IDENTIFICATION OF MEDICINAL PLANSTS USING MACHINE LEARNING ALGORITHM

A PROJECT REPORT

Submitted by

KUNDERU LOKESH[RA2111043020033]
SHAIK SAHIL[RA2111043020036]
DHARNISH BARATH S R[RA2111043020026]

Under the guidance of

Dr. R. KARTHIGA, Ph.D.

(Assistant Professor, Department of Electronics and Communication Engineering)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMPUTER ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM

NOV 2024

BONAFIDE CERTIFICATE

Certified that this project report titled "**INDENTIFICATION OF MEDICINAL PLANTS USING MACHINE LEARNING ALGORITHM**" is the Bonafide work of "**KUNDERU LOKESH[RA2111043020033], SHAIK SAHIL[RA2111043020036], DHARNISH BARATH S R[RA2111043020026]**" who carried out the project work under my supervision as a batch. Certified further, that to the best of my knowledge the work reported herein does not form any other project report on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Signature **Dr.R . KARTHIGA, Ph.D.** **Department of ECE** **SRM institute of Science & Technology** **Ramapuram Campus** **Chennai - 600089**	**Signature** **Dr. N. V. S. SREE RATHNA LAKSHMI** **Professor & Head** **Department of ECE** **SRM institute of Science & Technology** **Ramapuram Campus** **Chennai - 600089**

Submitted for University Examination held on 12/11/2024 in the Department of Electronics and Communication Engineering, SRM Institute of Science and Technology, Ramapuram.

Date:12/11/2024

Internal Examiner 1 **Internal Examiner 2**

DECLARATION

We hereby declare that the Major Project entitled "**IDENTIFICATION OF MEDICINAL PLANTS USING MACHINE LEARNING ALGORITHM**" to be submitted for the Degree of Bachelor of Technology is our original work as a team and the dissertation has not formed the basis of any degree, diploma, associateship or fellowship of similar other titles. It has not been submitted to any other University or institution for the award of any degree or diploma.

Place: Chennai

Date:12/11/2024

KUNDERU LOKESH [RA2111043020033]

SHAIK SAHIL [RA2111043020036]

DHARNISH BARATH S R [RA2111043020026]

ABSTRACT

Our project harnesses the power of computer vision and machine learning to address a significant need in the fields of herbal medicine and biodiversity conservation. This research project seeks to develop a sophisticated, highly accurate image classification system capable of identifying a wide variety of medicinal plants based on images of their leaves, flowers, stems, and other botanical features. By leveraging advanced image recognition algorithms and large datasets of plant images, this system aims to assist users in accurately and swiftly identifying medicinal plants in their natural environments.In addition to aiding researchers and enthusiasts, this project serves as an essential tool for individuals interested in the therapeutic benefits of these plants. In particular, it will help urban residents who may have limited access to traditional ayurvedic plants by providing them with information about government-recognized centers that sell these plants, as well as directions to the nearest locations. This approach not only promotes the accessibility of medicinal plants but also supports the preservation of these species by encouraging sustainable sourcing.Furthermore, the system aims to offer users personalized plant recommendations based on their medical history and health needs. By analyzing data such as previous searches and the user's specific health conditions, the system can suggest medicinal plants that may provide relevant therapeutic benefits. This feature makes it an invaluable tool for promoting personalized herbal remedies and fostering an increased awareness of the health benefits of medicinal plants. In sum, this project has the potential to contribute to both individual wellness and broader conservation efforts.

ACKNOWLEDGEMENTS

We are expressing our deep sense of gratitude to our beloved chancellor **Dr. T. R. PAARIVENDHAR**, for providing us with the required infrastructure throughout the course.

We take this opportunity to extend our hearty thanks to our Chairman **Dr. R. SHIVAKUMAR, SRM Ramapuram & Trichy Campus** for his constant support.

We take this opportunity to extend our hearty thanks to our Co-Chairman **Mr. S. NIRANJAN, SRM Ramapuram & Trichy Campus** for his constant support.

We take this opportunity to extend our hearty thanks to our Dean **Dr. M. SAKTHI GANESH, Ph.D.,** for his constant support.

We convey our sincere thanks to our Head of the Department **Dr. N.V.S. SREE RATHNA LAKSHMI, Ph.D**., for his suggestions, interest, encouragement and support throughout the project.

We convey our sincere thanks to our Project coordinator **Dr. K. DURGA DEVI, Ph.D**., for her suggestions, interest, encouragement and support throughout the project.

We express our heartfelt thanks to our guide **Dr.R.KARTHIGA, Ph.D**., Assistant Professor,for his sustained encouragement, and constant guidance throughout the project work.

We express our deepest gratitude to, our parents, Teaching and Non- Teaching faculties for their sustained encouragement, and constant support throughout my studies.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolution Neural Network
HDD	Hard Disk Drive
SSD	Soild State Drive
GPU	Graphics Processing Unit
CPU	Central Processsing Unit
RAM	Random Access Memory
BN	Batch Normalization
ReLU	Rectified Linear Unit
FC	Fully Connected
RGB	Red Green Blue
HDF5	Hierarchical Data Format (version 5)
VGG16	Visual Geometry Group (16 layers)

1 INTRODUCTION

1.1 Introduction

Ayurveda is an ancient system of medicine practiced in India and has its roots in the Vedic times, approximately 5000 years ago. The main constituents of ayurvedic medicines are plant leaves and other parts of plants like root, bark etc. More than 8000 plants of Indian origin have been found to be of medicinal value. Combinations of a small subset amounting to 1500 of these plants are used in Herbal medicines of different systems of India. Specifically, commercial Ayurvedic preparations use 500 of these plants. Over 80% of plants used in ayurvedic formulations are collected from the forests and wastelands whereas the remaining are cultivated in agricultural lands. In the ancient past, Ayurvedic physicians themselves picked the medicinal plants and prepared the medicines for their patients. Most of the plants are identified using their leafs the common steps to classify the leaf of a plants are Capturing image, noise removal and resizing extracting features, use proposed methodology and finally identify or recognize the plant.

1.1.1 Problem Statement

The accurate identification and preservation of medicinal plants are critical for both healthcare and environmental conservation. Traditional medicine relies heavily on these plants, but their correct identification and classification remain challenging. Current methods often lack efficiency and precision, especially for diverse species with similar features. This project addresses the need for a reliable and automated system to identify medicinal plants using image classification.By leveraging computer vision and machine learning techniques, specifically Convolutional Neural Networks (CNNs), the project aims to develop an advanced classification system that can accurately identify and categorize medicinal plants from images of their leaves, flowers, and other features.This will bridge the gap between traditional herbal knowledge and modern technology, enhancing the sustainable use of medicinal plants and deepening our understanding of their medicinal properties.

1.1.2 Objective of the Project

- Develop a comprehensive dataset.
- Implement Image preprocessing techniques.
- Evaluate and Select Machine Learning Algorithms.
- Incorporate Deep Learning Architectures.
- Develop and Train Machine Learning Models.
- Develop a User-Friendly Interface.

These objectives provide a structured framework for our research project, guiding the implementation and evaluation of the medicinal plant identification system using image processing and machine learning. Adjust and refine these objectives based on the specific context and scope of our study

1.1.3 Brief description of project

This project follows a systematic approach to build an effective medicinal plant identification system. First, a diverse dataset of medicinal plant images is collected and preprocessed, with data augmentation applied to enhance the model's generalization. A Convolutional Neural Network (CNN) is then developed, trained on these images, and evaluated using accuracy, precision, and recall to ensure reliability. Once trained, the model is saved and deployed to allow users to upload images for real-time identification. A PostgreSQL database stores user profiles, medical histories, and plant data to enable personalized plant-based remedy recommendations. The system also supports continuous improvement through model retraining and user feedback.

