

A Project Report

On

DIGITAL PHENO-PARENTING: ENHANCED MODEL FOR PLANT-PHENOTYPES USING DEEP LEARNING

Submitted to

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
ANANTAPUR, ANANTHAPURAMU**

In Partial Fulfillment of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

In

**COMPUTER SCIENCE & ENGINEERING (ARTIFICIAL
INTELLIGENCE)**

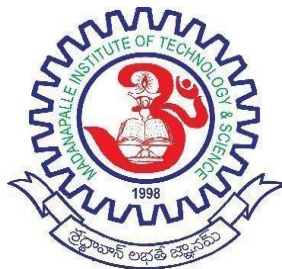
Submitted By

B. Bhuvan Kumar	—	20691A3108
M. Venkatesh	—	20691A3160
S. Nameera	—	20691A3135
C.R. Rohith Reddy	—	20691A3145

Under the Guidance of

Mr. K. Mahammad, M. Tech.
Assistant Professor

Department of Computer Science & Engineering (Artificial Intelligence)



**MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE
(UGC – AUTONOMOUS)**

(Affiliated to JNTUA, Ananthapuramu)

Accredited by NBA, Approved by AICTE, New Delhi)

AN ISO 21001:2018 Certified Institution

P. B. No: 14, Angallu, Madanapalle – 517325

2023-2024



MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE

(UGC – AUTONOMOUS)

Approved by AICTE, New Delhi and Affiliated to JNTUA, Ananthapuramu

www.mits.ac.in www.mits.edu

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
(ARTIFICIAL INTELLIGENCE)**

BONAFIDE CERTIFICATE

This is to certify that the project work entitled “ **DIGITAL PHENO-PARENTING:
ENHANCED MODEL FOR PLANT-PHENOTYPES USING DEEP LEARNING**”
is a bonafide work carried out by

B. Bhuvan Kumar	–	20691A3108
M. Venkatesh	–	20691A3160
S. Nameera	–	20691A3135
C. R. Rohith Reddy	–	20691A3145

Submitted in partial fulfillment of the requirements for the award of degree
Bachelor of Technology in the stream of **Computer Science & Engineering
(Artificial Intelligence)** in Madanapalle Institute of Technology & Science,
Madanapalle, affiliated to Jawaharlal Nehru Technological University
Anantapur, Ananthapuramu during the academic year 2023-2024.

Guide
Mr .K. Mahammad, M. Tech.
Assistant Professor
Department of CSE-AI

Head of the Department
Dr. K. Chokkanathan, ME., Ph.D
Associate Professor
Department of CSE-AI

Submitted for the University examination held on:

Internal Examiner
Date:

External Examiner
Date:

ACKNOWLEDGEMENTS

We sincerely thank the **MANAGEMENT of Madanapalle Institute of Technology & Science** for providing excellent infrastructure and lab facilities that helped me to complete this project.

We sincerely thank **Dr. C. Yuvaraj, Principal** for guiding and providing facilities for the successful completion of our project at **Madanapalle Institute of Technology & Science, Madanapalle.**

We express our deep sense of gratitude to **Dr. K. Chokkanathan, Associate Professor & Head of the Department of CSE-AI** for his continuous support in making necessary arrangements for the successful completion of the Project.

We express our deep gratitude to my guide **Mr. K. Mahammad, Assistant Professor, Department of CSE-AI** for his guidance and encouragement that helped us to complete this project.

We express my deep sense gratitude to **Dr. Vamsi Bandi, Assistant Professor, Project Coordinator, Department of CSE-AI** for his valuable guidance and encouragement that helped us to complete this project.

We also wish to place on record my gratefulness to other **Faculty members of Department of CSE-AI** and to our friends and our parents for their help and cooperation during our project work.



MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE

(UGC-AUTONOMOUS INSTITUTION)

Affiliated to JNTUA, Ananthapuramu & Approved by AICTE, New Delhi

NAAC Accredited with A+ Grade, NIRF India Rankings 2022 - Band: 251-300 (Engg.)

NBA Accredited - B.Tech. (CIVIL, CSE, ECE, EEE, MECH), MBA & MCA

www.mits.ac.in




RECOGNISED RESEARCH CENTER

Plagiarism Verification

Certificate

This is to certify that the B. Tech Project report titled, “**DIGITAL PHENO-PARENTING: ENHANCED MODEL FOR PLANT-PHENOTYPES USING DEEP LEARNING**” submitted by **B.Bhuvan Kumar(20691A3108)**, **M. Venkatesh(20691A3160)**, **S. Nameera(20691A3135)** and **C. R. Rohith Reddy(20691A3145)** has been evaluated using **Anti-Plagiarism Software, TURNITIN** and based on the analysis report generated by the software, the report's similarity index is found to be 17%.

The following is the **TURNITIN** report for the project report consisting of 66 pages.

		Similarity Report ID: oid:3618:58476186	
PAPER NAME		AUTHOR	
Digital pheno parenting (1)		Rohith C	
WORD COUNT		CHARACTER COUNT	
9966 Words		61163 Characters	
PAGE COUNT		FILE SIZE	
66 Pages		5.9MB	
SUBMISSION DATE		REPORT DATE	
Apr 30, 2024 3:08 PM GMT+5:30		Apr 30, 2024 3:09 PM GMT+5:30	

● 17% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 11% Internet database
- 4% Publications database
- Crossref database
- Crossref Posted Content database
- 13% Submitted Works database

DECLARATION

We hereby declare that the results embodied in this project “**DIGITAL PHENO-PARENTING: ENHANCED MODEL FOR PLANT PHENOTYPES USING DEEP LEARNING**” by us under the guidance of **Mr. K. Mahammad, Assistant Professor, Department of CSE-AI** in partial fulfillment of the award of **Bachelor of Technology in Computer Science & Engineering (Artificial Intelligence)** from **Jawaharlal Nehru Technological University Anantapur, Ananthapuramu** and we have not submitted the same to any other University/institute for award of any other degree.

Date :

Place : Madanapalle

PROJECT ASSOCIATES

B. BHUVAN KUMAR

M. VENKATESH

S. NAMEERA

C. R. ROHITH REDDY

I certify that above statement made by the students is correct to the best of my knowledge.

Date

GUIDE

Mr. K. MAHAMMAD

ASSISTANT PROFESSOR

INDEX

S.NO	TOPIC	PAGE NO
1.	INTRODUCTION	
	1.1 Motivation	1
	1.2 Problem Definition	1
	1.3 Objective of the Project	2
	1.4 Limitations of Project	2
	1.5 Organization of Documentation	3
2.	LITERATURE SURVEY	
	2.1 Introduction	5
	2.2 Existing System	5
	2.3 Disadvantages of Existing System	10
	2.4 Proposed System	10
	2.5 Advantages over Existing System	11
3.	ANALYSIS	
	3.1 Introduction	13
	3.2 Software Requirement Specification	13
4.	DESIGN	
	4.1 Introduction	18
	4.2 ER/UML Diagrams	18
	4.3 Module Design and Organization	21
	4.4 Summary	21
5.	IMPLEMENTATION AND RESULTS	
	5.1 Introduction	24
	5.2 Implementation of key functions	25
	5.3 Method of Implementation	26
	5.4 Output Screens and Result Analysis	42
	5.5 Summary	45

6.	TESTING AND VALIDATION	
	6.1 Introduction	47
	6.2 Design of Test Cases and Scenarios	47
	6.3 Validation	49
	6.4 Summary	54
7.	CONCLUSION	
	7.1 Conclusion	56
	7.2 Future Enhancement	56
8.	REFERENCES	57

ABSTRACT

Artificial Intelligence (AI) is increasingly being used in agriculture to improve efficiency, productivity, and sustainability in various ways, one of the key applications of AI in agriculture is crop management and the detection and management of pests and diseases. Traditionally, plant phenotyping has been a manual and time-consuming process, limited by the subjective nature of visual assessments. However, the emergence of deep learning algorithms, have opened up new possibilities for enhancing plant phenotyping. The previous works where AI is used for pheno-parenting have less accurately detected disease of the plants and have used non-commercial plants, pre-trained models. In this work we aim to propose an approach called Digital Pheno-Parenting, that leverages deep learning techniques to improve the accuracy and efficiency of plant phenotyping. This Digital Pheno-Parenting involves a three-step process. Firstly, a large dataset of images of plant leaves is collected. These images contain various phenotypic traits such as leaf area, leaf color etc.. Next, multiple image classification deep learning models are fine-tuned and then employed to form a single concatenated model for analyzing the images and extracting meaningful features that accurately represent the leaf phenotypes. These extracted features are then used to predict various plant traits such as specie and health condition of the plant. And based on the predicted condition of a particular plant, a set of actions are given as parenting suggestions that can help those plant to give better yields.

Keywords : *Plant Pheno-parenting, CNN, feature extraction, parenting suggestion*

LIST OF FIGURES

Figures	Title	Page No
2.1	Architecture Of Proposed System	11
4.1	Data Flow Diagram	19
4.2	Sequence Diagram	20
4.3	Use Case Diagram	20
5.1	Starting stramlit server	42
5.2	User Interface	43
5.3	Uploading an Leaf Image	43
5.4	Displaying the uploaded image.	44
5.5	Displaying the specie, disease and suggestion	44
6.1	Healthy plant	48
6.2	Unhealthy plant with suggestion	48

LIST OF TABLES

Tables	Title	Page No
2.1	Summary of existing works	7
6.1	ResNet Validation Accuracy	49
6.2	EfficientNet Validation Accuracy	50
6.3	MobileNet Validation Accuracy	51
6.4	LeafNet Validation Accuracy	52
6.5	Model Accuracy	54

CHAPTER-1

INTRODUCTION

CHAPTER-1

INTRODUCTION

1.1 MOTIVATION

In the realm of agriculture, the quest for sustainable solutions to meet the growing demands of a burgeoning population has never been more pressing. Conventional plant breeding techniques, while effective, are often time-consuming, resource-intensive, and limited in their ability to rapidly respond to dynamic environmental challenges. In this landscape, the emergence of Digital Pheno Parenting represents a transformative paradigm shift, offering unparalleled opportunities to revolutionize crop improvement processes and drive agricultural innovation forward.

The conventional plant breeding process involves extensive manual observations and measurements, which are time-consuming and often limited in scalability. The Digital Pheno Parenting model aims to address these limitations by leveraging AI techniques, specifically deep learning, to automate and enhance the phenotyping process. By harnessing the power of AI, this product aspires to redefine the way plant traits are analyzed, paving the way for accelerated crop improvement and the development of resilient, high-yielding varieties.

Furthermore, the Digital Pheno Parenting model is motivated by the desire to empower researchers, agronomists, and plant breeders with sophisticated decision support tools. By providing actionable insights derived from the analysis of vast datasets, the Enhanced Model facilitates informed decision-making in selecting parental combinations, optimizing breeding strategies, and aligning crop development with specific agricultural objectives.

1.2 PROBLEM DEFINITION

- Manual phenoparenting, while once the standard practice in agriculture, is fraught with various challenges that hinder efficiency, accuracy, and scalability.
- Manual phenoparenting requires a significant amount of human labor, involving tasks such as manual measurements, observations, and data recording. This labor-intensive process is time-consuming and costly, particularly when dealing with large populations of plants. Human observers may exhibit subjective biases in phenotypic trait assessment, leading to inconsistencies in data collection. This manual approach increases the risk of data errors, loss, and inconsistency, impeding data-driven decision-making in breeding programs.
- Scaling up manual phenotyping to handle large experimental fields becomes challenging. The process may not be easily adaptable to accommodate the growing demand for increased crop productivity and genetic diversity.
- There requires an Automated phenotyping using AI algorithms that ensures objective and consistent measurements, reducing the impact of human subjectivity and bias in data collection. and also significantly reducing the time and labor required for data collection, thereby accelerating breeding programs and research initiatives. Advanced imaging technologies and real-time monitoring capabilities in automated systems enable the capture of dynamic changes in plant traits, providing insights into plant responses to environmental stresses.

1.3 OBJECTIVE OF THE PROJECT

The objective of a Digital Pheno Parenting focused on daily used plants and yield production index creation is to revolutionize agricultural practices by leveraging advanced technologies like deep learning to enhance crop yield prediction, optimize resource allocation, and improve overall productivity. Specifically, the project aims to achieve the following objectives:

- To employ leaf images of commonly cultivated plants for conducting pheno-parenting.
- To develop a concatenated model of deep learning models like ResNet, EfficientNet, MobileNet for predicting the specie and disease.
- To give plant parenting suggestions as future course of action based on the specie and disease detected from the input image.
- To implement a user-friendly interface to facilitate seamless interaction and decision making for users.

1.4 LIMITATIONS OF THE PROJECT

- **Data Collection:** Having the need of data that is related to complete lifecycle of plants in a farm, collection such data is hard and extremely time consuming.
- **Data Quality and Accuracy:** The accuracy and reliability of the AI system's predictions and recommendations depend on the quality of input data, including plant phenotypic data, environmental parameters, and historical yield records. Inaccuracies or inconsistencies in data collection, processing, or interpretation could lead to erroneous insights and decision-making.
- **Model Generalization:** AI models trained on specific datasets and environmental conditions may struggle to generalize their predictions to new or diverse agricultural settings. Variability in crop genetics, soil types, climate patterns, and management practices across different regions could limit the applicability and effectiveness of the AI system in providing accurate recommendations for all users.
- **User Acceptance and Adoption:** Farmers and agricultural stakeholders may exhibit resistance or skepticism towards adopting AI-driven technologies due to factors such as lack of familiarity, perceived complexity, or cultural barriers. Effective communication, training, and support initiatives are necessary to promote user acceptance and encourage widespread adoption of the Digital Pheno Parenting AI project.
- **Environmental Variability:** Unpredictable environmental events such as extreme weather conditions, natural disasters, or sudden climate changes can impact crop performance. The AI model may struggle to adapt quickly to such unforeseen circumstances, affecting the accuracy of yield predictions.

