A PROJECT REPORT

On

Leaf Disease Detection using CNN Model With Binary Classification

Submitted by

Student names: Nanday Das, Diptanu Debnath, Srija Ghosh

Student ID No's.: 20IUT0030009, 20IUT0030019, 20IUT0030046

in partial fulfilment for the award of the degree of

BCA

Under the Guidance of

Guide Name: Prof. Arunangshu Pal

Designation: Assistant Professor

The ICFAI University, Tripura

DEPARTMENT OF COMPUTER APPLICATION

ICFAI Technical School

Faculty of Science and Technology

Course Code: BCA308

Course Title: IT Project

May,2023

The ICFAI University, Tripura

DEPARTMENT OF COMPUTER APPLICATION

Declaration of Student

I hereby declare that the project entitled “Leaf Disease Detection using CNN model with Binary Classification” submitted for the BCA degree is our original work suberised under Prof. Arunangshu Pal and the project has not formed the basis for the award of any other degree, diploma, fellowship, or any other similar titles.

Sl. No.	Student Name	Enrolment No.	Program	Signature
1	Nanday Das	20IUT0030009	BCA	
2	Diptanu Debnath	20IUT0030019	BCA	
3	Srija Ghosh	20IUT0030049	BCA	

The ICFAI University, Tripura

DEPARTMENT OF COMPUTER APPLICATION

CERTIFICATE

This is to certify that the project titled "Leaf disease detection using CNN model with binary classification" is the bona fide work carried out by Srija Ghosh, Nanday Das and Diptanu Debnath, students of BCA of ICFAI University, Tripura during the academic year 2023, in partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Application and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Signature	**Signature**
Prof. Arunangshu Pal	**Dr Subhrajyoti Deb**
Faculty Guide, Assistant Professor	**HOD, Computer Application**

Signature

Dr Prof. Prasanta Kumar Sinha

Principal, ICFAI Technical School

ABSTRACT

Agriculture is the most important sector of our Economy. In the field of modern agriculture, plant disease detection plays a vital role in improving crop productivity. To increase the yield on a large scale, it is necessary to predict the onset of the disease and give advice to farmers. Previous methods for detecting plant diseases rely on manual feature extraction, which is more expensive and time-consuming. Therefore, image-based techniques are gaining interest in the research area of plant disease detection. However, existing methods have several problems due to the improper nature of the captured image, including improper background conditions that lead to occlusion, illumination, orientation, and size. Also, cost complexity, misclassifications, and overfitting problems occur in several real-time applications.

CNN (Convolutional Neural Network) is the solution for leaf disease detection and classification. The main aim of this research is to detect whether a given leaf image is healthy or diseased. We built a simple CNN model with deep-learning networks to detect plant diseases. Here, the model is a deep convolutional neural network for the prediction of plant diseases that were previously trained for the distinctive dataset. Plant leaf disease detection has a wide range of applications available in various fields such as Biological Research and Agriculture Institutes. Plant leaf disease detection is one of the required research topics as it may prove beneficial in monitoring large fields of crops, and thus automatically detect the symptoms of diseases as soon as they appear on plant leaves.

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to Honorable Pro-VC Sir, Prof.(Dr.) Biplab Halder, Honorable Register sir, Dr A. Ranganath, Dean sir, Dr P.R. Borthakur, Principal sir, Dr Prasanta Kumar Sinha, HOD sir, Dr. Subhrajyoti Deb,and the esteemed faculty members from FST ,as well as our guide Sir, Prof. Arunangshu Pal sir, for giving us the golden opportunity to do this wonderful project titled "Leaf disease detection using CNN model with binary classification", which helped us to learn new techniques and enabled us to do lots of research and gather valuable knowledge. We are very thankful for their guidance, support and encouragements.

In addition, we would like to extend our sincere thanks to our team members, who generously shared their time, insights, and hard work with us in finalizing and completing this project within the limited time frame.

Their willingness to engage with our research was essential to the success of this project, and we are deeply grateful for their participation.

Finally, we would also like to thank our families and friends for their constant source of support and encouragement throughout this project.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Leaf diseases can cause significant damage to crop, leading to reduced yield and economic losses for farmers. Early detection and accurate classification of these diseases are crucial for timely intervention and effective management. In recent years, deep learning techniques, specifically Convolutional Neural Networks (CNNs), have demonstrated remarkable success in various computer vision tasks, including image classification. [1]

This project aims to leverage the power of CNNs to develop a binary classification model for leaf disease detection.