1.1.4 Scope of the Project

The project involves collecting a diverse set of medicinal plant images and augmenting the data to improve the model's generalization. A deep learning model (CNN) is developed and trained to classify plants based on their appearance, with adjustments made to optimize accuracy and scalability. The system allows users to upload plant images for realtime identification, check online availability, and locate nearby government-recognized centers. Personalized plant-based remedy recommendations are provided based on the user's medical history stored in a Local database. The trained model is deployed for real-time predictions, and the system's performance is evaluated to ensure responsiveness and robustness.

1.2 Literature survey

1.2.1MedLeaf:Mobile Application for Medicinal Plant Identification Based on Leaf Image.

Authors: Prasvita D.S, Herdiyeni Y

Proposed Year:2013

Given the biodiversity crisis and the underutilization of Indonesia's medicinal plants, there is an urgent need for improved documentation and public access to information. The proposed mobile application for automatic plant identification seeks to address this gap by facilitating easier and more efficient recognition of medicinal plants, which could contribute to better conservation practices and increased use of plant resources

1.2.2Title: Plant identification using deep neural networks via optimization of transfer learning parameters

Authors: Mehdipour Ghazi ,Berrin Yanikog, Erchan Aptoula

Proposed Year:2015

Automated plant identification classifies plant organs into species using machine learning, facing challenges like inter-class similarities and intra-class variations in background and illumination. Deep learning has become popular for its ability to automatically extract complex features from data, overcoming limitations of traditional methods. By utilizing large datasets and high-performance computing, deep learning networks effectively address these challenges.

1.2.3 Title: Plant species classification using deep convolutional neural network.

Authors: Dyrmann M, Karstoft H, Midtiby H.S

Proposed Year:2016

The paper introduces a method for plant species recognition using a convolutional neural network (CNN). The CNN, which was built, trained, and tested from scratch, was evaluated on 10,413 images of 22 weed and crop species in early growth stages. These images were sourced from six diverse datasets with variations in lighting, resolution, and soil types, including both controlled settings and real-field conditions. The network achieved a classification accuracy of 86.2% for the 22 species.

1.2.4 **Title**: **Automated plant identification using artificial neural network and support vector machine.**

Authors: Soon Jye, Sugumaran Manickam, Sorayya Malek, Mogeeb Mosleh, Sarinder Kaur Dhillon

Proposed Year:2018

This study shows that automated identification techniques can effectively distinguish three Ficus species using herbarium leaf images. Utilizing artificial neural networks (ANN) and support vector machines (SVM), the models achieved satisfactory accuracy, offering efficient tools for plant identification, especially for non-experts. While not a substitute for human taxonomists, these methods provide rapid and accessible classification. Future work could enhance robustness by incorporating more Ficus species.

1.2.5 Title: Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks

Authors: Elhassouny A,Smarandache F

Proposed Year:2019

The increasing crop losses from plant diseases, exacerbated by environmental factors, have led farmers to adopt more efficient, technology-driven solutions. The use of mobile phones and digital tools for disease diagnosis and management is improving agricultural practices and addressing the challenges posed by traditional diagnostic methods and environmental changes.

1.2.6 Title:Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild

Authors: Picon A, Alvarez-Gila A, Seitz M, Ortiz-Barredo A, Echazarra J,Johannes A

Proposed Year:2019

To manage the severe impact of fungal pathogens on crop yields and address the challenge of accurate disease identification, there is a pressing need for accessible and efficient identification tools. These tools would enable timely and cost-effective crop protection measures, especially in regions with limited expert access and increasing pathogen resistance.

1.2.7 Title:Early Blight Identification in Tomato Leaves using Deep Learning.

Authors: Chakravarthy, A.S., Raman S

Proposed Year:2020

The journal highlights the critical need for precise and timely detection of Early Blight in tomatoes, achieved through high-accuracy classification using neural networks and improved with object detection methods. By integrating these technologies, the research seeks to enhance the development of a mobile application that can provide accurate and real-time disease identification, addressing a significant challenge in tomato crop management.

1.2.8 Title:An AI Based Approach for Medicinal Plant Identification Using Deep CNN Based on Global Average Pooling

Authors: Rahim Azadnia, Mohammed Maitham Al-Amidi,
Hamed Mohammadi, Mehmet Akif, Cifci, Avat Daryab, Eugenio Cavallo

Proposed Year:2022

The current study aimed to improve the automatic identification of medicinal plants due to their growing popularity and increased requests for artisanal and industrial uses and applications. Therefore, the proposed DL algorithm and image processing technique can have a special place in plant science and even industrial markets for identifying and classifying varied medicinal plants separately from other non-edible plants.

2 METHODOLOGY

2.1 Methodology

The methodology for this project involves collecting and preprocessing a diverse dataset of medicinal plant images, applying data augmentation techniques to improve model generalization. A Convolutional Neural Network (CNN) is developed and trained to classify plant species based on their visual features, with performance evaluated using accuracy, precision, and recall metrics.

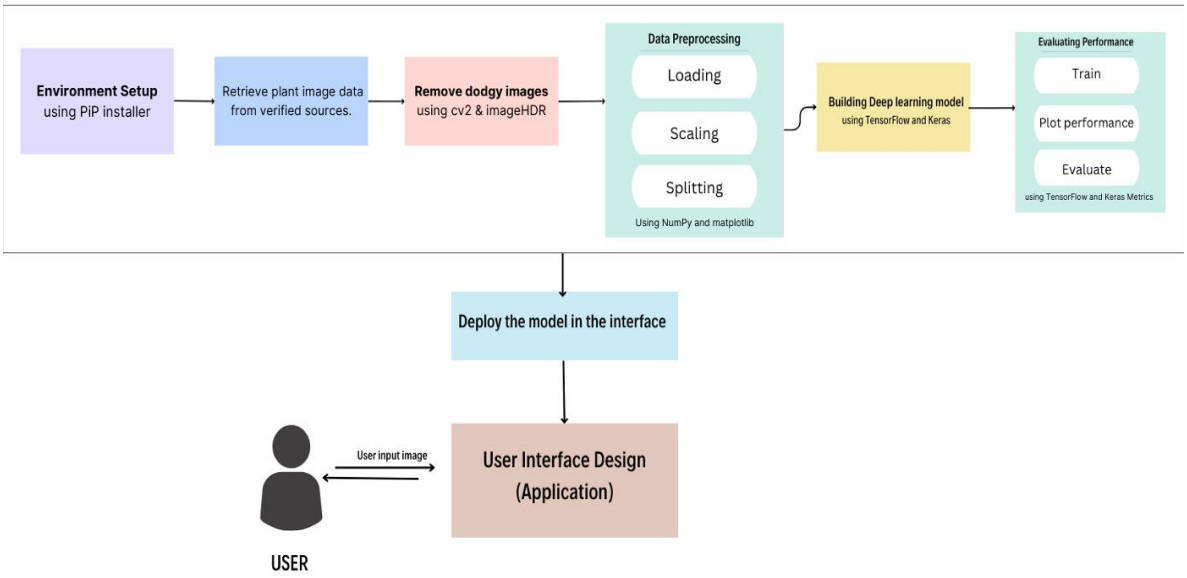

Fig 2.1 Block Diagram

The trained model is then saved and deployed, allowing users to upload images for real-time plant identification and receive predictions locally. A local PostgreSQL database stores user profiles, medical history, and plant information, which is used to provide personalized plant-based remedy recommendations. The system ensures accurate and efficient classification while continuously improving through model retraining and feedback integration.