1.5 ORGANIZATION OF DOCUMENTATION

The project documentation, including the literature review, analysis, design, implementation and results, testing, and validation, is presented in this section. This is how the report is set up. The basic introduction, problem definition, project purpose, and limits are covered in

Chapter 1: Describes the motivation of the work, definition of the problem delves into the limitations of the project.

Chapter 2: The Literature Survey is a thorough examination of previous research, studies, and publications pertinent to the project issue. The basis for the phases of analysis and design is laid out in this section, which also highlights the most significant and pertinent information on the project.

Chapter 3: The Analysis part contains a thorough analysis of the project's specifications and goals. The project's functional and non-functional requirements are listed in this section, together with the project's scope definition.

Chapter 4: The design models are included in the solution architecture and the project design is described in the Design section. Also, Data Flow, Sequence and User case diagrams have been shown.

Chapter 5: Implementation of the suggested system and outcomes is covered in This section outlines the project's implementation strategy, difficulties faced, and end results.

Chapter 6: Model testing is covered in the testing procedure required to confirm that the solution satisfies the specified requirements is described in the Testing and Validation section.

Chapter 7: In conclusion, the documentation of a project is critical to its success and complete project is concluded in this section.

CHAPTER-2

LITERATURE SURVEY

CHAPTER-2

LITERATURE SURVEY

2.1 INTRODUCTION

The field of agriculture is a multifaceted domain that demands precision and efficiency in plant phenotyping, a critical aspect in crop improvement programs. Traditional methods for monitoring plant traits are often labor-intensive and prone to inaccuracies, leading to suboptimal outcomes. To address these challenges, a cutting-edge digital pheno parenting project is proposed, leveraging advanced technologies such as deep learning algorithms and data analytics. This innovative solution aims to revolutionize plant phenotyping by offering precise and automated analysis of key traits throughout the plant life cycle. By harnessing the power of modern technology, the project seeks to empower farmers and researchers with actionable insights, ultimately enhancing agricultural productivity and sustainability.

2.2 EXISTING SYSTEM

In [1], the study presents a novel method utilizing YoloV5 and PointNet for predicting grain count in sorghum panicles by integrating global features from point cloud models and grain counts detected from RGB images. The models achieved a mean absolute percent error of 6.5% and 6.8% for high and low-resolution datasets, respectively, demonstrating the feasibility of multimodal approaches for accurate estimation of grain count.

In [2], the study explores the use of point clouds and deep learning models for high-throughput phenotyping of sorghum plants. PointNet++ achieved the best segmentation results with 91.5% accuracy, correlating well with manually measured phenotypic traits, demonstrating rapid and accurate trait extraction capabilities.

In [3], authors implement deep learning techniques for phenotyping stem diameter, branch angle, and branch diameter of loblolly pine trees. Results show accurate estimations with RMSEs of 0.055 m, 5.0°, and 5.6 mm respectively, indicating potential for a low-cost, high-throughput phenotyping tool in tree improvement programs.

In [4], the paper examines the effectiveness of ResNet-based CNN architecture in two plant phenotyping tasks: species recognition (SR) and infection detection. This work addresses imbalanced datasets, employs data augmentation, and evaluates multiclass classifier parameters. ResNet 20 (V2) architecture has performed significantly well in the tasks of Species Recognition (SR) and Identification of Healthy and Infected Leaves (IHIL) with a Precision of 91.84% and 84.00%, Recall of 91.67% and 83.14% and F1 Score of 91.49% and 83.19%, respectively.

In [5] authors employed three different models: KNN, Naïve Bayes and Visual Geometry Group (VGG-16) to extract distinctive features of plant and perform multiclass classification to categorize plant species. The study is conducted on three different datasets, namely Flavia, Swedish, and the intelligent computing laboratory (ICL) datasets.

In [6] authors have introduced a high-precision phenotyping technique for rice panicles using a Faster R-CNN-based model. This approach eliminates the need for separation and threshing and allows for the effective extraction of key rice traits using visible light scanning imaging and deep learning, to evaluate the quality and yield potential.

In [7] the authors proposed a comparative study of various deep learning models for identification and classification of various plant stresses and diseases. The study employed various models including, CNN, DNN, SRCNN and uses datasets of tomato diseases dataset, wheat disease dataset and cucumber viral diseases dataset.

In [8] the authors discussed the use of ResNet20 in species recognition and identification of health condition. The study outperforms benchmarked machine learning models that use random forest and support vector machines.

In [9] this study, they have Developed a self-supervised CNN method for segmenting overlapping plants in field images, enabling high-throughput phenotyping. Implemented a pipeline to extract plant heights over time, estimating growth curves, and conducted functional data analysis on maize plants to reveal genotype effects on growth patterns.

In [10] this report the author used Multi-layer perceptron(MLP) to train baseline networks on agrometeorological data, which consist of numerical or time series data. Additionally, CNN were utilized to process image data.

In [11], a small-scale hydroponic system was utilized to collect and analyze image data of petunia, pansy, and calendula plants. Employing a Deep Neural Network (DNN), the research focused on plant species recognition, growth analysis, health assessment, and yield stage identification. Calendula species were accurately recognized with over 95% detection accuracy, while GDI metric provided insights into plant growth and productivity.

In [12] they use a data-driven deep learning approach to accurately identify and count wheat ears/spikes in digital images taken in an open field environment. We used two variants of FasterRCNN, EfficientDet-D5, and EfficientDet-D7, to detect the target ears/spikes in the wheat crop images. They achieved an accuracy of 88.7% using Faster-RCNN, 92.7% accuracy on EfficientDet-D5, and 93.6% accuracy on efficientDet-D7, respectively.

In [13] this study conducted experiments to highlight the significance of Data Augmentation practices for limited data sets with narrow distributions. The authors introduced techniques like cut and paste, Graphical modeling and Generative Adversarial Networks, which showed that data augmentation is significant to regularize the models trained on limited data sets.

In [14] authors demonstrated a high-throughput phenotyping tool, to accurately capture high night temperature induced chalkiness in rice and used a fully automated approach based on CNN to distinguish between chalky and non-chalky grains and subsequently identifying the area of a grain that is indicative of the chalky class using Gradient-weighted Class Activation Mapping (Grad-CAM).

In [15] authors introduced a CNN based open-source deep learning tool called Deep Plant Phenomics, which provides pre-trained neural networks for performing several common plant phenotyping tasks. The dataset used was IPPN image-based plant phenotyping .

In [16] the authors used deep learning for plant recognition and segmentation, extracting features with modest heritability important in breeding, and developed a Raspberry Pi camera setup and Greenotyper pipeline to monitor 1800 white clover plants, taking 350,000 photos.

In [17] the study proposed a CNN-based deep learning method that shows the potential of multi-task deep architectures for plant phenotyping by precisely localizing wheat spikes and spikelets and performing simultaneous image classification. The authors presented a new dataset, called ACID, that provides hundreds of accurately annotated images of wheat spikes and spikelets, along with image level class annotation.

In [18] the study presented a multi-temporal dataset with manually labeled high-resolution 3D point clouds of tomato and maize plants for computer vision tasks like 3D reconstruction and segmentation which offers the potential to derive various plant traits like biomass, plant height, the quantity and size of pertinent plant organs.

In [19] this study discussed the utilization of YOLOv5-based single-stage detectors, either independently or in ensembles, for effectively detecting tomato plant phenotyping traits, including nodes, fruit, and flowers, in a challenging dataset obtained from a stress experiment across various tomato genotypes.

Table 2.1: Summary of existing works

S. No	Author	Methodology	Model	Limitations
1	Anirban Jyoti Hati et al. 2023 [6]	Deep Neural Networks (DNN) are utilized to perform crucial tasks such as identifying plant species, analyzing growth, and determining the yield stage	Yolo V3	Difficulty in detecting stressed plant or dead plant accurately
2	A.J. Hati et al. 2021 [13]	A convolutional neural network (CNN) based on the second version of the Residual Network (ResNet)	ResNet 20 (V2)	Species recognition is based on only leaves which may be identical for plants of same family
3	Asheesh Kumar Singh et al. 2018 [17]	Several deep learning models were employed in this study for the automatic detection and categorization of different types of plant diseases and stress conditions	SRCNN, DNN, CNN	Model performance is reduced when tested on images from real field conditions and training data from complex field environments is required to improve robustness.
4	Radek Zenkl et al. 2022 [10]	In this research, ResNet 20 (V2) was employed for the identification of species and the detection of their health status.	ResNet	The performance of the method is will improve more with labeled data and addressing dataset shortcomings, emphasizing the need for high-resolution images.

S. No	Author	Methodology	Model	Limitations
5	Angelo Cardellicchio et al. 2023 [7]	Identifying phenotyping characteristics in tomato plants through the use of single-stage detection systems.	YOLOv5	The balance between model intricacy and efficiency was not optimized, and attention layers were not incorporated.
6	Deepti Barhate et al. 2023 [5]	Capturing images, extracting features, and classifying for the identification of plant species using various deep learning models.	VGG-16, KNN,NB	The system cannot detect occluded leaves and does not include weed species detection.
7	Ehsan Ullah et al 2022 [11]	Utilizing EfficientDet and Faster-RCNN for the identification and enumeration of spikes.	Faster R-CNN with Resnet50, EfficientDet-D5, EfficientDet-D7	No usage of data during the training phase with varying illuminations and conditions like occlusions, overlapping, blur
8	Chaoxin wang et al. 2022 [12]	This study employed Convolutional Neural Networks to differentiate between chalky and non-chalky grains, then identified the chalky grain using used Grad-CAM.	CNN, Grad-CAM	The method was developed to assess overall grain chalkiness, but it overlooks distinct types of chalkiness such as white-belly, white core, or white-base.
9	Chrisbin James et al. 2024	Multimodal approach based on point clouds and RGB images	PointNet, YoloV5	Each Image contain only one panicle (No localization of panicles)
10	Ajay Kumar Patel et al. 2023	Leica BLK360 Imaging scanner, LiDAR (light detection and ranging) sensor	PointNet, PointNet++, PointCNN	No Growth Analysis or Yield Identification
11	Nariman Niknejad et al. 2023	3D imaging, deep learning-based instance segmentation, and image and point cloud processing techniques	Mask R-CNN	Quality of stem and branches is not estimated based on stress, color
12	Yuwei Lu et al. 2023	Grain phenotyping based on visible light scanning imaging and deep learning technology.	R-CNN, Pix2Pix	Incorporating more occlusion scenes and varying occlusion degrees to boost accuracy,
13	J. Dhakshayani et al.2023	Multi layer perceptron to train unimodal baseline networks on agrometeorological data and CNN for image data	MLP, CNN, MLP+CNN	Not integrated of IOT devices, which holds great potential for enhancing the practical implications of this work

S. No	Author	Methodology	Model	Limitations
14	Yumou Qiu et al.2023	Separates overlapping foreground and background plants, enabling accurate height estimation and efficient assessment of treatment and genotype impacts on plant growth in field environments using computer vision techniques.	CNN and simple linear iterative clustering(slic)	Due to space limitations they fail when plants overlap and cannot provide complete growth curve estimates
15	David Schunck et al. 2021	Collecting point clouds using Laser scanning system and segmenting data into classes using semantic and instance segmentation.	PointNet. PointNet++, LatticeNet	The current dataset excels in plant quantity, measuring dates, and data accuracy, yet further enhancements could be achieved by incorporating background details like genotypes, phenology.
16	Douglas Pinto Sampaio Gomes et al. 2020	Data augmentation using cut & paste, Graphical modelling, and Generative network methods.	RCNN	Augmentations on the development set, with the same distribution as training, demonstrate underperformance, indicating that augmentations have a regularizing effect on models.
17	Marmi Tausen et al. 2020	This work comprises the camera system setup and image analysis pipeline for result generation and details on experimental tests conducted to monitor the systems performance in greenhouse.	U-net	Prone to batch effects with multiple assessors.

2.3 DISADVANTAGES OF THE EXISTING SYSTEM

There are several issues with the current system that reduce its effectiveness. First of all, high resolution images are essential to the system's performance as low-quality images can prone to errors. Subsequently, the current data available may not be adequate or sufficiently representative to enable accurate results. Thirdly, the process requires lot of resources and time for collection of data, which might result in inefficiencies and delays. These limitations highlight the need for more advanced and automated methods to overcome them and improve the accuracy and potency of models.

There exist certain limitations with the studies cited within the current system. An example would be “High-throughput and separating-free phenotyping method for on-panicle rice grains based on deep learning” [6] the accuracy of the model can be boosted by incorporating more occlusion scenes and varying occlusion degrees. “Deep Learning for Plant Stress Phenotyping: Trends and Future Perspectives” [7] where the performance of the model is reduced when tested on images from real field conditions. The dataset used in “Outdoor Plant Segmentation with Deep Learning for High-Throughput Field Phenotyping on a Diverse Wheat Dataset” [8] requires more proper labeling and high resolution images. Comparable to the earlier example “Deep Plant Phenomics: A Deep Learning Platform for Complex Plant Phenotyping Tasks” [15] uses a small dataset, which might lead to overfitting and model becomes less reliable. “AI-Driven Pheno-Parenting: A Deep Learning Based Plant Phenotyping Trait Analysis Model on a Novel Soilless Farming Dataset” [11] finds difficulty in detecting stressed plant or dead plant accurately. “Detection of tomato plant phenotyping traits using YOLOv5-based single stage detectors” [19] requires more denser models to reduce false negatives.

2.4 PROPOSED SYSTEM

The proposed Digital Pheno Parenting system aims to revolutionize agricultural practices by leveraging cutting-edge technologies, particularly deep learning, to enhance plant phenotyping and yield prediction for daily used plants. The integration of advanced AI-driven systems, comprehensive yield production indexing, and plant parenting suggestions will form the foundation of this transformative agricultural solution.

