of damage or discoloration.

Organic Control:

To maintain leaf health organically, regular pruning of affected foliage and ensuring adequate spacing between plants to promote air circulation can help prevent disease. Alternatively, applying neem oil or a garlic-based spray can act as organic deterrents against pests and pathogens.

Chemical Control:

For chemical control, fungicides containing chlorothalonil or copper-based compounds can effectively prevent fungal infections and maintain leaf health.

DISEASED LEAF

Symptoms:

Diseased tomato leaves may show symptoms such as yellowing, browning, or spotting, indicating the presence of a fungal or bacterial infection.

Organic Control:

To combat disease organically, removing infected leaves promptly and ensuring proper sanitation practices in the garden can help prevent further spread. Additionally, applying a mixture of baking soda and water or a compost tea solution can help suppress fungal growth.

Chemical Control:

Chemical control involves using fungicides like maneb or captan, which effectively target fungal pathogens and halt the progression of disease on tomato leaves.

APPENDIX I
Curriculum Vitae

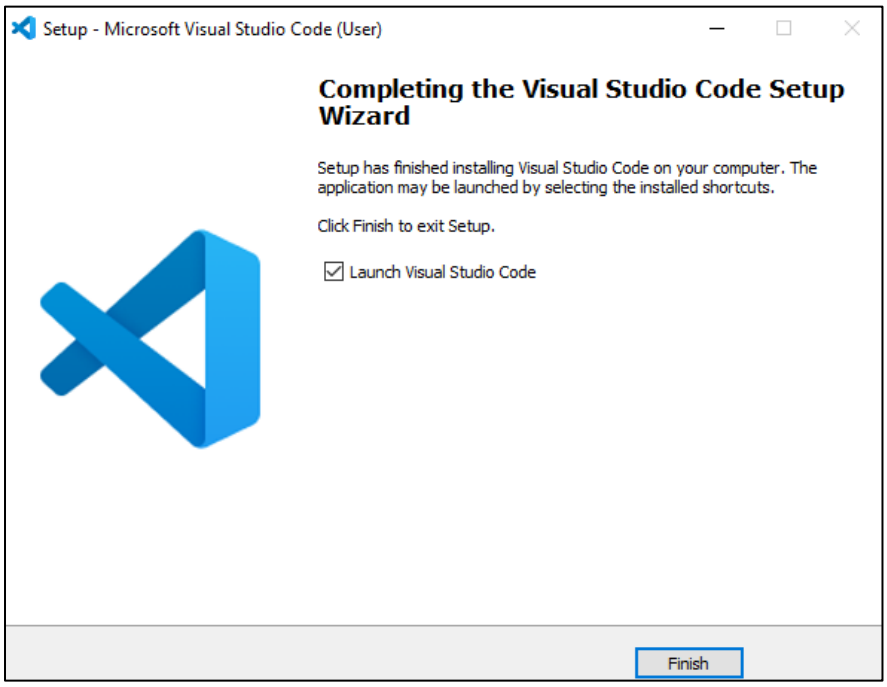
APPENDIX J
Other Attachments

Tools and Libraries Installation

a. Software Installation

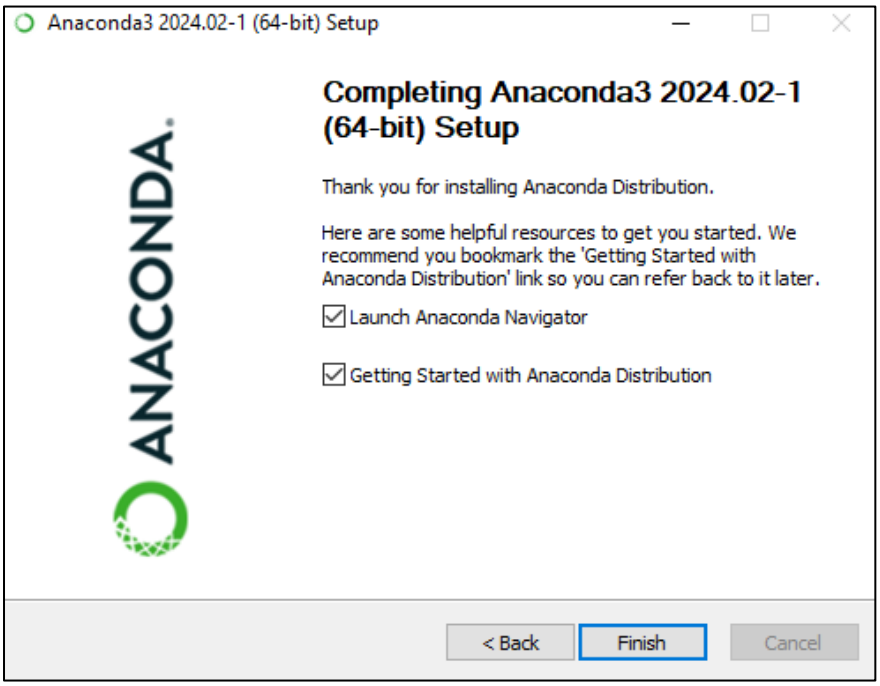
Visual Studio Code

- Used for coding and editing various programming languages, Visual Studio Code is a versatile integrated development environment (IDE).
- Downloaded from https://code.visualstudio.com/download
- Version: 1.84.2



Anaconda Python

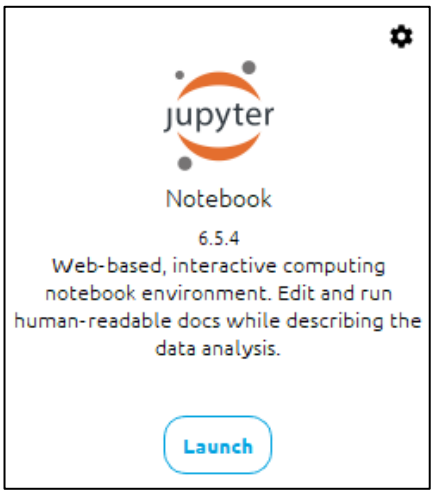
- Anaconda Python is used for managing Python distributions and packages, facilitating data science workflows.
- Downloaded from https://www.anaconda.com/download
- Version: 23.7.4