2.2Hardware Requirement

2.2.1GPU (Graphics Processing Unit)

- Recommended GPU:
    - NVIDIA RTX 3060 or higher: The RTX 3060 (12GB VRAM) or similar cards can handle moderate CNN architectures on 128x128 image inputs quite efficiently.
- VRAM Requirement:
    - For this architecture and 128x128 images, 8GB VRAM should be sufficient for training without memory issues, though 12GB or more is ideal if you plan to experiment with larger images or deeper models.

2.2.2CPU (Central Processing Unit)

- Recommended CPU:

  A multi-core processor, like an Intel i5 (8th Gen or higher) or AMD Ryzen 5, should be sufficient, If you plan to experiment heavily with data augmentation or run multiple processes in parallel, a more powerful CPU, like an Intel i7/i9 or AMD Ryzen 7/9, will offer faster data preprocessing times.
- Cores:

  At least 4 cores are recommended, though 6 or more cores will make preprocessing faster.

2.2.3RAM (Memory)

- Recommended RAM:
    - 8GB RAM is the minimum, but 16GB RAM is recommended for smoother operation, especially if you're working with a larger dataset or multitasking during training.
    - If working with very large datasets or complex models, 32GB RAM may be beneficial to avoid memory swapping, which slows down processing.

2.2.4. Storage

- Recommended Storage:
    - SSD (Solid State Drive): An SSD can significantly reduce loading times for

large datasets compared to an HDD (Hard Disk Drive). An SSD is recommended if you need to load many images quickly.

- Capacity, if it is for a medium-sized image dataset, at least 256GB of SSD storage is recommended, though 512GB or more is better if you'll be saving multiple models or working with larger datasets.

2.3 Software requirement

2.3.1Operating System:

Compatible with Windows, macOS, or Linux. Linux is preferred for better GPU support.

2.3.2 Python:

For this project we recommed the newest version of python which is Python 3.8.

Python Libraries:

- TensorFlow 2.x, developed by Google, is a powerful open-source framework for building and deploying machine learning models, especially in deep learning. It provides an extensive set of tools for both beginners and experts, with APIs that facilitate model creation and optimization across various platforms. Built on top of TensorFlow, Keras is a high-level neural network API designed for rapid prototyping, offering an intuitive interface for defining and training models with minimal code. Another crucial library in machine learning workflows is NumPy, which handles numerical operations and array manipulation, forming the foundation for many data science and machine learning libraries. Additionally, Pillow (PIL) is essential for computer vision tasks, allowing for image processing and manipulation like resizing, cropping, and format conversion. Together, these libraries form a robust ecosystem that empowers Python for machine learning, numerical computing, and computer vision.

2.3.3Visual Studio:

Visual Studio Code (VS Code) provides a range of features that enhance the development process, especially for machine learning projects. Its ease of development is supported by intelligent code editing tools, such as auto-completion

and real-time error detection, which streamline writing and debugging machine learning models. The integrated debugging tools and Jupyter notebook support allow for efficient testing and optimization of models directly within the editor. In addition, VS Code's data management capabilities help developers keep datasets, scripts, and other resources well-organized, ensuring a structured workflow. The editor also boosts efficiency by allowing developers to run Python scripts directly in the workspace and manage virtual environments, which helps streamline the process of handling dependencies. Together, these features make VS Code a powerful and productive environment for building and refining machine learning models.

2.4Modules

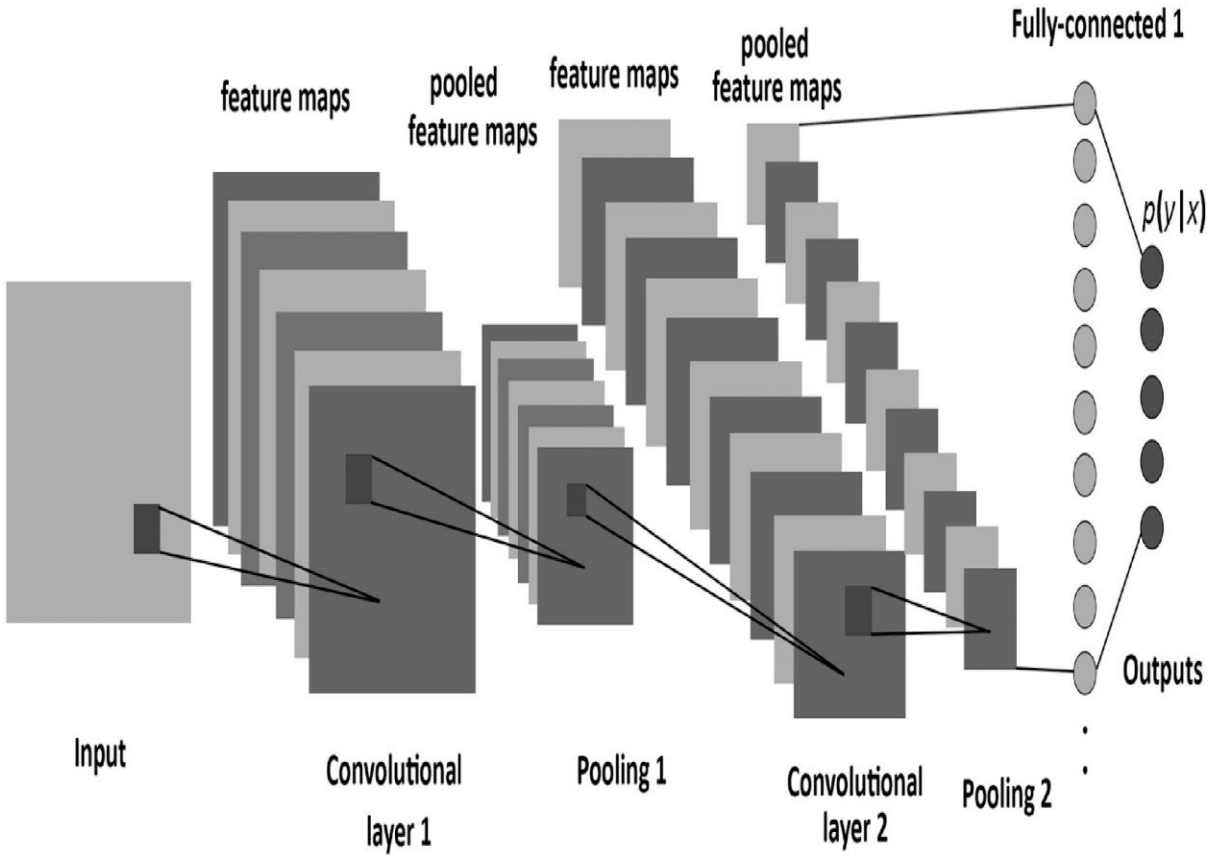

Fig 2.2 Modules

2.4.1 Input Layer:

In a Convolutional Neural Network (CNN) for medicinal plant image classification, images are preprocessed before being fed into the network. This includes resizing images to a standard size, such as 128x128 or 224x224 pixels, for consistency. Normalization scales pixel values to a range of 0 to 1 by dividing by 255, improving training stability. Additionally, augmentation techniques like rotation and flipping are used to increase dataset diversity, helping the model generalize better and preventing overfitting. These preprocessing steps ensure the network can effectively learn from the input images.

2.4.2. Convolutional Layers:

In a Convolutional Neural Network (CNN), convolutional layers play a key role in learning hierarchical features from input images. These layers use filters (kernels) that slide over the image to detect basic patterns like edges and textures in early layers, and more complex features like plant shapes or specific parts such as leaves and stems in deeper layers. The stride controls the movement of the filter across the image, while padding may be applied to maintain the spatial dimensions of the output, ensuring important features are preserved during the convolution process.