The heart of the proposed system lies in the development of AI-driven systems capable of monitoring plant growth and development. Leveraging advanced imaging techniques, such as high-resolution cameras, the system will capture real-time data on a myriad of plant phenotypic traits. These traits include but are not limited to leaf characteristics, growth patterns, stress responses, and overall yield quantity. The use of deep learning models, trained on diverse datasets of daily used plants, will enable precise analysis, ensuring a comprehensive understanding of plant development over time.

This system will offer personalized plant parenting suggestions based on the condition of the plant in the given image. Utilizing the vast dataset of daily used plants, the deep learning models will not only assess the current state of the plant but also provide tailored recommendations for optimal care and maintenance. This could include insights on irrigation schedules, nutrient requirements, pest management, and other cultivation practices. By integrating this feature, the

system becomes a valuable decision support tool for farmers, helping them implement precise and efficient plant management strategies.

The proposed Digital Pheno Parenting system stands at the forefront of agricultural innovation, leveraging advanced deep learning models to enhance plant phenotyping, yield prediction, and personalized plant parenting suggestions. By addressing the objectives of daily plant monitoring, yield production indexing, and tailored suggestions, the system aims to empower farmers with unprecedented insights, ultimately contributing to increased agricultural productivity and sustainability.

In the Fig 2.1, we can understand the architecture of the proposed system in which multiple pre-trained models are employed to form a single concatenated model that predicts the condition of the plant and then based on its prediction further parenting suggestions are given as future course of action.

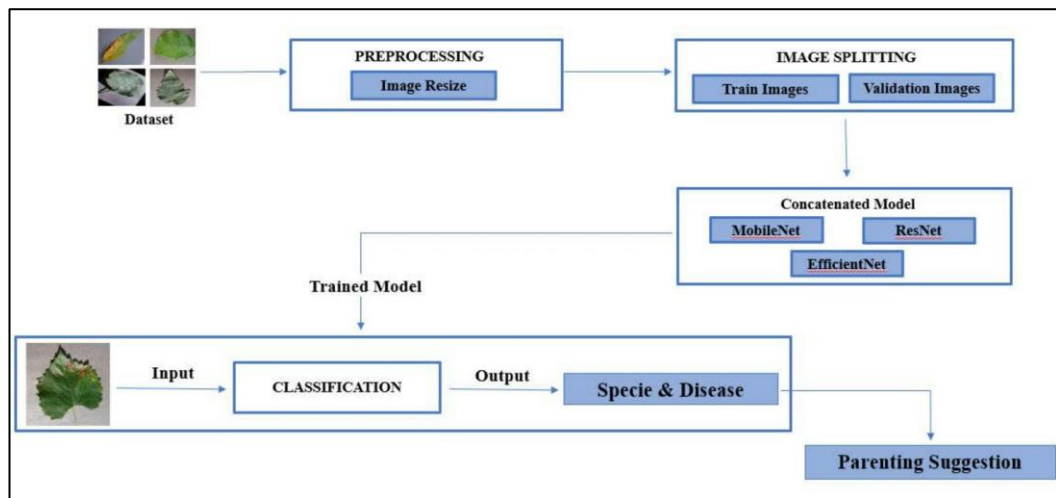


Fig 2.1: Architecture of Proposed System

2.5 ADVANTAGES OF THE PROPOSED SYSTEM

Proposed system has various advantages to the farmers, few of which are:

- **Optimal Resource Allocation:** Efficient allocation of resources based on yield production index, maximizing productivity while minimizing waste.
- **Personalized Plant Care:** Tailored recommendations for plant management, optimizing cultivation practices.
- **Improved Breeding Programs:** Evaluation of plant varieties for superior yield properties, enhancing breeding efforts for future crop improvement.
- **The comprehensive yield production index** contributes to increased agricultural productivity, sustainable practices.

CHAPTER-3

ANALYSIS

CHAPTER-3

ANALYSIS

3.1 INTRODUCTION

Agriculture, a complex and vital industry globally, is undergoing a transformative shift with the introduction of innovative digital pheno parenting technology. This project focuses on leveraging cutting-edge advancements such as deep learning algorithms to revolutionize plant phenotyping. By automating and enhancing the analysis of crucial plant traits throughout their life cycle, the project aims to address challenges in traditional monitoring methods. The utilization of modern technology seeks to empower farmers and researchers with accurate, data-driven insights, ultimately contributing to improved agricultural productivity and sustainability.

3.2 SOFTWARE REQUIREMENT SPECIFICATION

Functional requirements define how a system or its components operate, detailing inputs, behaviors, and outputs. They specify the system's capabilities in terms of computations, data processing, and specialized functionalities. Use cases illustrate the system's behavior in various scenarios, aligning with functional requirements. Complementing these are non-functional requirements, which impose constraints on design or implementation quality. In the context of digital pheno parenting, functional requirements delineate desired system outcomes, guiding system behavior and functionality. These requirements, distinct from non-functional criteria like cost and dependability, drive the technical architecture of the application, shaping its design and capabilities. The objective of the Digital Pheno Parenting project is to provide users with intelligent plant phenotyping solutions. The system comprises a web application with a Python backend and a Streamlit frontend, employing various deep learning methods to analyze and predict plant phenotypic traits. This document outlines the software requirements of the system.

3.2.1 FUNCTIONAL REQUIREMENTS

Image acquisition and processing: Getting an image of a specified plant as input and processing it to extract the features that speaks about plant phenotypic traits.

Feature mapping with Phenotypes: Extracted features are mapped with respective physical characteristics.

Phenotypic Trait Classification: Based on retrieved physical characteristics or phenotypes, that image is taken for phenotypic trait classification.

Prediction and Suggestion: As the plant condition in the input image is predicted, based on predicted values future course of action is recommended.

3.2.2 NON FUNCTIONAL REQUIREMENTS

Accuracy and Reliability: Assurance of high accuracy and reliability in phenotypic processing, and analysis to ensure robust scientific outcomes and breeding decisions. Implementation of quality control measures to minimize errors and inconsistencies in data interpretation.

Front-End: Streamlit will be used to develop User Interface.

Back-End: Python will be used to develop the Deep Learning models and a python module called Flask is used to build backend server.

3.2.3 HARDWARE REQUIREMENTS

System	:	Pentium 4, Intel Core i3, i5 and 2 GHz Minimum
RAM	:	4 GB or above
Hard Disk	:	10 GB or above
Input Device	:	Keyboard and Mouse
Output Device	:	Monitor or PC

3.2.4 SOFTWARE REQUIREMENTS

Operating System	:	Windows 10 or Higher Versions ,Linux
Platform	:	Visual Studio Code ,Jupyter Notebook ,Google Collab, Pycharm
Modules	:	Pandas, torch, torchvision, os, matplotlib, sklearn
Back End	:	Python 3.11.0 or Higher Versions
Front End	:	Streamlit

3.2.5 PACKAGE REQUIREMENTS

Pandas :

Pandas is a powerful Python library designed for data manipulation and analysis. It provides easy-to-use data structures and functions for working with structured data, such as tabular data, time series, and observational data. The primary data structure in Pandas is the DataFrame, which is a two-dimensional, size-mutable, and labeled data structure with columns of potentially different types, similar to a spreadsheet or SQL table. Mostly used functions are `pd.set_option()`, `pd.read_excel()` to read the excel data, `fillna()` to fill the empty or NaN values, `loc[]` for locating a particular row and column.

OS :

The `os` module in Python provides a way of interacting with the operating system, allowing you to perform various operations such as navigating the file system,

executing system commands, and retrieving information about the environment. From this module, `listdir()` to get the filenames in a particular directory.

Matplotlib :

Matplotlib is a Python library used for creating static, interactive, and animated visualizations. It offers a wide range of plotting functions to generate plots, histograms, scatter plots, and more. Matplotlib provides flexibility and customization options to create high-quality graphics for data analysis and presentation purposes. In this module, `subplots()` to visualize multiple images in single figure, `axes.flat` that gives an iterator for all the plots in the same grid, `imshow()` to display images, `axis()` to alter the axes in the plots, `set_title()` to set the title of the plot, `tight_layout()` to give padding in grid of subplots and `show()` to display the plots.

Torchvision :

Torchvision is a package in the PyTorch library specifically designed for computer vision tasks. It provides a collection of utilities, datasets, models, and transforms to facilitate the development of computer vision applications using PyTorch.

`datasets.ImageFolder` is a class provided by the torchvision package in PyTorch, specifically designed for loading image datasets stored in a folder structure where each class has its own subdirectory.

`torchvision.models` module offers pre-trained models for various tasks such as image classification, object detection, and semantic segmentation. These models are built on deep learning architectures like ResNet, EfficientNet, and MobileNet, and can be easily used or fine-tuned for specific tasks.

Imported modules are:

1. `torchvision.models.resnet18()`
2. `torchvision.models.efficientnet_b7()`
3. `torchvision.models.mobilenet_v3_large()`

`torchvision.transforms` module provides a set of common image transformations such as resizing, cropping, rotation, and normalization. These transformations are applied to the input data to augment the training dataset and improve model generalization. Here, `transforms.Compose()`, `transforms.Resize()`, `transforms.Tensors()`.

Torch :

Torch offers utilities for data loading and processing, including datasets and data loaders. It also supports transformations and augmentation techniques for preprocessing input data. Here `DataLoader` is used from `torch.utils.data.DataLoader`.

Torch seamlessly integrates with CUDA and cuDNN libraries, enabling efficient GPU acceleration for tensor operations and model training using `torch.device("cuda" if torch.cuda.is_available() else "cpu")`

Torch includes optimization algorithms like stochastic gradient descent (SGD), Adam, RMSprop, and others, making it easy to train deep learning models with different optimization strategies. Uses `torch.optim` to add optimizers.

Torch provides a rich collection of pre-built modules and layers for constructing various types of neural network architectures, such as fully connected layers, convolutional layers, recurrent layers, and more. Using `torch.nn` to utilize the neural network layers from torch.

Sklearn :

Scikit-learn, commonly abbreviated as sklearn, is a popular open-source machine learning library in Python. It provides simple and efficient tools for data mining and data analysis, built on top of other scientific computing libraries such as NumPy, SciPy, and matplotlib. Here sklearn is majorily used to get the classification report for eeach model that is trained on the dataset. Used funtions are `sklearn.metrics.classification_report`.

CHAPTER-4

DESIGN

CHAPTER-4

DESIGN

4.1 INTRODUCTION

Developing a Digital Pheno parenting: Enhanced Model for Plant Phenotypes requires a methodical strategy that incorporates different phases of development, validation, and implementation. The first step in the procedure is gathering a representative and varied dataset of plant species' leaves that illustrate a range of characteristics. The next step is to apply data preprocessing techniques to improve image clarity and standardize image resolution. One of the most important steps after preprocessing the data is to extract features. Advanced feature extraction techniques like MobileNet, EfficientNet, and ResNet are used for extracting features of leaves.

Following feature extraction, the model architecture is designed cautiously considering all the factors for classification of plant specie and health condition. Thorough training and validation methods are carried out after the model architecture is designed. To reliably evaluate the model's performance, the dataset is usually divided into test, validation, and training sets. The next step is concatenation, where all the models are concatenated for better results. The models include ResNet, EfficientNet, MobileNet. Following validation, if the model performs satisfactorily, rigorous testing is carried out on a separate test set to assess its robustness and capacity for generalization. To evaluate the model's efficiency and reliability the model is evaluated using various performance metrics like accuracy, precision, recall.

Finally, the validated model is deployed for phenotyping tasks, where it analyzes features of plant leaves, predicts the type of plant specie and its health condition, and gives parental suggestions based on the condition.

4.2 UML DIAGRAMS

4.2.1 DATA FLOW DIAGRAM

The flow diagram shows the sequential steps that are engaged in every stage of the process, from gathering data to interpreting the findings. After data collection and preprocessing of images, Advanced feature extraction techniques like MobileNet, EfficientNet, and ResNet are used for extracting features of leaves. Next, the designed model is fed with the features where training and validation procedures are carried out using labelled datasets. Following validation, if the model performs satisfactorily, rigorous testing is carried out on a separate test set to assess its robustness and capacity for generalization. To evaluate the model's efficiency and reliability the model is evaluated using various performance metrics like accuracy, precision, recall. Finally, the validated model is deployed for phenotyping tasks, where it analyzes features of plant leaves, predicts the type of plant specie and its health condition, and gives parental suggestions based on the condition. The model's continued efficacy and applicability in clinical settings are ensured by ongoing monitoring and improvement, which leads to better results. Additionally, feedback loops are incorporated into the flow diagram to guarantee iterative modelmodification based on performance assessments and new research findings. Adaptive changes to maximize the model's effectiveness

are made possible by ongoing monitoring of its sensitivity and accuracy. To reduce potential biases and guarantee the accuracy of results, the flow diagram also includes procedures for data validation and quality assurance. In the end, the flow diagram acts as a guide for the smooth fusion of agriculture and advanced technology.