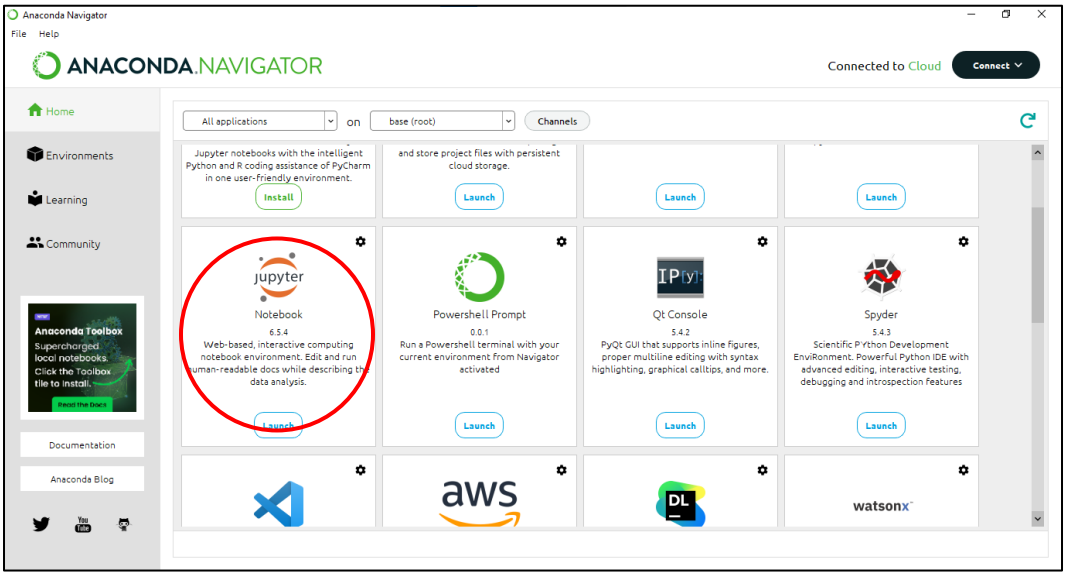


b. Package Installation

Jupyter Notebook

- Used for interactive computing and data visualization, Jupyter Notebook provides a web-based environment for creating and sharing documents. https://jupyter.org/
- Package included in Anaconda Navigator. https://jupyter.org/install
- Version: 6.5.4





OpenCV

- OpenCV is used for computer vision tasks, offering a library of functions for image and video analysis. https://opencv.org/
- Package included in Anaconda. https://opencv.org/get-started/
- Version: 4.9.0.80

Installing OpenCV

Step 1

```
pip install opencv-python
```

Step 2

```
import cv2
```

Pandas

- Pandas is used for data manipulation and analysis, providing powerful data structures and functions. https://pandas.pydata.org/
- Package included in Anaconda. https://pandas.pydata.org/docs/getting_started/install.html
- Version: 2.0.3

Installing Pandas

Step 1

```
pip install pandas
```

Step 2

```
import pandas as pd
```

Matplotlib

- Matplotlib is used for creating static, animated, and interactive visualizations in Python. https://matplotlib.org/
- Package included in Anaconda. https://matplotlib.org/stable/users/installing/index.html
- Version: 3.7.2

Installing Matplotlib

Step 1

```
pip install matplotlib
```

Step 2

```
import matplotlib.pyplot as plt
```

Seaborn

- Seaborn is used for statistical data visualization, providing a high-level interface for drawing attractive and informative graphics. https://seaborn.pydata.org/
- Package included in Anaconda. https://seaborn.pydata.org/installing.html
- Version: 0.12.2

Installing Seaborn

Step 1

```
pip install seaborn
```

Step 2

```
import seaborn as sns
```

Numpy

- Numpy is used for numerical computing in Python, offering support for large, multi-dimensional arrays and matrices. https://numpy.org/
- Package included in Anaconda. https://numpy.org/install/
- Version:1.24.3

Installing Numpy

Step 1

```
pip install numpy
```

Step 2

```
import numpy as np
```

TensorFlow and Keras

- TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. https://www.tensorflow.org/
- Package included in Anaconda. https://www.tensorflow.org/install
- Version: 2.15.0

Installing TensorFlow and Keras

Step 1

```
pip install tensorflow
```

```
pip install keras
```

Step 2

```
import tensorflow as tf
```

```
import keras
```

scikit-learn

- scikit-learn is used for machine learning tasks, offering a wide range of algorithms and tools for data mining and analysis. https://scikit-learn.org/stable/
- Package included in Anaconda. https://scikit-learn.org/stable/install.html
- Version: 1.3.0

Installing scikit-learn

Step 1

```
pip install scikit-learn
```

Step 2

```
from sklearn.metrics import confusion_matrix, classification_report
```

Pillow (PIL)

- Pillow (PIL) is used for image processing tasks in Python, providing support for opening, manipulating, and saving many different images file formats. https://python-pillow.org/
- Package included in Anaconda. https://pillow.readthedocs.io/en/stable/installation.html
- Version: 9.4.0

Installing Pillow (PIL)

Step 1

```
pip show pillow
```

Step 2

```
from PIL import Image
```

Flask

- Flask is used for building web applications in Python, providing a lightweight and modular framework for web development. https://flask.palletsprojects.com/en/3.0.x/
- Package included in Anaconda. https://flask.palletsprojects.com/en/3.0.x/installation/
- Version: 2.2.2