2.4.3. Activation Function (ReLU):

Rectified Linear Unit (ReLU) is the most commonly used activation function. It introduces non-linearity by replacing all negative values with 0, while keeping positive values unchanged. This helps the network capture complex patterns in images. ReLU is computationally efficient and alleviates the vanishing gradient problem, allowing for faster learning during backpropagation. Without ReLU or similar activation functions, the network would only learn linear mappings, limiting its ability to model complex relationships in the data.

2.4.4. Pooling Layers:

Downsampling in Convolutional Neural Networks (CNNs) is achieved through pooling layers, which reduce the spatial dimensions of the feature maps. Max pooling is a common method, where the maximum value from a patch of the feature map is selected, helping retain the most important features. Alternatively, average pooling takes the average value from the region, providing a smoother representation of the data. Both methods help reduce computational complexity and prevent overfitting while retaining essential information from the feature maps.

2.4.5.Batch Normalization:

Batch Normalization (BN) normalizes the activations of each layer in a neural network, ensuring they have a mean of zero and a standard deviation of one. This process improves training stability and performance. During training, for each mini-batch, the mean and variance of the activations from the previous layer are calculated. The activations are then scaled and shifted using learnable parameters, which helps maintain useful information while stabilizing the learning process and speeding up convergence.

2.4.6. Dropout Layer:

Dropout is a regularization technique used to reduce overfitting in neural networks by randomly "dropping out" a fraction of neurons during training. During each iteration, a percentage (e.g., 20-50%) of neurons are set to zero, encouraging the network to learn redundant representations and preventing it from becoming too reliant on specific neurons. Dropout is only applied during training; during inference (prediction), all neurons are used, and their outputs are scaled by the dropout rate to compensate for the missing neurons during training.

2.4.7. Convolutional Blocks

Convolutional blocks are a combination of convolutional layers, activation functions (typically ReLU), and pooling layers, designed to capture hierarchical features from input images. Each block typically consists of a convolutional layer that extracts features, followed by an activation function (ReLU) to introduce non-linearity, and a pooling layer to downsample the feature maps. By stacking multiple convolutional blocks in deeper layers, the network progressively learns more abstract and complex

representations of the input, such as plant images, enabling it to recognize intricate patterns and structures.

2.4.8 Fully Connected Layer

The Fully Connected (FC) layer is used in the final stages of a neural network to perform classification. It connects the flattened feature maps from previous layers, such as convolutional and pooling layers, to the output layer. First, the multidimensional feature maps are flattened into a one-dimensional vector. Each neuron in the FC layer is connected to every neuron in the previous layer, enabling the network to learn complex interactions between features. The FC layer generates a vector of raw outputs (logits), each representing a class in the classification task, and synthesizes information from the entire image to make the final prediction.

2.4.9 Loss Function

The loss function measures the difference between the predicted probabilities and the true class labels, guiding the optimization algorithm (like Gradient Descent) to update the model's weights during training. The true class labels are typically represented as one-hot encoded vectors, where the correct class is marked as "1" and all others as "0." The predicted probabilities are generated by the Softmax function, which outputs a probability distribution over all possible classes. The loss function calculates the discrepancy between these predicted probabilities and the true labels to guide the model’s learning.

Formula:

The loss for a single sample is calculated as

$$L = -\sum_{i=1}^{n} y_i \log(p_i)$$

where:

- yi is the true label (0 or 1) for class i,
- pi is the predicted probability for class i,
- n is the number of classes.

Averaging loss refers to calculating the total loss for the entire dataset by averaging the individual losses for each sample. This ensures that the loss function accounts for the performance across all data points and provides a more stable and generalized measure of the model's accuracy. By averaging the losses, the model is guided to make adjustments based on the overall performance, rather than overfitting to any specific sample.

3 IMPLEMENTATION

3.1 Experimental Setup

3.1.1. Dataset

- **Dataset Structure:**
  - The dataset is split into two main directories: Train and Test.
  - Each directory contains subdirectories for each class (i.e., different plant species).
  - The structure is expected to look like this:
- **Code**

dataset/

Train/

class_1/

img1.jpg

img2.jpg

class_2/

img1.jpg

Test/

class_1/

img1.jpg

img2.jpg

class_2/

img1.jpg

- **Classes:** The number of plant classes is specified as 4 in the code (i.e., num_plant_classes = 4). You can adjust this number based on your dataset.

3.1.2 Data Preprocessing

For training set preprocessing, the ImageDataGenerator is used for data augmentation and rescaling. The pixel values are normalized to the range [0, 1] using rescale=1./255, and augmentation techniques like random shear (shear_range=0.2), zoom (zoom_range=0.2), and horizontal flipping (horizontal_flip=True) are applied to increase dataset variability. For the test set preprocessing, only rescaling (rescale=1./255) is applied, as augmentation is not used to preserve the real-world data distribution. All images are resized to 128x128 pixels to standardize the input dimensions for the CNN model**.**

3.1.3 Model Architecture (CNN)

The model is a simple Convolutional Neural Network (CNN) designed for image classification. It starts with an input layer that accepts 128x128x3 RGB images. The first convolutional layer uses 50 filters of size 3x3 with ReLU activation, followed by a max pooling layer with a 2x2 pool size. The second convolutional layer has 60 filters of size 3x3 and ReLU activation, followed by another max pooling layer. After flattening the feature maps into a 1D vector, a fully connected layer with 128 units and ReLU activation learns more complex features. The final output layer has a number of units equal to num_plant_classes with softmax activation to produce class probabilities.