There are seven main steps in our data flow diagram which are show cased in the below Fig 4.1

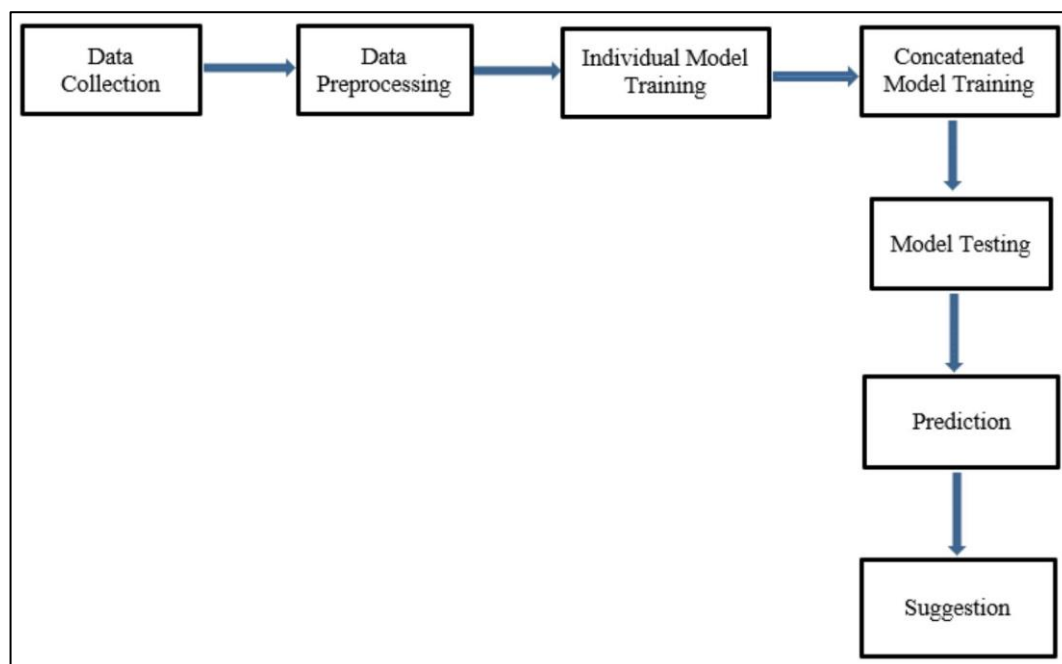


Fig 4.1 Data Flow Diagram

4.2.2 SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram in UML (Unified Modelling Language) that illustrates how objects interact in a particular scenario or over a specific period. The diagram begins with the collection of genetic data from plants, which serves as an input to the deep learning model. These genetic data could include information about the plants leaf images.

The model architecture involves multiple layers of neural networks, depicted in the sequential diagram, representing the process of data transformation and feature extraction. As the genetic data flows through the model, each layer performs specific operations, such as convolutional, pooling and activation functions, to learn intricate patterns and relationships within the data. These learned features are then aggregated and processed further through subsequent layers, enhancing the model's ability to capture complex genetic interactions that influence plant phenotypes.

Once the deep learning model has processed the genetics data, the output predictions or representations of plant phenotypes. This prediction phase is represented in the sequential diagram, illustrating how model transforms genetic information into actionable insights about specific plant traits or responses to environmental factors and as shown in Fig 4.2 based on the predicted insights like specie and disease name suggestions are given for the betterment of the plants.

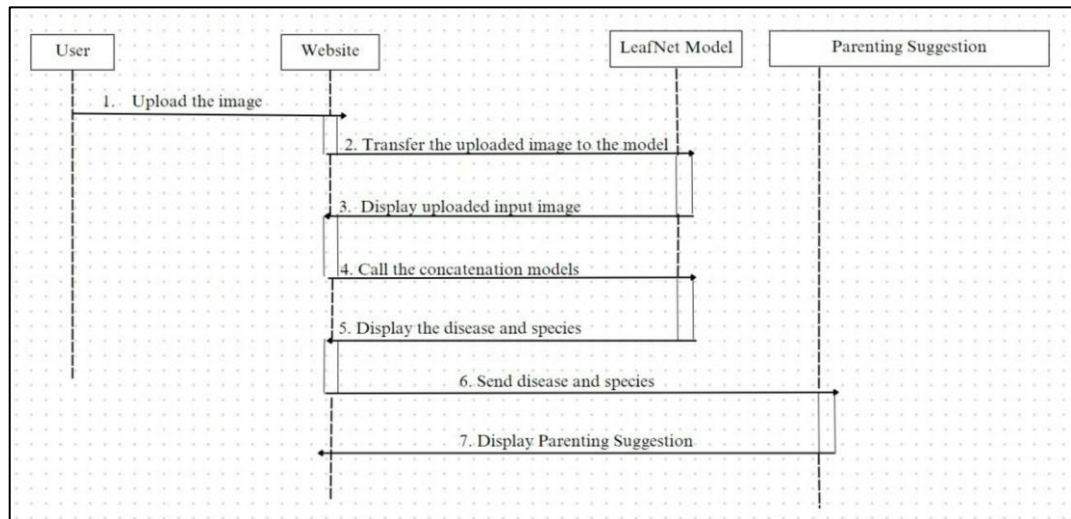


Fig 4.2 Sequence Diagram

4.2.3 USE CASE DIAGRAM

The use case diagram that is depicted in Fig 4.3 for Leaf Disease Prediction and Parenting Suggestions encapsulates a seamless process for users to interact with the system. It begins with users uploading images of leaves, which could exhibit symptoms of disease or be healthy for comparison. The system then analyzes these images, identifying any diseases present on the leaves and providing informative outcomes to the user. These outcomes may include the specific disease affecting the leaf and its severity, enabling users to take appropriate actions. Additionally, the system offers parenting suggestions tailored to the identified disease and plant species, facilitating effective plant care. This user-centric approach ensures that users can effortlessly engage with the system and derive meaningful insights to maintain the health of their plants. The use case diagram of this work with respect to user is shown in below.

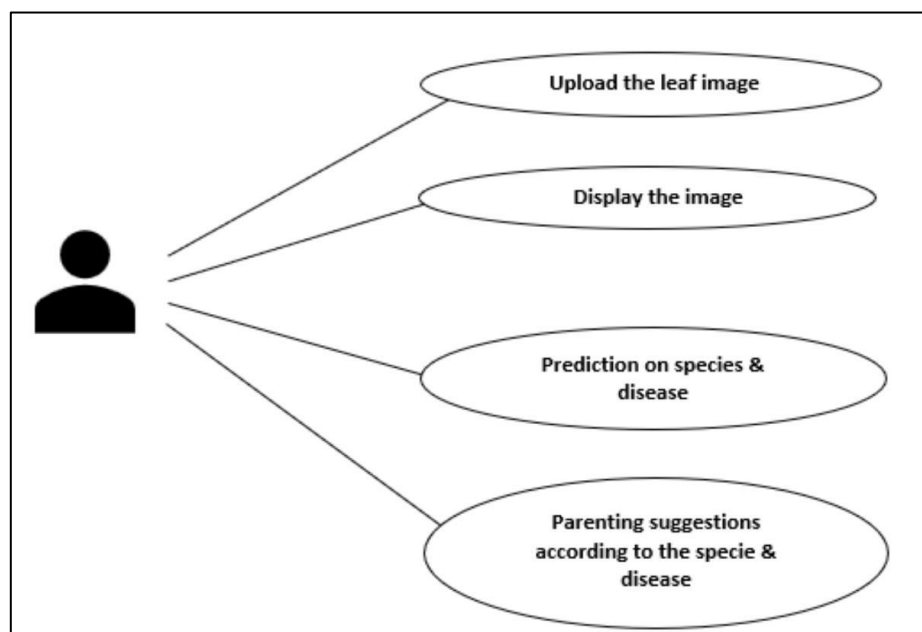


Fig 4.3 Use Case Diagram

4.3 MODULE DESIGN AND ORGANISATION

The entire module is constructed in 8 modules:

1. Data Collection

Collecting data for plant leaf detection involves sourcing images of plant leaves along with relevant metadata. This data can be obtained from a variety of sources such as botanical gardens, agricultural research institutions, or online databases dedicated to plant species.

2. ResNet Fine – Tuning

ResNet-18 model loaded with default weights. Replace last fully connected layer for output classes. Fine-tune specific layers, using Adam optimizer, step-wise scheduler, cross-entropy loss. Transferred to GPU for computation.

3. EfficientNet Fine – Tuning

An EfficientNet model pretrained on ImageNet is customized for a specific classification task. Selected layers are fine-tuned while others are frozen. Optimizer, scheduler, and loss criterion are configured for training.

4. MobileNet Fine – Tuning

A pre-trained MobileNetV3-Large model is prepared by freezing its layers, except for the classifier. Custom fully connected layers are added for the task, then fine-tuned and optimized for training.

5. LeafNet Model

Fine-trained ResNet, EfficientNet, and MobileNet models have their classification heads removed. Features are concatenated, passed through a fully connected layer, and trained with optimizer and scheduler.

6. Testing Model

After Training all the models, testing each model with the test directory from the dataset is important so that we can understand how well the models are performing on new data.

7. Parenting Modules

Trained LeafNet Model is used to predict plant species and disease. Based on the predictions, a set of actions are given as parenting suggestions.

8. User Interface

The LeafNet model is deployed on a backend server and presented with a user-friendly interface through the Streamlit framework, enabling users to obtain plant insights via leaf images.

4.4 SUMMARY

The design chapter outlines the architectural blueprint of the AI project, encompassing the identified use cases, sequence diagrams illustrating the interaction between system components, and data flow diagrams delineating the movement of

information within the system. This holistic representation elucidates the project's structure and functionality, facilitating a comprehensive understanding of its design principles and implementation strategies. By integrating use cases, sequence diagrams, and data flow diagrams, the design chapter provides a succinct overview of the project's architecture, ensuring clarity and coherence in the development process.

CHAPTER-5

IMPLEMENTATION AND

RESULTS

CHAPTER-5

IMPLEMENTATION AND RESULTS

5.1 INTRODUCTION

In an age marked by remarkable advancements in Artificial Intelligence technology, the field of plant Phen parenting is undergoing a transformative evolution. This project endeavors to develop a sophisticated concatenated neural network tailored for the identification of plant species and the detection of leaf diseases, coupled with intelligent suggestions based on predicted conditions. By harnessing the power of deep learning algorithms, this initiative seeks to redefine the landscape of plant health monitoring and management.

Utilizing a diverse array of datasets comprising images of plant leaves and associated metadata, the proposed system aims to accurately classify plant species and detect various leaf diseases. Through meticulous training and optimization of the neural network, the system will be equipped to distinguish between healthy foliage and manifestations of diseases such as fungal infections, bacterial infestations, and nutrient deficiencies. This precision in diagnosis holds the promise of facilitating timely interventions to mitigate the impact of plant diseases and optimize crop yield.

End-users of the system will benefit from an intuitive interface designed to facilitate seamless interaction with the plant species and leaf disease detection functionalities. Leveraging data extracted from leaf images, including features based on features such as color, texture, and shape, the deep learning algorithms will undergo rigorous training to ensure robust classification and disease detection capabilities. Moreover, the system's ability to provide tailored suggestions for disease management and plant care based on predicted conditions.

The implementation of a web-based platform using frameworks such as Streamlit will enable effortless communication between plant enthusiasts, agricultural professionals, and researchers. This platform will empower users to access comprehensive insights into plant health, enabling informed decision-making and proactive management strategies. Additionally, stakeholders in the agriculture sector will leverage the platform to streamline the process of diagnosing and addressing leaf diseases, fostering productivity and sustainability in crop cultivation practices.

The outcomes of this project hold significant promise for revolutionizing plant health monitoring and management practices. By offering more accurate predictions of plant species and leaf disease conditions, the system empowers users to make informed decisions regarding crop management and disease mitigation strategies. Furthermore, the integration of classification and recommendation techniques enhances the system's efficiency and accessibility, paving the way for more effective plant care practices and improved agricultural outcomes.

In summary, the proposed concatenated neural network for plant species and leaf disease detection represents a paradigm shift in the field of agriculture. By providing a comprehensive, efficient, and user-friendly solution, the system offers a transformative approach to plant health monitoring and management, ultimately contributing to the advancement of sustainable agriculture practices and global food security efforts.

5.2 IMPLEMENTATION OF KEY FUNCTIONS

- **Data Collection :**

The first step in collecting data for plant leaf detection involves sourcing images of plant leaves along with relevant metadata. This data can be obtained from a variety of sources such as botanical gardens, agricultural research institutions, or online databases dedicated to plant species. Each image is annotated with metadata providing information about the plant species and any visible signs of disease on the leaves.

- **ResNet Fine-tuning :**

A ResNet-18 model is loaded with weights initialized to the default values. The last fully connected layer is replaced with a new linear layer to match the number of output classes. Specific layers, including the final convolutional and batch normalization layers of the fourth residual block, and the new linear layer, are identified for fine-tuning. Only these identified layers are set to be trainable while all other layers are frozen. The model is then transferred to the designated device (e.g., GPU) for computation. An Adam optimizer with a learning rate of 0.01, a step-wise learning rate scheduler, and a cross-entropy loss function are initialized for training.

- **EfficientNet Fine-tuning :**

Initializing an EfficientNet model pretrained on ImageNet and replaces its final fully connected layer with a new one suited for the specific classification task (n_class). Certain layers are identified for fine-tuning, allowing their weights to be updated during training while freezing others to retain their pretrained weights. The model is then transferred to the specified device (GPU or CPU), and an optimizer and scheduler are set up for training. Finally, a cross-entropy loss criterion is defined for model evaluation.

- **MobileNet Fine-tuning :**

A pre-trained MobileNetV3-Large model is loaded and its layers are frozen to prevent weight updates during training, except for the classifier layers. The classifier layers are replaced with custom fully connected layers tailored to the specific classification task. These layers are then unfrozen to allow fine-tuning on the new dataset. Finally, the model is moved to the appropriate device, and an optimizer and scheduler are set up for training. This process enables efficient transfer learning by leveraging the pre-trained model's feature extraction capabilities while adapting it to the new classification task.

- **LeafNet Model :**

All the three pre-trained models (ResNet, EfficientNet, and MobileNet) are loaded and their classification heads are removed. These models are then integrated into a custom LeafNet architecture, where their extracted features are concatenated and passed through a fully connected layer for classification. The LeafNet model is initialized with the desired number of output classes. Additionally, the parameters of the ResNet and EfficientNet models are frozen to prevent them from being updated during training, while the parameters of the fully connected layer are set to trainable. Finally, the model is prepared for training with an optimizer, loss function, and learning rate scheduler.