Installing Flask

Step 1

```
pip install flask
```

Step 2

```
from flask import Flask, request, url_for, render_template, redirect
```

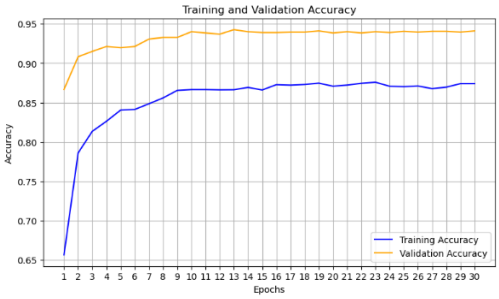
Customized Model's Learning Curve and Evaluation

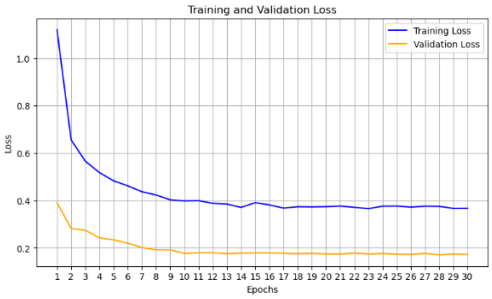
First Attempt

Model Hyperparameter Specifications and Configurations

Attribute	Value
Image Size	224
Channel	3
Batch Size	32
Epoch	30
Regularization	Batch Normalization
Dropout	0.5
Number of hidden layers	2
Number of neurons per layer	1024, 512
Activation	ReLU
Number of Classes	2
Output Layer Activation	softmax
Layers of Model	All layers except the last eight are set to be non-trainable, while the last eight layers are set to be trainable.
Optimizer	Stochastic Gradient Descent (SGD)
Learning Rate	0.001
Loss	categorical cross-entropy
Model Checkpoint, CSV Logger, Learning Rate Scheduler, Early Stopping	Yes

Learning Curve of the Custom DenseNet-121





Training and Validation Evaluation

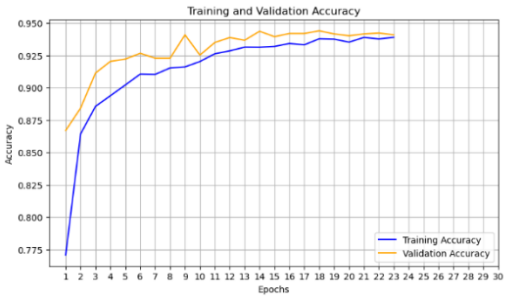
	Train	Validation
Accuracy	0.8742 or 87.42%	0.9411 or 94.11%
Loss	0.3671 or 36.71%	0.1725 or 17.25%

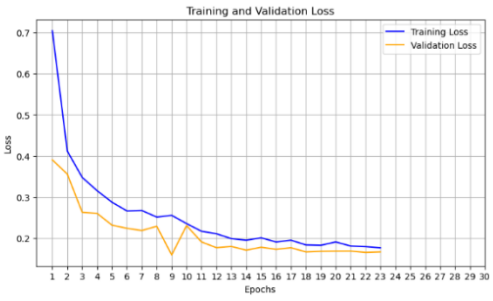
Second Attempt

Model Hyperparameter Specifications and Configurations

Attribute	Value
Image Size	224
Channel	3
Batch Size	32
Epoch	30
Regularization	Batch Normalization
Dropout	0.5
Number of hidden layers	1
Number of neurons per layer	512
Activation	ReLU
Number of Classes	2
Output Layer Activation	softmax
Layers of Model	All layers except the last eight are set to be non-trainable, while the last eight layers are set to be trainable.
Optimizer	Adam
Learning Rate	0.001
Loss	categorical cross-entropy
Model Checkpoint, CSV Logger, Early Stopping	Yes