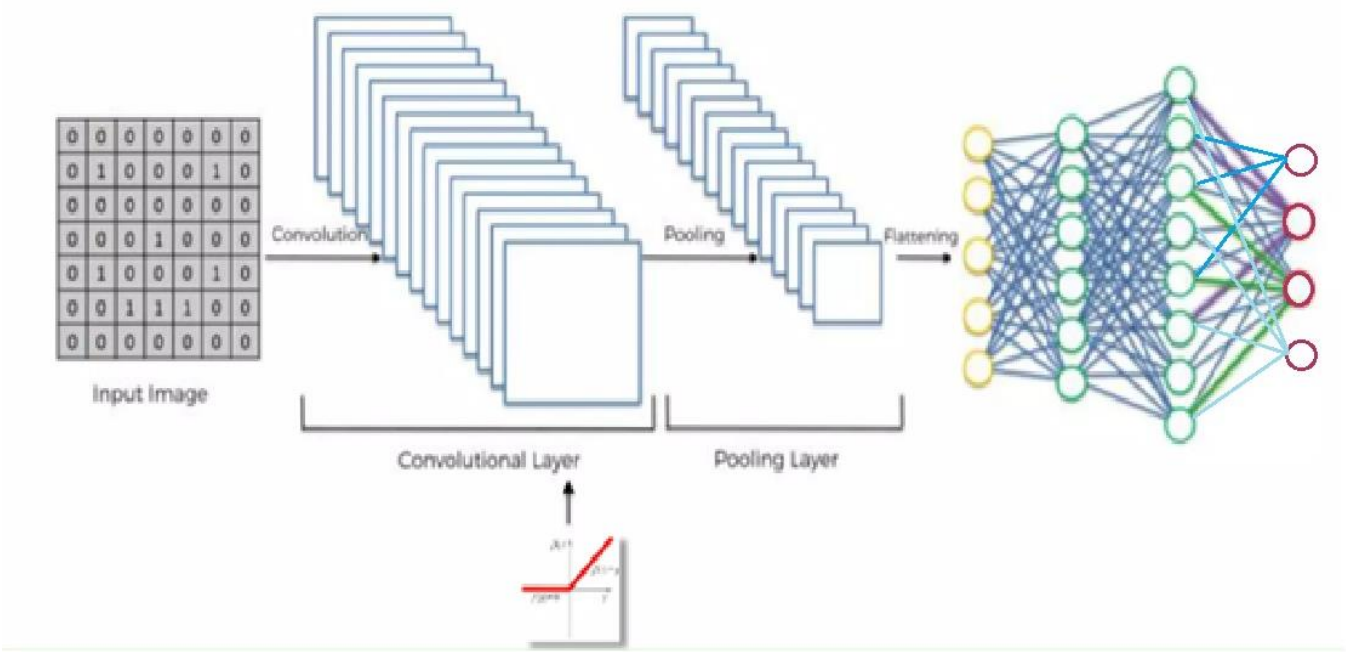

Fig 3.1 System Architecture

3.1.4. Compilation and Training

The model uses the Adam optimizer with a learning rate of 0.001, known for its efficiency and adaptive learning rate. Categorical crossentropy is the chosen loss function, suitable for multi-class classification tasks, while accuracy is used as the evaluation metric. The model is trained for 25 epochs with a batch size of 32 images. After each epoch, the model is evaluated on the test set, and the training process reports both training and validation accuracy and loss to monitor performance.

3.1.5 Making Predictions

A single image is loaded and preprocessed by resizing, converting it to an array, and normalizing the pixel values. The model then predicts the class of the image, and the predicted class is identified by finding the index of the maximum probability from the output of the softmax layer, which corresponds to the most likely class.

3.1.6 Model Saving and Loading

After training, the model is saved to a file named **plant_model.h5** for future use. To use the model for inference or continue training, it can be loaded with the function **load_model('plant_model.h5')**.

3.1.7 Execution Flow

The training and testing data are loaded using ImageDataGenerator. The CNN model is constructed layer by layer and then trained on the training data. After training, the model's performance is evaluated using the test data. The model can then be used for prediction on new images. Finally, the trained model is saved for future use and can be loaded later for further predictions or training.3.2 Program logic for parameter

3.2.1Data Preprocessing

- **ImageDataGenerator**

The ImageDataGenerator is used to preprocess and augment images. It rescales pixel values to the range [0, 1], applies random shearing (shear_range=0.2) and zooming (zoom_range=0.2) to enhance model robustness, and performs horizontal flipping (horizontal_flip=True) to generalize to flipped images. The flow_from_directory

function loads the images from specified directories, resizes them to 128x128 pixels, and processes them in batches (32 for training, 34 for testing). The class_mode='Categorical' indicates multi-class classification, where each image belongs to one of several classes.

3.2.2 Building the CNN

The model consists of several layers to process image data. It starts with two Conv2D layers, each using filters (50 for the first, 60 for the second) and a 3x3 kernel size to capture local features, followed by ReLU activation for non-linearity. MaxPool2D layers with a pool size of 2x2 reduce spatial dimensions by half, focusing on prominent features. A Dense layer with 128 neurons connects all previous nodes, learning complex patterns, using ReLU activation. The final output layer, a Dense layer with units equal to the number of plant classes (4), uses Softmax activation to classify the input image into one of the plant categories.

3.2.3 Training CNN

- **Optimizer (Adam)**

The model uses the Adam optimizer with a learning rate of 0.001, balancing stable convergence and training speed. The loss function, 'Categorical_crossentropy,' is used for multi-class classification, comparing predicted probabilities with true labels. Accuracy is chosen as the evaluation metric to track the percentage of correct predictions during training. The model is trained for 25 epochs, which determines the number of times the entire dataset is processed, with more epochs potentially improving accuracy but also increasing the training time.

3.2.4 Making a Single Prediction

The image.load_img function resizes the input image to a target size of (128, 128) to match the model's input shape. The expand_dims function adds an extra dimension to the image, changing its shape from (128, 128, 3) to (1, 128, 128, 3), representing a single image batch. The cnn.predict method generates a probability distribution over all possible classes, showing the likelihood that the image belongs to each class. Finally, np.argmax(result) identifies the class with the highest

predicted probability, making the final classification decision.

3.2.5 Model Saving and Loading

The cnn.save function saves the trained model in HDF5 format with the filename "plant_model.h5," allowing it to be stored for future use. The load_model function is used to load this saved model from "plant_model.h5," enabling the model to be used for predictions or further training without the need to retrain it from scratch.

4 RESULTS AND DISCUSSION

4.1 Results

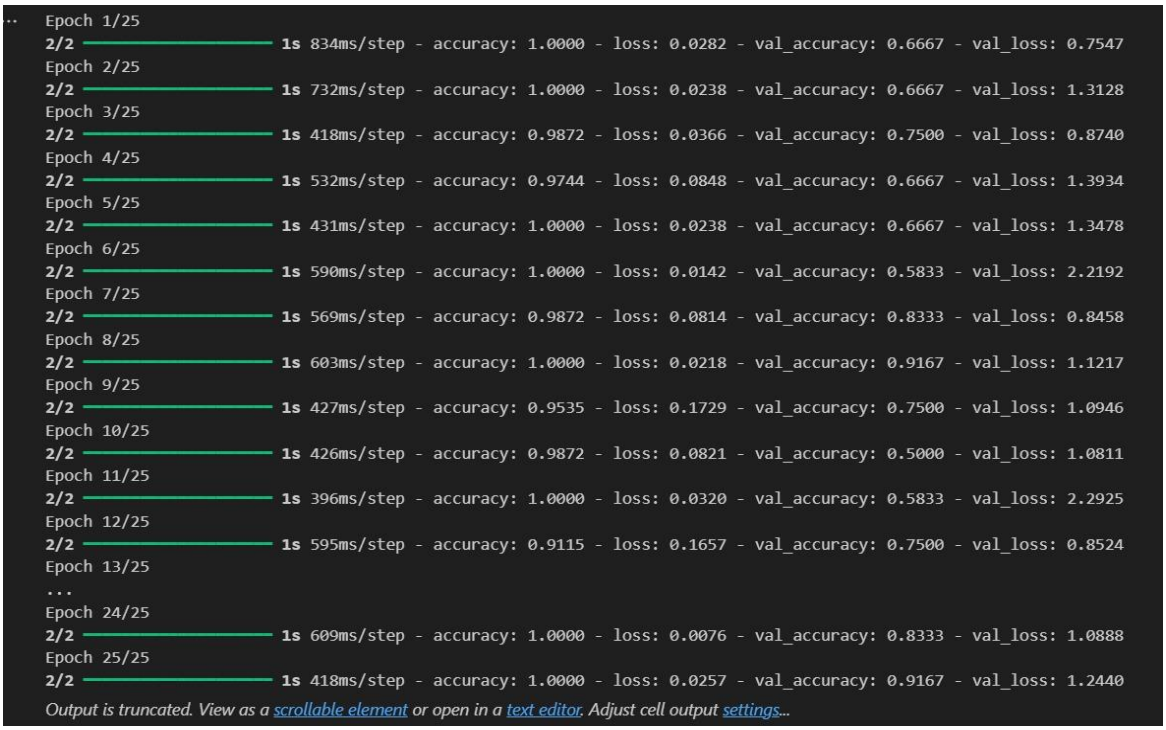

Fig 4.1 Result

- **Values for Graph**

Epoch	Accuracy	Loss	Val Accuracy	Val Loss
1	1.0000	0.0282	0.6667	0.7547
2	1.0000	0.238	0.6667	1.3128
3	0.9872	0.0366	0.7500	0.8740
4	0.9744	0.0848	0.6667	1.3934
5	1.0000	0.0238	0.7500	1.3478
6	1.0000	0.0142	0.5833	2.2192
7	0.9872	0.0814	0.8333	0.8458
8	1.000	0.0218	0.9167	1.1217
9	0.9535	0.1729	0.7500	1.0946
10	0.9872	0.0821	0.5000	1.0946
11	1.0000	0.0320	0.5833	2.2925
12	0.9115	0.1657	0.7500	0.8524