- **Testing Module :**

After Training all the models, testing each model with the test directory from the dataset is important so that we can understand how well the models are performing on new data.

- **Parenting Suggestions Module :**

Trained LeafNet Model is used to predict plant species and disease. Based on the predictions, a set of actions are given as parenting suggestions.

- **Model Deployment & User Interface :**

Finally obtained LeafNet Model is deployed using a Backend server and given an user-friendly interface using streamlit framework . This allows the user to seamlessly visit the web application and get the insights on their plants by giving the images of the plant leaf as input for the deployed model.

5.3 METHOD OF IMPLEMENTATION

LeafNet.ipynb

```
from google.colab import drive
drive.mount('/content/drive')
data_dir = "/content/drive/MyDrive/Project/New Plant Diseases
Dataset(Augmented)"
train_dir = data_dir + "/train"
valid_dir = data_dir + "/valid"
import torch
import torchvision
from torchvision import transforms, datasets
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import cv2
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
```

```

train_data = datasets.ImageFolder(train_dir, transform=train_transform)
test_data = datasets.ImageFolder(valid_dir, transform=test_transform) #transform
removed

```

```

train_loader = DataLoader(train_data, batch_size=350,
shuffle=True,num_workers=4) #1024
test_loader = DataLoader(test_data, batch_size=350,
shuffle=False,num_workers=4)
train_data.class name
n_class = len(train_data.classes)
n_class
# Visualize a few sample images
fig, axes = plt.subplots(3, 3, figsize=(10, 10))
for i, ax in enumerate(axes.flat):
    image, label = train_data[i] # Get image and label
    ax.imshow(image.permute(1, 2, 0)) # Matplotlib expects channels last
    ax.axis('off')
    ax.set_title(f'Class: {train_data.classes[label]}")
plt.tight_layout()
plt.show()
# Model Validation
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from sklearn.metrics import classification_report

```

```

classes = train_data.classes
# print(n_class,len(classes))
# Function to evaluate the model on validation data
def evaluate_model(model, dataloader):
    model.eval()
    correct = 0
    total = 0
    predictions = []
    ground_truths = []

    with torch.no_grad():

```

```

for images, labels in dataloader:
    images, labels = images.to(device), labels.to(device)
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
    predictions.extend(predicted.cpu().numpy())
    ground_truths.extend(labels.cpu().numpy())
accuracy = 100 * correct / total
print(f'Accuracy on validation set: {accuracy:.2f}%')
# Generate classification report
print("Classification Report:")
print(classification_report(ground_truths, predictions, target_names=classes))
# Model Testing
import os

test_dir = "/content/drive/MyDrive/Project/New Plant Diseases
Dataset(Augmented)"
test = datasets.ImageFolder(test_dir, transform=test_transform)

test_images = sorted(os.listdir(test_dir + '/test'))
test_images
def predict_image(img, model):
    img =img.unsqueeze(0)
    image = img.to(device)
    output = model(image)
    _, preds = torch.max(output, 1)
    return train_data.classes[preds[0].item()]

# ResNet
resnet = torchvision.models.resnet18(weights='DEFAULT')

num_fts = resnet.fc.in_features
resnet.fc = nn.Linear(num_fts, n_class)

layers_to_unfreeze = ['layer4.1.conv2.weight', 'layer4.1.bn2.weight',
'layer4.1.bn2.bias', 'fc.weight', 'fc.bias']

```

```

# Unfreeze the identified layers
for name, param in resnet.named_parameters():
    if any(layer_name in name for layer_name in layers_to_unfreeze):
        param.requires_grad = True
    else:
        param.requires_grad = False
resnet.to(device)

optimizer = optim.Adam(resnet.parameters(), lr=0.01)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
criterion = nn.CrossEntropyLoss()
num_epochs = 10
best_val_loss = float('inf')
patience = 5
for epoch in range(num_epochs):
    resnet.train()
    running_loss = 0.0

    # Training loop
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = resnet(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)
    epoch_loss = running_loss / len(train_loader.dataset)

    # Validation loop
    resnet.eval()
    with torch.no_grad():
        running_val_loss = 0.0
        for val_images, val_labels in test_loader:
            val_images, val_labels = val_images.to(device), val_labels.to(device)
            val_outputs = resnet(val_images)
            val_loss = criterion(val_outputs, val_labels)
            running_val_loss += val_loss.item() * val_images.size(0)

```



```

val_epoch_loss = running_val_loss / len(test_loader.dataset)
if val_epoch_loss < best_val_loss:
    best_val_loss = val_epoch_loss
    patience = 5
else:
    patience -= 1
    if patience == 0:
        print("Early stopping triggered!")
        break
    # Step the scheduler
    scheduler.step()
    print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {epoch_loss:.4f},
Validation Loss: {val_epoch_loss:.4f}')
    evaluate_model(resnet, test_loader)
    torch.save(resnet, 'resnet.pth')
    path = "/kaggle/input/pdc/pytorch/individual_model/2/resnet.pth"

resnet = torch.load(path)
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, resnet))

# EfficientNet
efficientnet = torchvision.models.efficientnet_b7(weights='DEFAULT')

num_fts = efficientnet.classifier[-1].in_features
efficientnet.classifier[-1] = nn.Linear(num_fts, n_class)

layers_to_unfreeze = ['features.8.1.weight', 'features.8.1.bias', 'classifier.1.weight',
'classifier.1.bias']

# Unfreeze the identified layers
for name, param in efficientnet.named_parameters():
    if any(layer_name in name for layer_name in layers_to_unfreeze):
        param.requires_grad = True
    else:
        param.requires_grad = False
efficientnet.to(device)

```

```

optimizer = optim.Adam(efficientnet.parameters(), lr=0.001)
# scheduler = optim.lr_scheduler.CyclicLR(optimizer, base_lr=0.001, max_lr=0.1,
step_size_up=100, mode='triangular')
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
criterion = nn.CrossEntropyLoss()
#### EfficientNet Training
num_epochs = 10
best_val_loss = float('inf')
patience = 5

for epoch in range(num_epochs):
    efficientnet.train()
    running_loss = 0.0
    # Training loop
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = efficientnet(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)

    # Validation loop
    efficientnet.eval()
    with torch.no_grad():
        running_val_loss = 0.0
        for val_images, val_labels in test_loader:
            val_images, val_labels = val_images.to(device), val_labels.to(device)

            val_outputs = efficientnet(val_images)
            val_loss = criterion(val_outputs, val_labels)
            running_val_loss += val_loss.item() * val_images.size(0)

        val_epoch_loss = running_val_loss / len(test_loader.dataset)
        if val_epoch_loss < best_val_loss:

```

```

        best_val_loss = val_epoch_loss
        patience = 5
    else:
        patience -= 1
        if patience == 0:
            print("Early stopping triggered!")
            break

    scheduler.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {epoch_loss:.4f},
Validation Loss: {val_epoch_loss:.4f}')

    evaluate_model(efficientnet, test_loader)
    torch.save(efficientnet, 'efficientnet.pth')
    path = "/kaggle/input/pdc/pytorch/individual_model/2/efficientnet.pth"

    efficientnet = torch.load(path)
    # getting all predictions (actual label vs predicted)
    for i, (img, label) in enumerate(test):
        print('Label:', test_images[i], ', Predicted:', predict_image(img, efficientnet))

# MobileNet
from torchvision.models import mobilenet_v3_large

# Load the pre-trained MobileNetV3-Large model
mobilenet = mobilenet_v3_large(weights='DEFAULT')

# Freeze all the layers
for param in mobilenet.parameters():
    param.requires_grad = False

# Unfreeze the last few layers
unfreeze_layers = ['classifier']

for name, param in mobilenet.named_parameters():
    for layer in unfreeze_layers:
        if layer in name:

```

```

        param.requires_grad = True

# num_classes = n_class # Change this to your desired number of classes
# print("class:", num_classes)
mobilenet.classifier = nn.Sequential(
    nn.Linear(960, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, n_class)
)
from torch.optim.lr_scheduler import ReduceLROnPlateau

mobilenet.to(device)
optimizer = optim.Adam(mobilenet.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=3, factor=0.1,
verbose=True)
#### MobileNet Training
num_epochs = 10
best_val_loss = float('inf')
patience = 5

for epoch in range(num_epochs):
    mobilenet.train()
    running_loss = 0.0

    # Training loop
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = mobilenet(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)

```

```

# Validation loop
mobilenet.eval()
with torch.no_grad():
    running_val_loss = 0.0
    for val_images, val_labels in test_loader:
        val_images, val_labels = val_images.to(device), val_labels.to(device)

        val_outputs = mobilenet(val_images)
        val_loss = criterion(val_outputs, val_labels)
        running_val_loss += val_loss.item() * val_images.size(0)

    val_epoch_loss = running_val_loss / len(test_loader.dataset)

    if val_epoch_loss < best_val_loss:
        best_val_loss = val_epoch_loss
        patience = 5
    else:
        patience -= 1
        if patience == 0:
            print("Early stopping triggered!")
            break

# Step the scheduler
scheduler.step(val_epoch_loss)

print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {epoch_loss:.4f},
Validation Loss: {val_epoch_loss:.4f}')

evaluate_model(mobilenet, test_loader)
torch.save(mobilenet, 'mobilenet.pth')
path = "/kaggle/input/pdc/pytorch/individual_model/2/mobilenet.pth"

mobilenet = torch.load(path)
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, mobilenet))

```

```

# LeafNet
path1 = "/kaggle/input/pdc/pytorch/individual_model/2/resnet.pth" #Resnet
path2 = "/kaggle/input/pdc/pytorch/individual_model/2/efficientnet.pth"
#EfficientNet
path3 = "/kaggle/input/pdc/pytorch/individual_model/2/mobilenet.pth"
#MobileNet

resnet = torch.load(path1)
efficientnet = torch.load(path2)
mobilenet = torch.load(path3)
print(f'Resnet : {resnet.fc.in_features}')
print(f'EfficientNet : {efficientnet.classifier[-1].in_features}')
print(f'MobileNet : {mobilenet.classifier[-4].in_features}')
num_fts_resnet = resnet.fc.in_features
resnet.fc = nn.Identity() # Remove classification head

num_fts_efficientnet = efficientnet.classifier[-1].in_features
efficientnet.classifier = nn.Identity() # Remove classification head

num_fts_mobilenet = mobilenet.classifier[-4].in_features
mobilenet.classifier = nn.Identity() # Remove classification head

print(num_fts_resnet+num_fts_efficientnet+num_fts_mobilenet)
class LeafNet(nn.Module):
    def __init__(self, num_classes):
        super(LeafNet, self).__init__()
        self.resnet = resnet
        self.efficientnet = efficientnet
        self.mobilenet = mobilenet
        self.fc = nn.Linear(num_fts_resnet + num_fts_efficientnet +
num_fts_mobilenet, num_classes)

    def forward(self, x):
        x1 = self.resnet(x)
        x2 = self.efficientnet(x)
        x3 = self.mobilenet(x)
        x = torch.cat((x1.view(x1.size(0), -1), x2.view(x2.size(0), -1),
x3.view(x3.size(0), -1)), dim=1)

```

```

        x = self.fc(x)
        return x

leafnet = LeafNet(n_class)

for param in leafnet.resnet.parameters():
    param.requires_grad = False
for param in leafnet.efficientnet.parameters():
    param.requires_grad = False
for param in leafnet.mobilenet.parameters():
    param.requires_grad = False
for param in leafnet.fc.parameters():
    param.requires_grad = True

print(leafnet.fc)
leafnet.to(device)

from torch.optim.lr_scheduler import ReduceLROnPlateau

optimizer = optim.Adam(leafnet.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=3, factor=0.1,
verbose=True)
### LeafNet Training
num_epochs = 10
best_val_loss = float('inf')
patience = 5

for epoch in range(num_epochs):
    leafnet.train()
    running_loss = 0.0

    # Training loop
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = leafnet(images)
        loss = criterion(outputs, labels)

```

```

        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)

epoch_loss = running_loss / len(train_loader.dataset)

# Validation loop
leafnet.eval()
with torch.no_grad():
    running_val_loss = 0.0
    for val_images, val_labels in test_loader:
        val_images, val_labels = val_images.to(device), val_labels.to(device)

        val_outputs = leafnet(val_images)
        val_loss = criterion(val_outputs, val_labels)
        running_val_loss += val_loss.item() * val_images.size(0)

val_epoch_loss = running_val_loss / len(test_loader.dataset)

if val_epoch_loss < best_val_loss:
    best_val_loss = val_epoch_loss
    patience = 5
else:
    patience -= 1
    if patience == 0:
        print("Early stopping triggered!")

        break
scheduler.step(val_epoch_loss)

print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss: {epoch_loss:.4f},
Validation Loss: {val_epoch_loss:.4f}")

torch.save(leafnet, 'LeafNet.pth')
path = "/kaggle/working/LeafNet.pth"

leafnet = torch.load(path)

```



```

evaluate_model(leafnet, test_loader)
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, leafnet))
# Parenting Suggestion
class LeafNet(nn.Module):
    def __init__(self, num_classes):
        super(LeafNet, self).__init__()
        self.resnet = resnet
        self.efficientnet = efficientnet
        self.mobilenet = mobilenet
        self.fc = nn.Linear(num_fts_resnet + num_fts_efficientnet +
num_fts_mobilenet, num_classes)

    def forward(self, x):
        x1 = self.resnet(x)
        x2 = self.efficientnet(x)
        x3 = self.mobilenet(x)
        x = torch.cat((x1.view(x1.size(0), -1), x2.view(x2.size(0), -1),
x3.view(x3.size(0), -1)), dim=1)
        x = self.fc(x)
        return x
path = "/content/drive/MyDrive/Project/LeafNet.pth"

leafnet = torch.load(path, map_location=torch.device('cpu'))
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, leafnet))
    if i==31:
        break
import pandas as pd

pd.set_option('display.max_colwidth', None)

suggestionsDF =
pd.read_excel("/content/drive/MyDrive/Project/Suggestions.xlsx",
engine='openpyxl')
suggestionsDF.fillna("", inplace=True)