Learning Curve of the Custom DenseNet-121





Training and Validation Evaluation

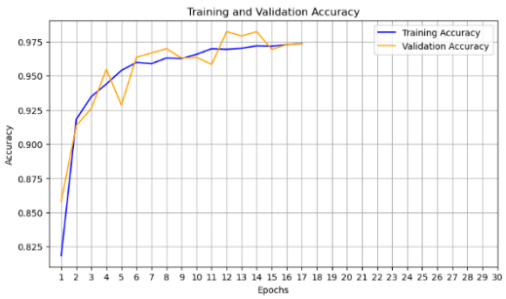
	Train	Validation
Accuracy	0.9391 or 93.91%	0.9410 or 94.10%
Loss	0.1764 or 17.64%	0.1665 or 16.65%

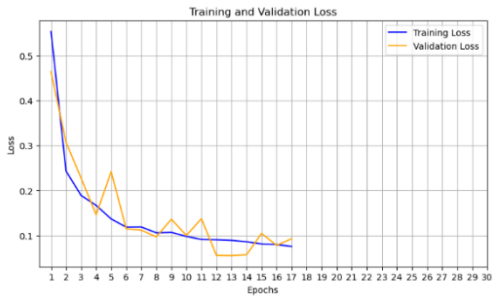
Third Attempt

Model Hyperparameter Specifications and Configurations

Attribute	Value
Image Size	224
Channel	3
Batch Size	32
Epoch	30
Regularization	Batch Normalization
Dropout	0.5
Number of hidden layers	1
Number of neurons per layer	512
Activation	ReLU
Number of Classes	2
Output Layer Activation	softmax
Layers of Model	Set the layers of the base model so that all layers except the last ten are made non-trainable.
Optimizer	Adam
Learning Rate	0.001
Loss	categorical cross-entropy
Model Checkpoint, CSV Logger, Early Stopping	Yes

Learning Curve of the Custom DenseNet-121





Training and Validation Evaluation

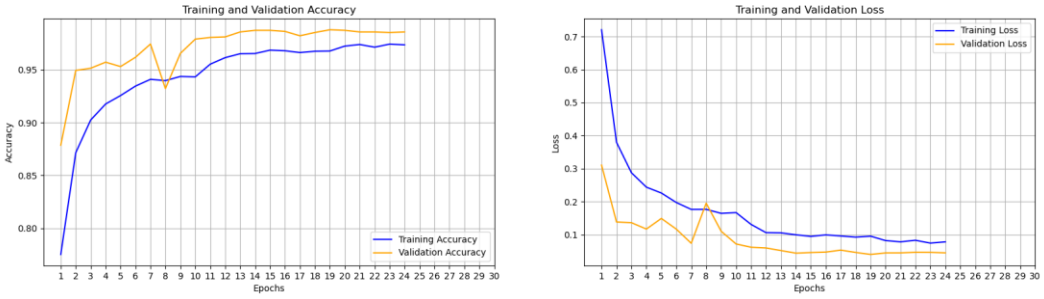
	Train	Validation
Accuracy	0.9736 or 97.36%	0.9734 or 97.34%
Loss	0.0753 or 7.53%	0.0925 or 9.25%

Final Attempt

Model Hyperparameter Specifications and Configurations

Attribute	Value
Image Size	224
Channel	3
Batch Size	32
Epoch	30
Regularization	Batch Normalization
Dropout	0.5
Number of hidden layers	2
Number of neurons per layer	1024, 512
Activation	ReLU
Number of Classes	2
Output Layer Activation	softmax
Layers of Model	Set the layers of the base model so that all layers except the last ten are made non-trainable.
Optimizer	Adam
Learning Rate	0.001
Loss	categorical cross-entropy
Model Checkpoint, CSV Logger, Learning Rate Scheduler, Early Stopping	Yes

Learning Curve of the Custom DenseNet-121



Training and Validation Evaluation

	Train	Validation
Accuracy	0.9737 or 97.37%	0.9859 or 98.59%
Loss	0.0782 or 7.82%	0.0448 or 4.48%