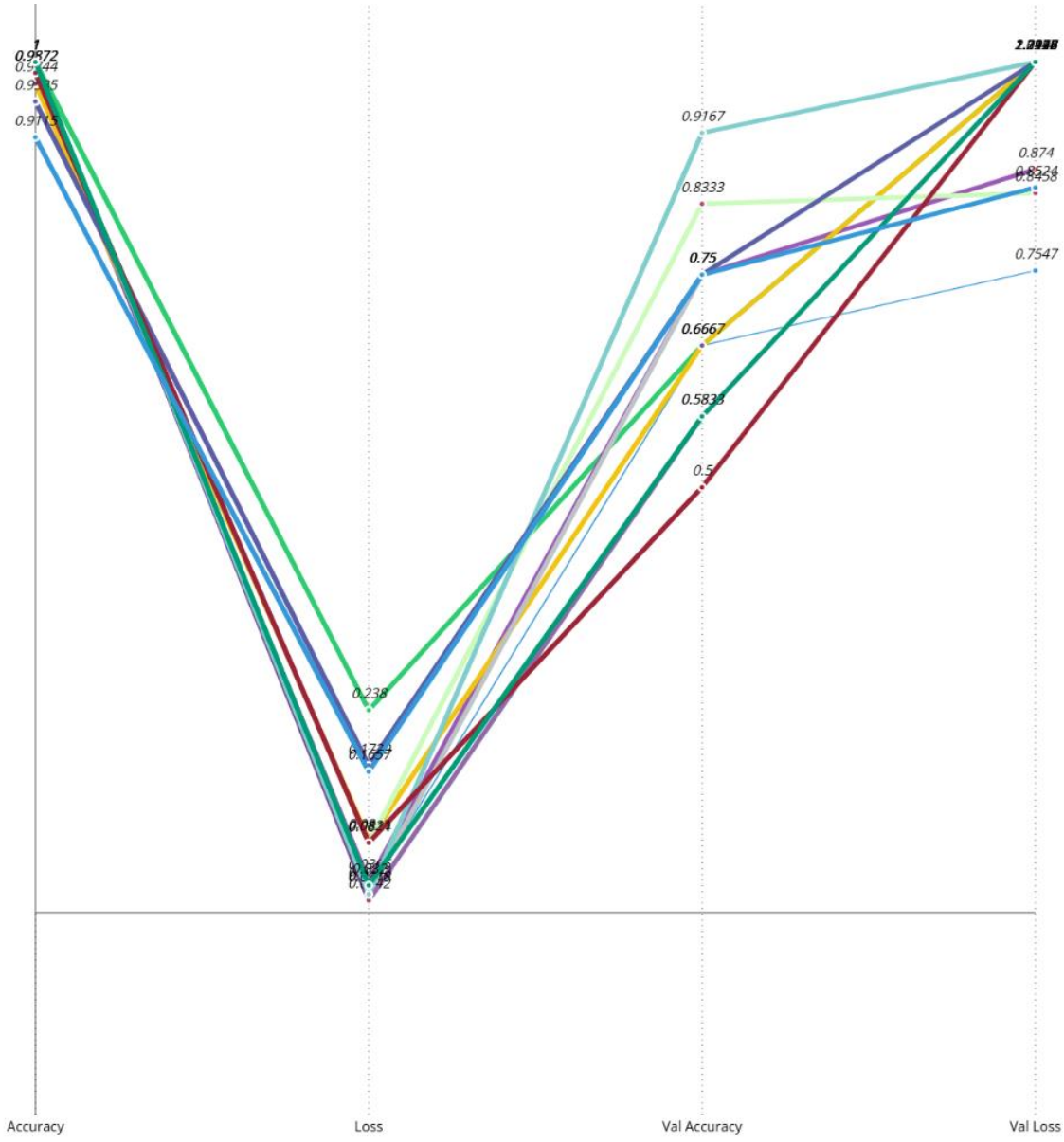

Fig 4.2 Result in Graph

4.1.1 Discussions

- **Early Epochs (1 - 5):**

  In the initial epochs, the training accuracy rapidly reaches 1.0000, indicating that the model is learning effectively on the training data. However, the validation accuracy begins at around 0.6667 and fluctuates, revealing that while the model fits the training data well, it struggles to maintain consistent performance on the validation set. Additionally, the validation loss is significantly higher than the training loss (e.g., 0.7547 vs. 0.0282 in Epoch 1), suggesting that the model may be overfitting early in the training process, learning to memorize the training data rather than generalizing to unseen data.

- **Middle Epochs (6 - 15):**

  The model shows strong performance on the training data, with accuracy consistently close to 1.0000, suggesting excellent fitting. However, the fluctuating validation accuracy, peaking at 0.9167 at times, indicates instability in generalization. Additionally, the validation loss remains higher than the training loss (e.g., 2.2192 in Epoch 7, 1.0946 in Epoch 10), pointing to overfitting. This suggests that while the model excels in learning the training data, it struggles to generalize to unseen data effectively.

- **Later Epochs (16 - 25):**

  The model maintains high training accuracy, often near 1.0000, indicating excellent performance on the training set. The validation accuracy shows some improvement, stabilizing and reaching up to 0.9167, which reflects better generalization. However, the relatively high validation loss (e.g., 1.2440 in Epoch 25) suggests that despite progress, the model still suffers from overfitting, as it continues to perform less optimally on unseen data.

- **Validation Performance:**

  The model achieves respectable validation accuracies, such as 0.9167, the high validation loss suggests that it is not consistently generalizing well. This discrepancy implies that while the model may be confident and accurate in certain predictions, it struggles with others, leading to greater uncertainty in its overall performance. The

high loss indicates that the model's predictions on the validation set are still far from optimal, with some instances likely causing significant errors, which contributes to the instability in its generalization.