```

```

suggestionsDF
index = 8
img, label = test[index]
plt.imshow(img.permute(1, 2, 0))
print('Label:', test_images[index])
predicted = predict_image(img, leafnet)
specie, disease = predicted.split("___")[0].strip(), "
".join(predicted.split("___")[1].split("_")).strip()
print('Predicted Specie: ',specie)
print(f'Predicted Disease: {disease}')

sg = suggestionsDF.loc[(suggestionsDF['specie']==specie) &
(suggestionsDF['disease']==disease)][["suggestion"]]
print("Suggestion: ")
sg.to_string(index=False)

```

app.py

```

import streamlit as st
import torch
from PIL import Image
import torchvision
from torchvision import transforms, datasets
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import cv2
import pandas as pd

suggestionsDF = pd.read_excel("Suggestions.xlsx", engine='openpyxl')
suggestionsDF.fillna("", inplace=True)

pd.set_option('display.max_colwidth', None)

class LeafNet(nn.Module):
    def __init__(self, num_classes):
        super(LeafNet, self).__init__()
        self.resnet = resnet
        self.efficientnet = efficientnet
        self.mobilenet = mobilenet
        self.fc = nn.Linear(num_fts_resnet + num_fts_efficientnet +
num_fts_mobilenet, num_classes)

```

```

def forward(self, x):
    x1 = self.resnet(x)
    x2 = self.efficientnet(x)
    x3 = self.mobilenet(x)
    x = torch.cat((x1.view(x1.size(0), -1), x2.view(x2.size(0), -1),
x3.view(x3.size(0), -1)), dim=1)
    x = self.fc(x)
    return x

leafnet = torch.load("LeafNet.pth", map_location=torch.device("cpu"))

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

classes = ['Apple___Apple_scab',
'Apple___Black_rot',
'Apple___Cedar_apple_rust',
'Apple___healthy',
'Blueberry___healthy',
'Cherry_(including_sour)___Powdery_mildew',
'Cherry_(including_sour)___healthy',
'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
'Corn_(maize)___Common_rust_',
'Corn_(maize)___Northern_Leaf_Blight',
'Corn_(maize)___healthy',
'Grape___Black_rot',
'Grape___Esca_(Black_Measles)',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
'Grape___healthy',
'Orange___Haunglongbing_(Citrus_greening)',
'Peach___Bacterial_spot',
'Peach___healthy',
'Pepper_bell___Bacterial_spot',
'Pepper_bell___healthy',
'Potato___Early_blight',
'Potato___Late_blight',
'Potato___healthy',
'Raspberry___healthy',
'Soybean___healthy',
'Squash___Powdery_mildew',
'Strawberry___Leaf_scorch',
'Strawberry___healthy',
'Tomato___Bacterial_spot',
'Tomato___Early_blight',
'Tomato___Late_blight',
'Tomato___Leaf_Mold',

```

```

'Tomato___Septoria_leaf_spot',
'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Target_Spot',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Tomato_mosaic_virus',
'Tomato___healthy']

# def suggestion()

def predict_image(img):
    img = transform(img).unsqueeze(0)
    # image = img.to(device)
    output = leafnet(img)
    _, preds = torch.max(output, 1)
    return classes[preds[0].item()]

# Streamlit app
def main():
    st.title("Digital Pheno-Parenting: Enhanced model for Plant-Phenotypes using Deep Learning")

    # Upload image
    st.header("Upload Image")

    uploaded_file = st.file_uploader("Choose an image...", type=["jpg"])

    if uploaded_file is not None:
        # Display uploaded image
        # print("1")
        image = Image.open(uploaded_file)
        # print("2")
        st.image(image.resize((200, 200)), caption="Uploaded Image")
        if st.button("Give Suggestion"):
            predicted = predict_image(image)
            # Display prediction
            st.header("Output")
            # st.write(predicted)

            specie, disease = predicted.split("___")[0].strip(), "
".join(predicted.split("___")[1].split("_")).strip()
            # print('Predicted Specie: ',specie)
            # print(f'Predicted Disease: {disease}')
            st.write(f'Predicted Specie is {specie}')
            # st.write(specie)
            if disease=="healthy":
                st.write("Given plant is healthy.")
            return
            st.write(f'Predicted Disease is {disease}')

```

```

        # st.write(disease)
        sg = suggestionsDF.loc[(suggestionsDF['specie']==specie) &
(suggestionsDF['disease']==disease)][["suggestion"]]
        # print("Suggestion: ")
        st.write("Parenting Suggestion : ")
        st.write(sg.to_string(index=False))
    else:
        st.write("Please uplaod an Image")

if __name__ == "__main__":
    main()

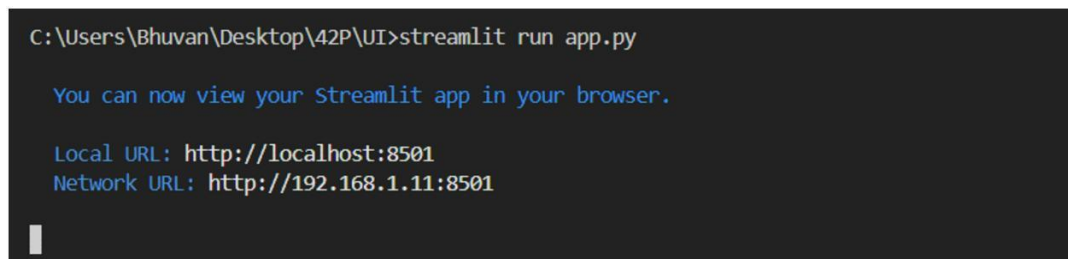
```

5.4 OUTPUT SCREENS AND RESULT ANALYSIS

To run the app in local system, first open Command prompt and navigate to project folder.

Run the app using below command as shown in Fig 5.1

streamlit run app.py



```

C:\Users\Bhuvan\Desktop\42P\UI>streamlit run app.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.11:8501

```

Fig 5.1 Starting streamlit server

Streamlit server will now be up and running on localhost in the port 8501.

Entering the above shown url `http://localhost:8501` in the browser URL bar will give you access to the web application as displayed in below Fig 5.2.

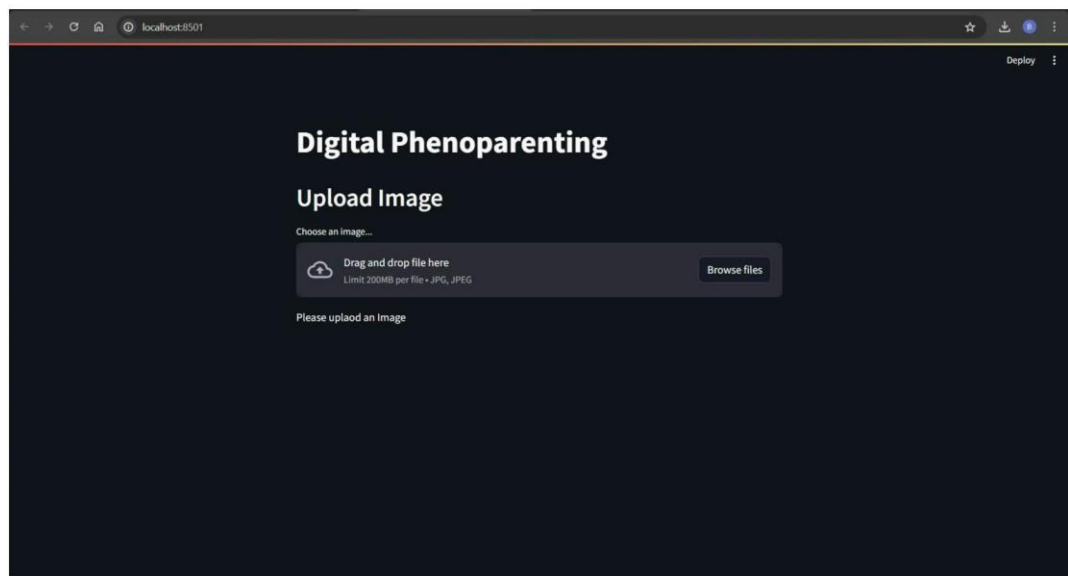


Fig 5.2 User Interface

click on 'Browse Files' button that will open an File explorer tab as shown in Fig 5.3 to upload a image as an input for the model.

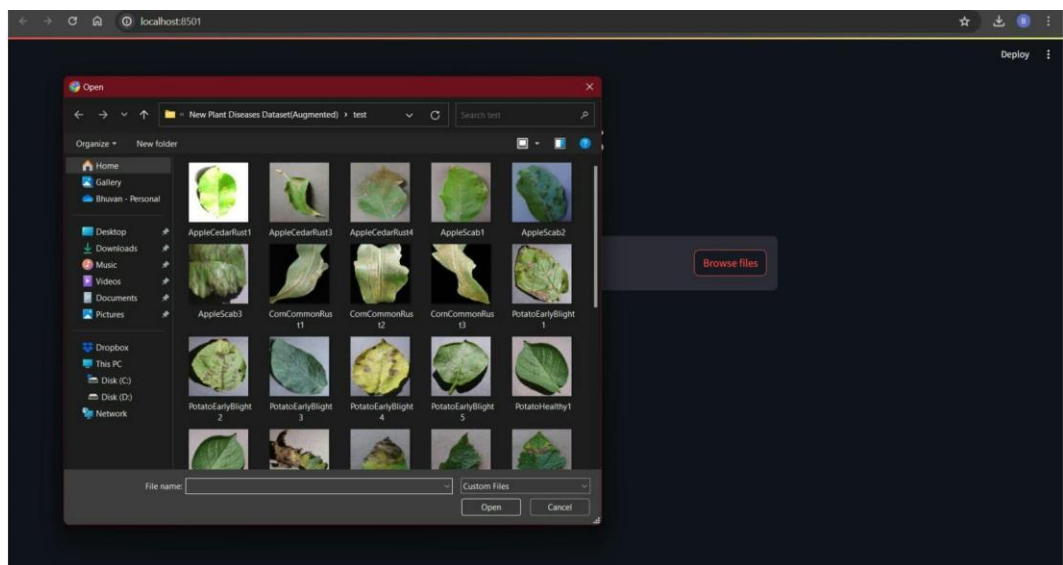


Fig 5.3 Uploading an Leaf Image

Select an image from the files and click on 'open'.

Now as in Fig 5.4, the input image will be loaded and displayed in the UI

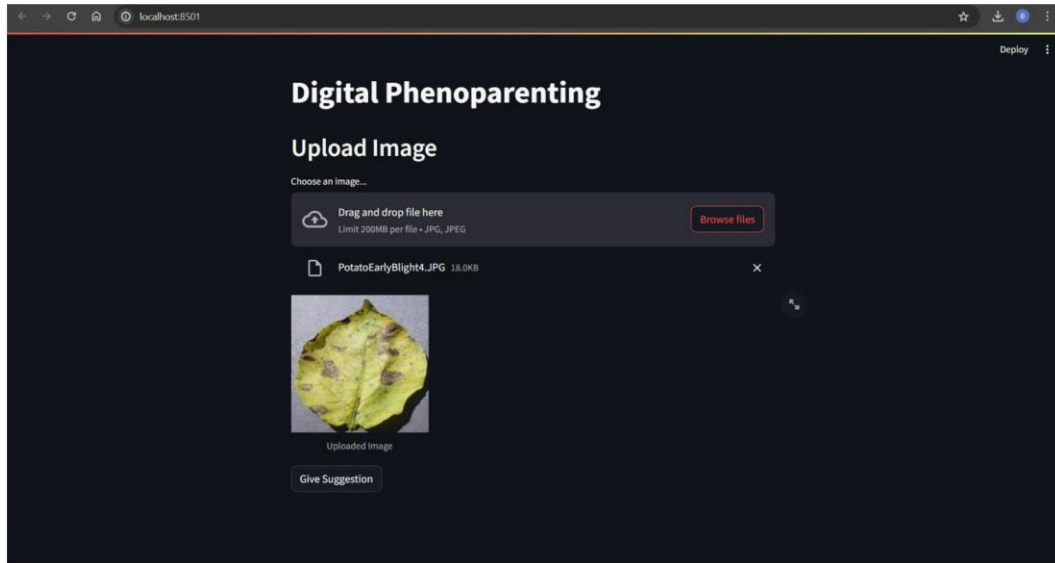


Fig 5.4 Displaying the uploaded image.

To get the parenting suggestions, you need to click on 'Give Suggestion' button.

And then you will be prompted with the specie and health condition of the plant as shown in the Fig 5.5. If the plant is unhealthy, disease name will be given with some future course of action as parenting suggestions.

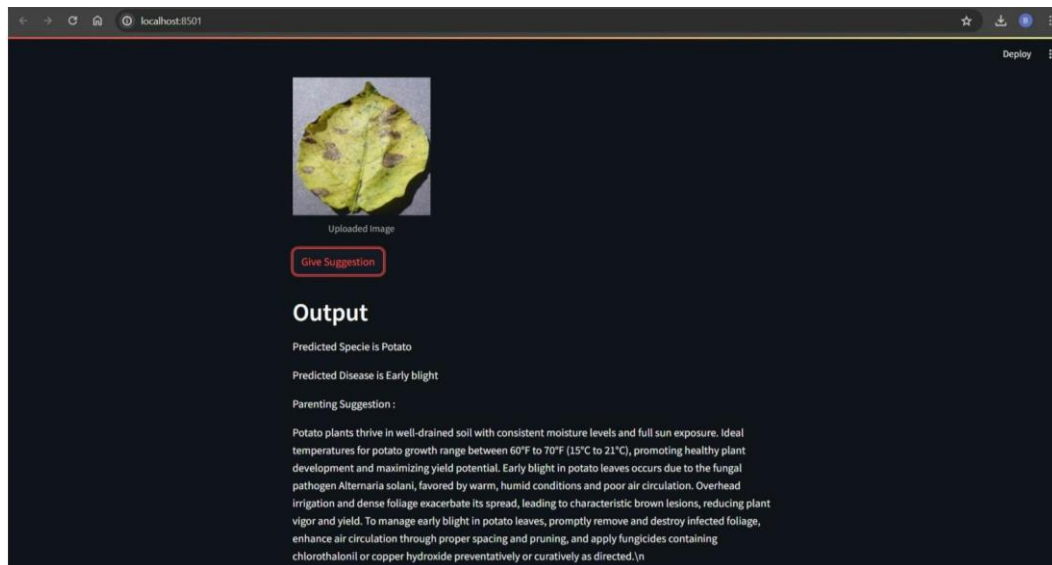


Fig 5.5 Displaying the specie, disease and suggestion

Local streamlit server can be stopped by pressing Ctrl+C in the command prompt where the server is started.