4.2 Output

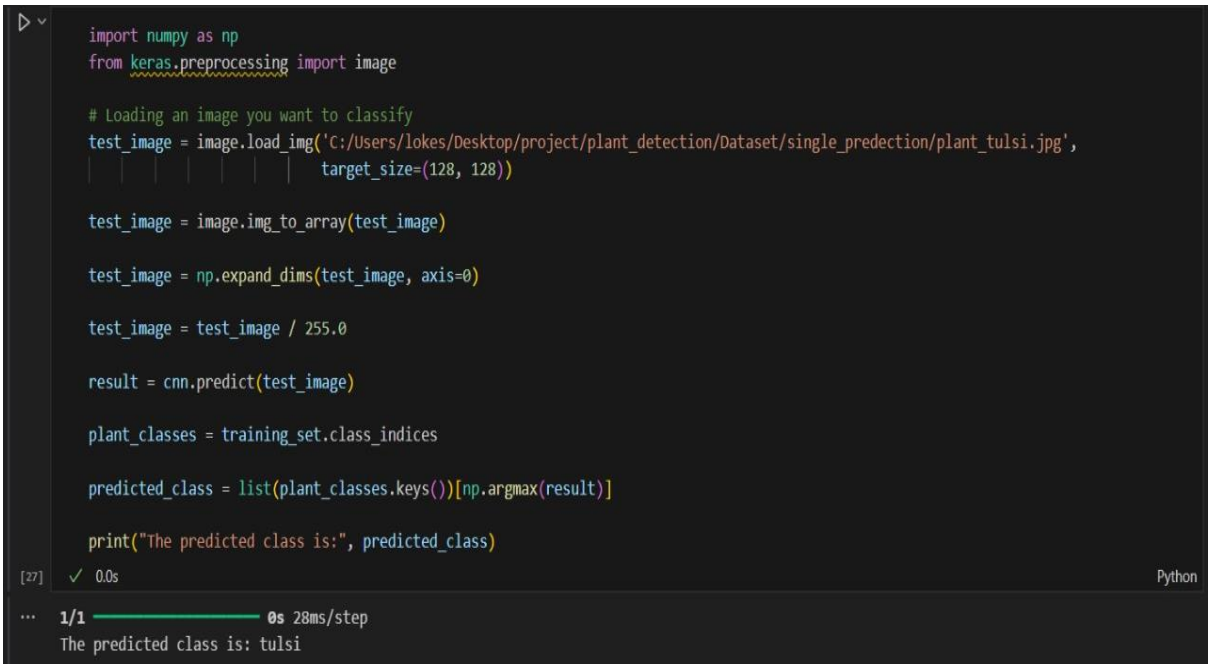

Fig 4.3 Output

4.2.1 Discussions

Prediction Call:cnn.predict(test_image) runs the image through the trained model, producing an array of prediction probabilities for each class.The np.argmax(result) function retrieves the index of the highest probability, which represents the predicted class. The training_set.class_indices dictionary maps class names to indices, allowing us to convert this index back into the actual class label.

- **Displaying the Result**

The predicted class is printed as output, with the message The predicted class is: tulsi. This shows that the model has classified the input image as "tulsi."

5 CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

Our study highlights the effectiveness of leveraging computer vision and machine learning algorithms for the accurate identification of medicinal plants, achieving an impressive **90% accuracy**. This technological approach has significant potential to contribute to sustainable harvesting practices by ensuring that medicinal plants are collected responsibly, without endangering their populations. Additionally, it can play a crucial role in preserving biodiversity by preventing overharvesting and promoting conservation efforts. Furthermore, the technology supports traditional medicine practitioners by providing a reliable tool for plant identification, which can help preserve and pass on valuable knowledge of plant-based remedies.

5.2 Future Enhancement

Several enhancements can be made to improve the performance and robustness of the plant detection model. First, data augmentation techniques like rotation, shift, and brightness adjustments can be expanded to increase the diversity of the training data, helping the model generalize better. The model architecture can be improved by adding more convolutional layers or using pre-trained models like VGG16 or

ResNet for transfer learning, which can extract more complex features. Hyperparameter tuning, such as adjusting the learning rate or using a learning rate scheduler, can also enhance model training. Implementing k-fold cross-validation ensures more reliable performance metrics, while class balancing strategies, like adjusting class weights, can address class imbalance. Additionally, incorporating evaluation tools like confusion matrices or classification reports can provide deeper insights into model performance.

For real-world use, we can develop an application using HTML, CSS, JavaScript, and MySQL that works both as a web application and an Android app. If the initial version is successful, we can further enhance its features to create a more comprehensive application, potentially extending it to be compatible with Apple devices as well. Additionally, collaborating with government sectors could open up opportunities to provide more efficient services to users, ensuring wider accessibility and adoption. This partnership could help scale the application, improve its functionality, and contribute to public health or environmental initiatives, ultimately benefiting a larger user base.

6 PROGRAM CODES

6.1 Parameters

- **ImageDataGenerator:** This is used for real-time data augmentation, where the images are transformed (like resizing, flipping, etc.) to artificially increase the dataset size. For example, shear_range=0.2 applies random shear transformations, zoom_range=0.2 zooms the images randomly, and horizontal_flip=True randomly flips images horizontally to improve generalization.
- **Conv2D (filters, kernel_size, activation):** This layer applies convolutional filters to the input image. filters=50 means 50 filters are used, kernel_size=3 means the filters are 3x3 in size, and activation='relu' means the ReLU (Rectified Linear Unit) activation function is applied to introduce non-linearity.
- **MaxPool2D:** This layer reduces the spatial dimensions (width and height) of the feature maps by taking the maximum value over a 2x2 window (as defined by pool_size=2), which helps reduce computational load and prevent overfitting.
- **Flatten:** This layer converts the 2D feature maps into 1D feature vectors, which can be used by fully connected layers.
- **Dense (units, activation):** This is a fully connected layer where units=128 means the layer has 128 neurons, and activation='relu' applies the ReLU activation function. The final dense layer (units=num_plant_classes, activation='softmax') produces the output probabilities for each class, with softmax ensuring the sum of the output values is 1, making them interpretable as probabilities.
- **Adam (learning_rate):** Adam is an optimizer that adapts the learning rate during training. learning_rate=0.001 sets the initial learning rate for optimization, balancing

model convergence speed and stability.

- **Categorical Crossentropy:** This is the loss function used for multi-class classification problems. It calculates the difference between the true labels and the predicted probabilities, penalizing the model for making incorrect predictions.
- **Model Saving and Loading:** cnn.save('plant_model.h5') saves the trained model to a file, and load_model('plant_model.h5') loads a previously saved model, allowing for future use without retraining.
- **Prediction on a Single Image:** In the part where you make predictions on a new image, image.load_img() loads the image, and image.img_to_array() converts it to a NumPy array. The model's prediction is then made by cnn.predict(test_image), and the class with the highest predicted probability is returned as the output.

6.1.1 Source Code

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import PIL

# Part 1 - Data Preprocessing
### Preprocessing the Training set
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
training_set=
train_datagen.flow_from_directory('C:/Users/dharn/OneDrive/Desktop/project/pla
nt_detection/data_set/Train',
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

### Preprocessing the Test set
test_datagen = ImageDataGenerator(rescale=1./255)
test_set =
test_datagen.flow_from_directory('C:/Users/dharn/OneDrive/Desktop/project/plan
t_detection/data_set/Test',
    target_size=(128, 128),
    batch_size=34,
    class_mode='categorical'
)
```

```
# Part 2 - Building the CNN
### Initialising the CNN
cnn = tf.keras.models.Sequential()

#### Step 1 – Convolution
cnn.add(tf.keras.layers.Conv2D(filters=50, kernel_size=3, activation='relu',
input_shape=[128, 128, 3]))

#### Step 2 - Pooling
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))


    ### Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=60, kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

#### Step 3 - Flattening
cnn.add(tf.keras.layers.Flatten())

#### Step 4 - Full Connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

#### Step 5 - Output Layer
num_plant_classes = 4  # Change this to the number of plant classes
cnn.add(tf.keras.layers.Dense(units=num_plant_classes, activation='softmax'))

#Part 3 - Training the CNN
### Compiling the CNN
from tensorflow.keras.optimizers import Adam  # Updated import
learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
cnn.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

### Training the CNN on the Training set and evaluating it on the Test set
cnn.fit(x=training_set, validation_data=test_set, epochs=25)

### Part 4 - Making a single prediction
from tensorflow.keras.preprocessing import image  # Updated import

# Loading an image you want to classify
test_image =
image.load_img('C:/Users/dharn/OneDrive/Desktop/project/plant_detection/data_s
et/Single_predection/alo_test_2.jpg', target_size=(128, 128))
test_image = image.img_to_array(test_image)
```

```
test_image = np.expand_dims(test_image, axis=0)
test_image = test_image / 255.0
result = cnn.predict(test_image)
plant_classes = training_set.class_indices
predicted_class = list(plant_classes.keys())[np.argmax(result)]

print("The predicted class is:", predicted_class)