5.5 SUMMARY

The process of collecting data for plant leaf detection begins with sourcing images of plant leaves and associated metadata from diverse sources such as botanical gardens and agricultural research institutions. Annotations including plant species and disease indications accompany each image. Subsequently, pre-trained models like ResNet-18, EfficientNet, and MobileNetV3-Large are fine-tuned for the specific task of plant species and leaf disease classification. This involves adjusting the models' architecture, replacing classification layers, and selectively unfreezing certain layers for training while keeping others frozen to retain learned features. The models are then integrated into a custom LeafNet architecture, where their features are concatenated and passed through a fully connected layer for classification. The LeafNet model undergoes further training with an optimized Adam optimizer, cross-entropy loss criterion, and learning rate scheduler. Upon training completion, the model predicts plant species and diseases, providing actionable parenting suggestions based on the predictions. Finally, the trained LeafNet model is deployed using a backend server and presented with a user-friendly interface developed with the Streamlit framework, enabling users to interactively input plant leaf images and receive insightful analyses on their plants.

CHAPTER-6

TESTING AND VALIDATION

CHAPTER-6

TESTING AND VALIDATION

6.1 INTRODUCTION

6.1.1 INTRODUCTION TO TESTING

Testing and validation play pivotal roles in the field of artificial intelligence, particularly in the realm of deep learning where complex models are employed. Testing involves assessing the performance and functionality of AI models to ensure they meet the desired objectives and standards. This process encompasses various techniques such as unit testing, integration testing, and system testing, which validate different aspects of the model's behavior and functionality.

Validation, on the other hand, focuses on evaluating the generalization ability and robustness of AI models. This involves splitting the dataset into training and validation sets, where the training set is used to optimize the model's parameters, while the validation set is used to assess its performance on unseen data. Cross-validation techniques like k-fold cross-validation further enhance model validation by partitioning the dataset into multiple subsets for training and validation iteratively.

In the context of deep learning models, testing and validation are crucial steps to ensure the reliability, accuracy, and generalization capability of the models. These processes help identify potential issues such as overfitting, underfitting, and data leakage, allowing researchers and practitioners to refine and improve their models iteratively. Additionally, testing and validation enable the comparison of different model architectures, hyperparameters, and optimization techniques, leading to the development of more effective and efficient AI solutions. Overall, robust testing and validation procedures are essential for building trustworthy and impactful AI systems across various domains.

6.2 DESIGN OF TEST CASES AND SCENARIOS

While validating the model, majorly there will be two cases for the output. One is that the plant in the input image can be healthy without any health problems. The second situation is that the plant in the input image can be unhealthy. In such scenarios the along with the plant specie name, name of the disease that infected the plant will also be given. Along with the set of actions that can be helpful to take further care for the plants. These set of actions are similar to parenting suggestions.

The considered dataset consists of 12 different plant species associated with different diseases that finally derives into 38 classes. There are around 32 images solely separated for the testing purpose from dataset other than that are considered for training and validation purposes.

There are mainly two categories in the output from the developed system which are either plant can be healthy or unhealthy. Firstly the specie name is given and then if the plant is healthy nothing other than specie name and its health condition as healthy is displayed as shown in the Fig 6.1 where as if the plant is

unhealthy, along with the species name, a set of suggestions for the betterment of plants will be displayed as shown in the Fig 6.2.

Case 1: If the plant is healthy.

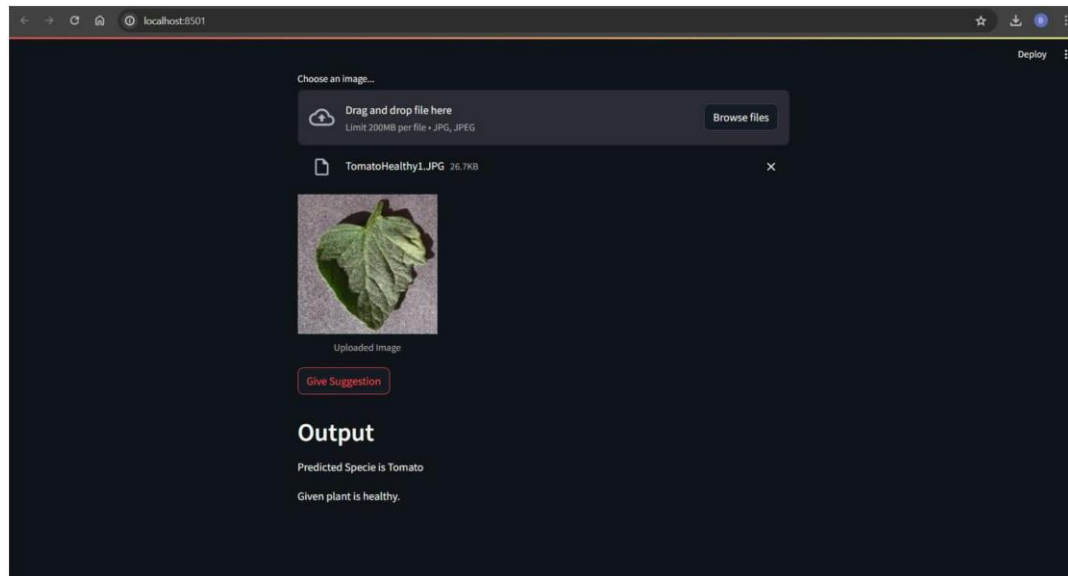


Fig 6.1 Healthy plant

Case 2 : If the plant is unhealthy.

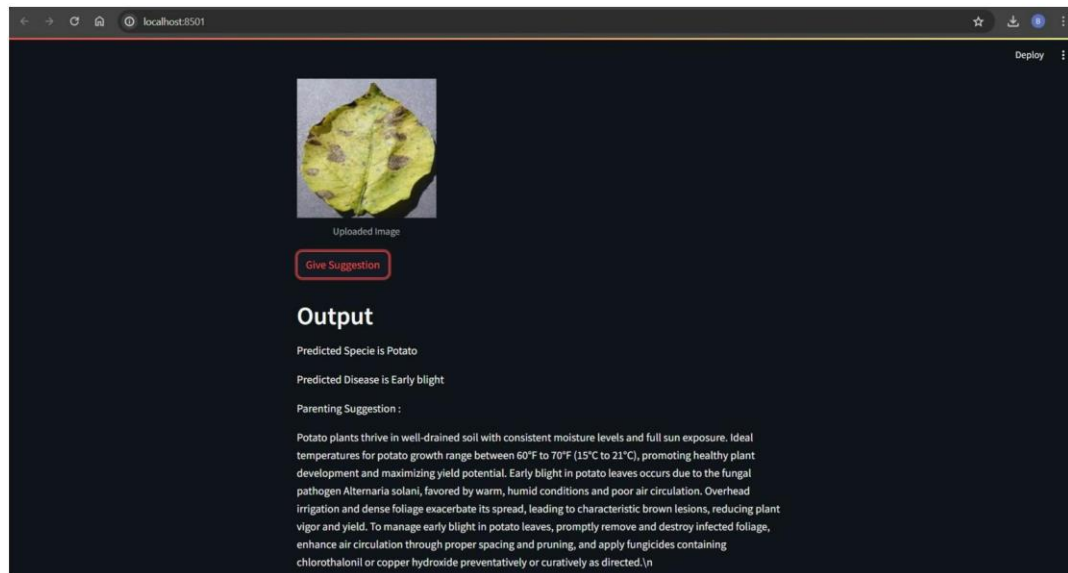


Fig 6.2 Unhealthy plant with suggestion

6.3 VALIDATION TESTING

Validation accuracy is a critical metric in the development of deep learning models as it serves as a key indicator of the model's performance and generalization ability. Unlike training accuracy, which measures how well a model fits the training data, validation accuracy assesses how effectively the model can make predictions on unseen data.

A high validation accuracy indicates that the model has learned meaningful patterns from the training data and can accurately classify or predict new instances. It signifies the model's ability to generalize well to unseen data, which is essential for real-world applications.

Validating a model against a separate validation set helps identify issues such as overfitting, where the model memorizes the training data without learning useful patterns, or underfitting, where the model fails to capture the underlying patterns in the data. By monitoring and optimizing validation accuracy during model development, researchers and practitioners can ensure the reliability and effectiveness of their deep learning models across diverse datasets and real-world scenarios.

In the process of developing a concatenated model, we have fine-tuned three other individual models ResNet, EfficientNet, MobileNet. These models validation is shown below

ResNet Validation :

Accuracy on Validation set : 98.63%

Classification Report :

Table 6.1 ResNet Validation accuracy

Plant/Disease	Precision	Recall	F1 Score
Apple - Apple scab	1	0.99	0.99
Apple - Black rot	1	1	1
Apple - Cedar apple rust	0.99	0.99	0.99
Apple - healthy	0.99	1	1
Blueberry - healthy	1	1	1
Cherry (including sour) - Powdery mildew	1	1	1
Cherry (including sour) - healthy	1	1	1
Corn (maize) - Cercospora leaf spot Gray leaf spot	0.96	0.94	0.95
Corn (maize) - Common rust	1	1	1
Corn (maize) - Northern Leaf Blight	0.94	0.97	0.95
Corn (maize) - healthy	1	1	1
Grape - Black rot	0.99	0.99	0.99
Grape - Esca (Black Measles)	0.99	0.99	0.99
Grape - Leaf blight (Isariopsis Leaf Spot)	1	1	1
Grape - healthy	1	1	1
Orange - Haunglongbing (Citrus greening)	0.99	1	1
Peach - Bacterial spot	0.99	0.99	0.99

Plant/Disease	Precision	Recall	F1 Score
Peach - healthy	1	1	1
Pepper, bell - Bacterial spot	1	1	1
Pepper, bell - healthy	0.99	0.99	0.99
Potato - Early blight	0.99	1	0.99
Potato - Late blight	0.98	0.99	0.98
Potato - healthy	0.99	0.99	0.99
Raspberry - healthy	1	1	1
Soybean - healthy	1	0.99	1
Squash - Powdery mildew	1	1	1
Strawberry - Leaf scorch	1	1	1
Strawberry - healthy	1	1	1
Tomato - Bacterial spot	0.98	0.98	0.98
Tomato - Early blight	0.95	0.95	0.95
Tomato - Late blight	0.96	0.95	0.96
Tomato - Leaf Mold	0.99	0.98	0.98
Tomato - Septoria leaf spot	0.97	0.95	0.96
Tomato - Spider mites Two-spotted spider mite	0.95	0.96	0.95
Tomato - Target Spot	0.93	0.94	0.93
Tomato - Tomato Yellow Leaf Curl Virus	0.99	1	0.99
Tomato - Tomato mosaic virus	0.99	0.99	0.99
Tomato - healthy	0.99	0.99	0.99

EfficientNet Validation :

Accuracy on Validation set : 93.92%

Classification Report :

Table 6.2 EfficientNet Validation accuracy

Plant/Disease	Precision	Recall	F1 Score
Apple - Apple scab	0.95	0.9	0.92
Apple - Black rot	0.97	0.96	0.96
Apple - Cedar apple rust	0.93	0.98	0.96
Apple - healthy	0.94	0.95	0.94
Blueberry - healthy	0.95	0.97	0.96
Cherry (including sour) - Powdery mildew	0.97	0.96	0.97
Cherry (including sour) - healthy	0.97	0.99	0.98
Corn (maize) - Cercospora leaf spot Gray leaf spot	0.91	0.85	0.88
Corn (maize) - Common rust	0.99	0.99	0.99
Corn (maize) - Northern Leaf Blight	0.86	0.94	0.89

Plant/Disease	Precision	Recall	F1 Score
Corn (maize) - healthy	0.98	0.99	0.99
Grape - Black rot	0.97	0.97	0.97
Grape - Esca (Black Measles)	0.98	0.97	0.97
Grape - Leaf blight (Isariopsis Leaf Spot)	0.97	0.98	0.97
Grape - healthy	0.99	1	0.99
Orange - Haunglongbing (Citrus greening)	0.98	0.99	0.99
Peach - Bacterial spot	0.96	0.94	0.95
Peach - healthy	0.96	0.97	0.96
Pepper, bell - Bacterial spot	0.93	0.95	0.94
Pepper, bell - healthy	0.95	0.94	0.94
Potato - Early blight	0.96	0.98	0.97
Potato - Late blight	0.94	0.95	0.94
Potato - healthy	0.96	0.97	0.97
Raspberry - healthy	1	0.98	0.99
Soybean - healthy	0.96	0.96	0.96
Squash - Powdery mildew	0.98	0.99	0.99
Strawberry - Leaf scorch	1	0.99	0.99
Strawberry - healthy	0.98	0.99	0.99
Tomato - Bacterial spot	0.89	0.91	0.9
Tomato - Early blight	0.82	0.77	0.79
Tomato - Late blight	0.88	0.79	0.83
Tomato - Leaf Mold	0.88	0.91	0.9
Tomato - Septoria leaf spot	0.84	0.83	0.83
Tomato - Spider mites Two-spotted spider mite	0.88	0.91	0.89
Tomato - Target Spot	0.8	0.77	0.78
Tomato - Tomato Yellow Leaf Curl Virus	0.96	0.95	0.96
Tomato - Tomato mosaic virus	0.94	0.96	0.95
Tomato - healthy	0.91	0.92	0.92