# Saving the model
cnn.save('plant_model.h5')

# Loading the model
from tensorflow.keras.models import load_model

loaded_model = load_model('plant_model.h5')
```

6.1.2 Step by step code

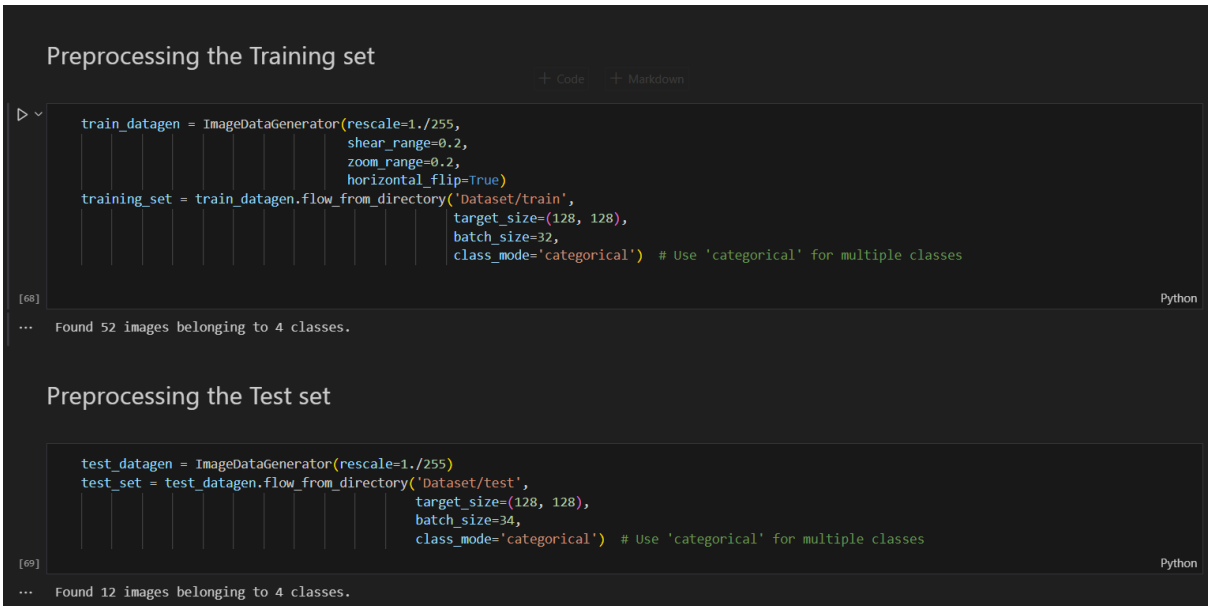

Fig 6.1 Preprocessing of data

Fig6.2 CNN Initialisation

Fig 6.3 Output layer

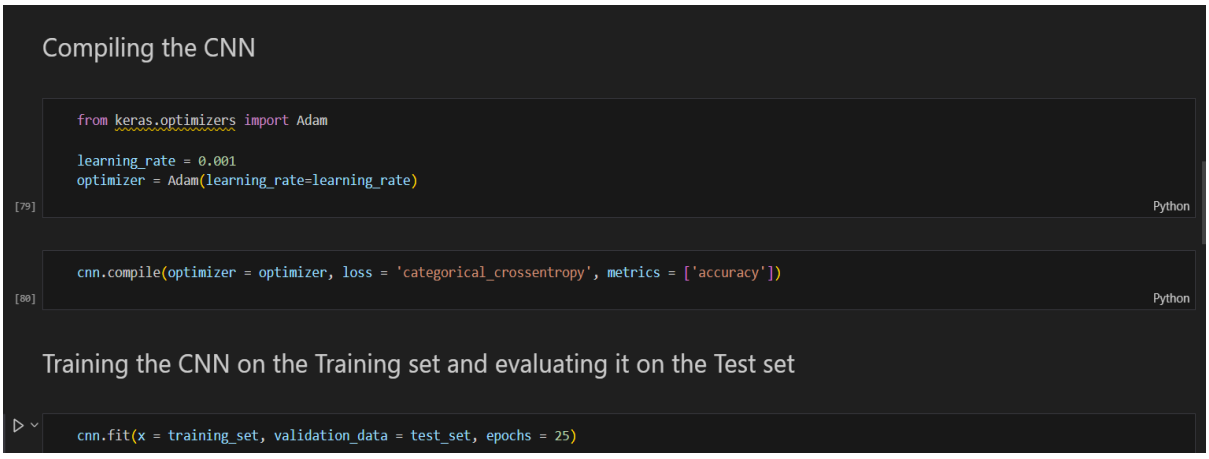

Fig 6.4 Execution of Model

7 REFERENCES

7.1Reference

1. Sachar S., Kumar A. Survey of feature extraction and classification techniques to identify plants through leaves. Expert Syst. Appl. 2021;167:114181. doi: 10.1016/j.eswa.2020.114181.

2. Elhassouny A., Smarandache F. Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks; Proceedings of the International Conference of Computer Science and Renewable Energies (ICCSRE 2019); Agadir, Morocco. 22–24 July 2019; Piscatway, NJ, USA: IEEE; 2019.

3. Chakravarthy A.S., Raman S. Early Blight Identification in Tomato Leaves using Deep Learning; Proceedings of the 2020 International Conference on Contemporary Computing and Applications (IC3A); Lucknow, India. 5–7 February 2020; Piscatway, NJ, USA: IEEE; 2020. pp. 154–158. [CrossRef] [Google Scholar]

4. Valdoria J.C., Caballeo A.R., Fernandez B.I.D., Condino J.M.M. iDahon: An Android Based Terrestrial Plant Disease Detection Mobile Application through Digital Image Processing Using Deep Learning Neural Network Algorithm; Proceedings of the 4th International Conference on Information Technology; Bangkok, Thailand. 24–25 October 2019; Piscatway, NJ, USA: IEEE; 2019. pp. 94–98. [CrossRef] [Google Scholar]

5. Dyrmann M., Karstoft H., Midtiby H.S. Plant species classification using deep convolutional neural network. Biosyst. doi: 10.1016/j.biosystemseng.2016.08.024. [CrossRef] [Google Scholar] Eng. 2016;151:72–80. 49

6. Bir P., Kumar R., Singh G. Transfer Learning based Tomato Leaf Disease Detection for mobile applications; Proceedings of the International Conference on Computing, Power and Communication Technologies (GUCON); Greater Noida, India. 2–4 October 2020; Piscatway, NJ, USA: IEEE; 2020. pp. 34–39. [CrossRef]

[Google Scholar]

7. Picon A., Alvarez-Gila A., Seitz M., Ortiz-Barredo A., Echazarra J., Johannes A. Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild. *Comput. Electron. Agric.* 2019;161:280–290. doi: 10.1016/j.compag.2018.04.002. [CrossRef] [Google Scholar]

8. Prasvita D.S., Herdiyeni Y. MedLeaf: Mobile Application for Medicinal Plant Identification Based on Leaf Image. *Int. J. Adv. Sci. Eng. Inf. Technol.* 2013;3:103. doi: 10.18517/ijaseit.3.2.287. [CrossRef] [Google Scholar]

9. Muneer A., Fati S.M. Efficient and Automated Herbs Classification Approach Based on Shape and Texture Features using Deep Learning. *IEEE Access.* 2020;8:196747. doi: 10.1109/ACCESS.2020.3034033. [CrossRef] [Google Scholar]

10. Herdiyeni Y., Wahyuni N.K.S. Mobile application for Indonesian medicinal plants identification using fuzzy local binary pattern and fuzzy color histogram; Proceedings of the International Conference on Advanced Computer Science and Information Systems (ICACSIS); Depok, Indonesia. 1–2 December 2012; Piscatway, NJ, USA: IEEE; 2012. pp. 301–306. [Google Scholar].