MobileNet Validation :

Accuracy on Validation set : 98.05%

Classification Report :

Table 6.3 MobileNet Validation accuracy

Plant/Disease	Precision	Recall	F1 Score
Apple - Apple scab	0.98	0.99	0.98
Apple - Black rot	1	1	1
Apple - Cedar apple rust	1	0.98	0.99
Apple - healthy	0.98	0.99	0.99
Blueberry - healthy	0.99	1	1
Cherry (including sour) - Powdery mildew	0.99	1	0.99

Plant/Disease	Precision	Recall	F1 Score
Cherry (including sour) - healthy	1	1	1
Corn (maize) - Cercospora leaf spot Gray leaf spot	0.96	0.96	0.96
Corn (maize) - Common rust	1	0.99	1
Corn (maize) - Northern Leaf Blight	0.96	0.96	0.96
Corn (maize) - healthy	1	1	1
Grape - Black rot	0.99	0.99	0.99
Grape - Esca (Black Measles)	0.99	0.99	0.99
Grape - Leaf blight (Isariopsis Leaf Spot)	1	1	1
Grape - healthy	1	0.99	1
Orange - Haunglongbing (Citrus greening)	1	1	1
Peach - Bacterial spot	0.97	0.98	0.98
Peach - healthy	1	0.99	0.99
Pepper, bell - Bacterial spot	1	0.99	0.99
Pepper, bell - healthy	0.98	0.99	0.99
Potato - Early blight	1	0.99	0.99
Potato - Late blight	0.99	0.97	0.98
Potato - healthy	0.99	0.97	0.98
Raspberry - healthy	1	1	1
Soybean - healthy	0.98	0.99	0.98
Squash - Powdery mildew	1	1	1
Strawberry - Leaf scorch	1	1	1
Strawberry - healthy	1	0.99	0.99
Tomato - Bacterial spot	0.99	0.97	0.98
Tomato - Early blight	0.91	0.95	0.97
Tomato - Late blight	0.95	0.95	0.98
Tomato - Leaf Mold	0.94	0.98	0.93
Tomato - Septoria leaf spot	0.93	0.95	0.94
Tomato - Spider mites Two-spotted spider mite	0.93	0.95	0.94
Tomato - Target Spot	0.97	0.82	0.89
Tomato - Tomato Yellow Leaf Curl Virus	0.99	0.99	0.99
Tomato - Tomato mosaic virus	0.98	1	0.99
Tomato - healthy	0.96	1	0.98

LeafNet Validation :

Accuracy on Validation set : 99.29%

Classification Report :

Table 6.4 LeafNet Validation accuracy

Plant/Disease	Precision	Recall	F1 Score
Apple - Apple scab	1	1	1
Apple - Cedar apple rust	1	1	1

Plant/Disease	Precision	Recall	F1 Score
Apple - healthy	0.99	1	1
Blueberry - healthy	1	1	1
Cherry (including sour) - Powdery mildew	1	1	1
Cherry (including sour) - healthy	1	1	1
Corn (maize) - Cercospora leaf spot Gray leaf spot	0.97	0.95	0.96
Corn (maize) - Common rust	1	1	1
Corn (maize) - Northern Leaf Blight	0.96	0.97	0.97
Corn (maize) - healthy	1	1	1
Grape - Black rot	1	0.99	0.99
Grape - Esca (Black Measles)	0.99	1	0.99
Grape - Leaf blight (Isariopsis Leaf Spot)	1	1	1
Grape - healthy	1	1	1
Orange - Haunglongbing (Citrus greening)	1	1	1
Peach - Bacterial spot	1	1	1
Peach - healthy	1	1	1
Pepper, bell - Bacterial spot	1	1	1
Pepper, bell - healthy	1	1	1
Potato - Early blight	0.99	1	1
Potato - Late blight	1	1	1
Potato - healthy	1	1	1
Raspberry - healthy	1	1	1
Soybean - healthy	1	1	1
Squash - Powdery mildew	1	1	1
Strawberry - Leaf scorch	1	1	1
Strawberry - healthy	1	1	1
Tomato - Bacterial spot	1	0.99	0.99
Tomato - Early blight	0.97	0.98	0.97
Tomato - Late blight	0.98	0.97	0.98
Tomato - Leaf Mold	0.99	0.99	0.99
Tomato - Septoria leaf spot	0.99	0.97	0.98
Tomato - Spider mites Two-spotted spider mite	0.97	0.97	0.97
Tomato - Target Spot	0.96	0.95	0.96
Tomato - Tomato Yellow Leaf Curl Virus	0.99	1	1
Tomato - Tomato mosaic virus	1	1	1
Tomato - healthy	1	1	1

6.4 SUMMARY

The validation and testing phase of our AI project played a crucial role in evaluating the performance and robustness of the developed models. Through rigorous validation techniques, we assessed the generalization capability of our models and identified any potential issues such as overfitting or underfitting. By splitting the dataset into training, validation, and testing sets, we ensured that our models were trained on diverse data and could accurately predict outcomes on unseen instances.

During validation, we fine-tuned hyperparameters, optimized model architectures, and monitored key performance metrics such as accuracy, precision, recall, and F1-score. This iterative process allowed us to refine our models and enhance their predictive capabilities. Subsequently, rigorous testing against unseen data helped validate the real-world applicability of our models and provided insights into their performance under various scenarios.

Overall, the validation and testing phase provided critical insights into the effectiveness and reliability of our AI models, paving the way for their deployment in real-world applications.

Table 6.5 Model Accuracy

Model	Accuracy
ResNet	98.63 %
EfficientNet	93.92 %
MobileNet	98.05 %
LeafNet (Concatenated Model)	99.29 %

CHAPTER-7

CONCLUSION

CHAPTER-7

CONCLUSION

7.1 CONCLUSION

By harnessing deep learning techniques and state-of-the-art neural network architectures such as ResNet, EfficientNet, and MobileNet, Each model has exhibited an accuracy of 98.63%, 93.92%, 98.05%. We have created a powerful framework capable of accurately identifying plant species and detecting leaf diseases. The integration of classification and feature selection techniques has further enhanced the accuracy and efficiency of the models, ensuring precise predictions and informed decision-making. The Concatenated model is performing with an accuracy of 99.29%.

The implementation of a user-friendly web-based platform using Streamlit facilitates seamless interaction between users and the AI-powered system, providing easy access to comprehensive plant health information and personalized suggestions. This platform serves as a valuable resource for plant enthusiasts and professionals alike, enabling them to make informed decisions and effectively manage plant health.

In conclusion, the Digital Phenoparenting project holds immense potential to transform the way we monitor and manage plant health, offering a streamlined, efficient, and accessible solution for plant care and management. With further refinement and deployment, this project has the capability to make a significant impact in agriculture, horticulture, and plant science domains, ultimately contributing to global food security and environmental sustainability.

7.2 FUTURE ENHANCEMENTS

- To perform pheno parenting on complete plant level.
- Develop real-time monitoring solutions utilizing IoT devices, drones to continuously monitor plant health indicators. Integrate the AI models into these systems for early detection of plant diseases and timely intervention.
- Performing the masking of a specified phenotype of the plant can give much awareness on area of damage in plants

8. REFERENCES

- [1] Chrisbin James, Daniel Smith, Weigao He, Shekhar S. Chandra, Scott C. Chapman, “GrainPointNet: A deep-learning framework for non-invasive sorghum panicle grain count phenotyping”, Volume 217, 2024, <https://www.sciencedirect.com/science/article/pii/S0168169923008736>
- [2] Ajay Kumar Patel, Eunsung Park, Hongseok Lee, G. G. Lakshmi Priya, “Deep Learning-Based Plant Organ Segmentation and Phenotyping of Sorghum Plants Using LiDAR Point Cloud”, Volume 16, 2023, <https://ieeexplore.ieee.org/document/10243159>
- [3] Nariman Niknejad, Rafael Bidese-Puhl, Kitt Payn, “Phenotyping of architecture traits of loblolly pine trees using stereo machine vision and deep learning: Stem diameter, branch angle, and branch diameter”, Volume 211, 2023. <https://www.sciencedirect.com/science/article/abs/pii/S0168169923003873>
- [4] Yuwei Lu, Jinhu Wang, Ling Fu, Lejun Yu, Qian Liu, “High-throughput and separating-free phenotyping method for on-panicle rice grains based deep learning Volume 14, 2023. <https://www.frontiersin.org/journals/plantscience/articles/10.3389/fpls.2023.1219584/full>
- [5] Deepti Barhate, Sunil Pathak, Ashutosh Kumar Dubey, “A DEEP LEARNING METHODOLOGY FOR PLANT SPECIES RECOGNITION USING MORPHOLOGY OF LEAVES”, Volume 13, 2023, <https://journals.utm.my/aej/article/view/19461>
- [6] Anirban Jyoti Hati and Rajiv Ranjan Singh “Artificial Intelligence in Smart Farms: Plant Phenotyping for Species Recognition and Health Condition Identification Using Deep Learning “2021, 274–289. <https://doi.org/10.3390/ai2020017>
- [7] Angelo Cardellicchio, Firozeh Solimani, Giovanni Dimauro, Angelo Petrozza, Stephan Summerer, Francesco Cellini, Vito Renò, ” Detection of tomato plant phenotyping traits using YOLOv5-based single stage detectors”, Volume 207, 2023, <https://www.sciencedirect.com/science/article/pii/S016816992300145X>
- [8] J.Dhakshavani, John Fowler"Deep Learning Applications in Precision Agriculture: Advanced Plant Phenotyping for Species Identification and Health Assessment in Smart Farming" <https://doi.org/10.1007/s00497-021-00407-2>
- [9] Guo X, Qiu Y, Nettleton D, Schnable PS. High-Throughput Field Plant Phenotyping: A Self-Supervised Sequential CNN Method to Segment Overlapping Plants. *Plant Phenomics* 2023;5:Article 0052. <https://doi.org/10.34133/plantphenomics.0052>
- [10] Radek Zenkl, Radu Timofte, Norbert Kirchgessner, Lukas Roth, Andreas Hund, Luc Van Goo, Achim Walter, Helge Aasen, “Outdoor Plant Segmentation With Deep Learning for High-Throughput Field Phenotyping on a Diverse Wheat Dataset”, Volume 12, 2021. <https://www.frontiersin.org/journals/plantscience/articles/10.3389/fpls.2021.774068/full>
- [11] Ehsan Ullah, Mohib Ullah, Sajjad Muhammed, Faouzi Alava Cheikh, “Deep learning based wheat ears count in robot images for wheat phenotyping”, Volume 34, 2022, <https://library.imaging.org/ei/articles/34/6/IRIACV-264>

- [12] Chaoxin Wang, Doina Caragea, Nisarga Kodadinne Narayana, Nathan T. Hein, Raju Bheemanahalli, Impa M. Somayanda, S. V. Krishna Jagadish, “Deep learning based high-throughput phenotyping of chalkiness in rice exposed to high night temperature”, Volume 18, 2022,
<https://plantmethods.biomedcentral.com/articles/10.1186/s13007-022-00839>
- [13] Anirban Jyoti Hati, Rajiv Ranjan Singh, “Artificial Intelligence in Smart Farms: Plant Phenotyping for Species Recognition and Health Condition Identification Using Deep Learning”, Volume 2, 2021,
<https://www.mdpi.com/2673-2688/2/2/17>
- [14] David Schunck, Federico Magistri, Radu Alexandru Rosu, Andre’ Cornelißen, Nived Chebrolu, Stefan Paulus, Jens Leon, Sven Behnke, Cyrill Stachniss, Heiner Kuhlmann, Lasse Klingbeil, “Pheno4D: A spatio-temporal dataset of maize and tomato plant point clouds for phenotyping and advanced plant analysis”, Volume 3, 2021,
<https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0256340>
- [15] Douglas Pinto Sampaio Gomes, Lihong Zheng, “Recent Data Augmentation Strategies for DeepLearning in Plant Phenotyping and Their Significance”, Volume 14, 2020,
<https://ieeexplore.ieee.org/document/9363383>
- [16] Marni Tausen, Marc Clausen, Sara Moeskjær, ASM Shihavuddin, Anders Bjorholm Dahl, Luc Janss and Stig Uggerhøj Andersen [Greenotyper: Image-Based Plant Phenotyping Using Distributed Computing and Deep Learning ec. Volume 11 - 2020
<https://doi.org/10.3389/fpls.2020.01181>
- [17] Asheesh Kumar Singh, Baskar Ganapathysubramanian, Soumik Sarkar, Arti Singh, “Deep Learning for Plant Stress Phenotyping: Trends and Future Perspectives”, Volume 23, 2018,
<https://www.sciencedirect.com/science/article/pii/S1360138518301572>
- [18] Jordan R Ubbens, Ian Stavness, “Deep Plant Phenomics: A Deep Learning Platform for Complex Plant Phenotyping Tasks”, Volume 8, 2017,
<https://www.frontiersin.org/journals/plantscience/articles/10.3389/fpls.2017.01190/full>
- [19] Michael P. Pound, Jonathan A. Atkinson, Darren M. Wells, Tony P. Pridmore, Andrew P. French, “Deep Learning for Multi-task Plant Phenotyping”, Volume 6, 2017,
<https://ieeexplore.ieee.org/document/8265451>
- [20] New Plant Disease Dataset from Kaggle
<https://www.kaggle.com/datasets/vipooool/new-plant-diseases-dataset>
- [21] Pytorch documentation
<https://pytorch.org/docs/stable/index.html>