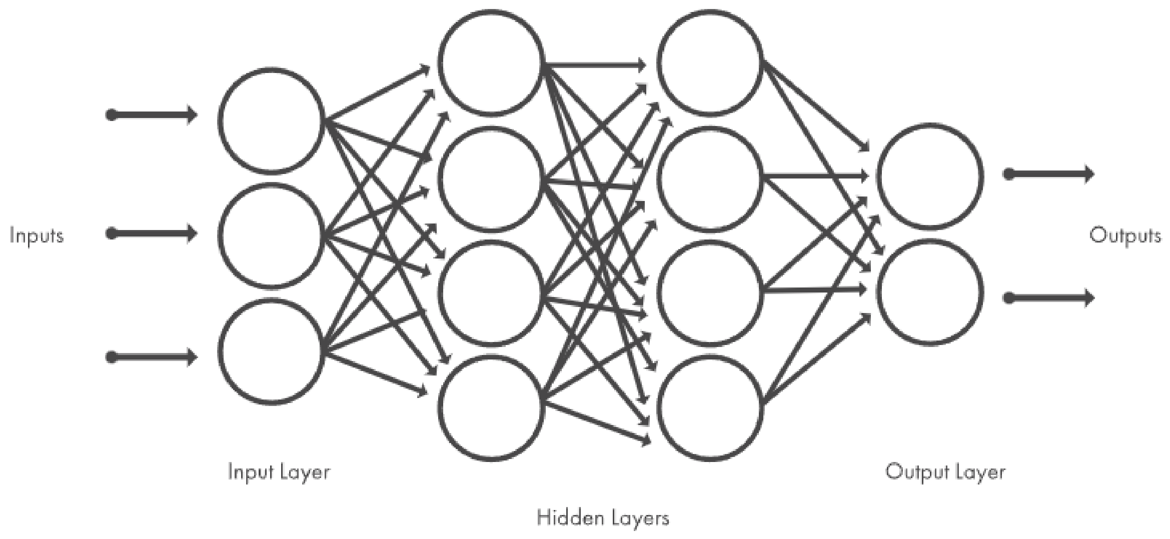


Figure 1.1: CNN Workflow

A Convolutional Neural Network (CNN) is a type of artificial neural network that is commonly used in image recognition and classification tasks. CNNs are designed to process input data in a way that mimics the way the human brain processes visual information. The key components of a CNN are convolutional layers, pooling layers, and fully connected layer. Convolutional layers apply filters to the input image to extract features that are relevant to the task at hand. Pooling layers then down sample the output of the convolutional layers to reduce the spatial dimensions of the data, while retaining the most important features. Finally, fully connected layers perform the classification or regression task based on the extracted features.

Deep learning approaches have been successfully used for automated image recognition and classification, including object recognition, in the past few years (Everingham et al., 2010) [1]

Krizhevsky et al., 2012) [2] , (Simonyan and Zisserman, 2014) [3], (Zeiler and Fergus, 2014) [4] , (He et al., 2015) [5], (Szegedy et al., 2015) [6], (Russakovsky et al., 2015) [7].

Deep convolutional neural networks (CNNs) have been used to classify images with high accuracy rates, making them suitable for use in smartphone applications (Krizhevsky et al., 2012). For example, the PlantVillage, [8] project has made available 54,306 images of 14 crop species with 26 diseases (or healthy) for deep learning approaches to be tested on (Hughes and Salathé, 2015). [9]

Neural networks provide a mapping between an input and an output, with deep neural networks mapping the input layer to the output layer over a series of stacked layers of nodes. These networks can be trained to classify images with high accuracy rates, making them ideal for use in automated disease diagnosis systems. Despite the potential benefits, there are still challenges that need to be addressed, such as ensuring that the technology is accessible and affordable for all farmers, as well as issues related to data privacy and ownership.

However, utilizing technology also comes with some difficulties, such as worries regarding security, privacy, and the potential negative effects of screen time. Individuals, organizations, and governments must use appropriate regulations and guidelines to address these issues.

In general, there are numerous advantages that technology provides to our day-to-day lives, and technological advancements have unquestionably improved our quality of life. However, it is essential to use technology in a responsible and cautious manner, as well as to continue investigating methods for maximizing its benefits while minimizing its drawbacks.

1.1 PROJECT OVERVIEW

Deep learning is a subset of machine learning that uses artificial neural networks with multiple layers to model and solve complex problems. It allows for automatic feature learning, making it highly flexible and applicable to a wide range of problems.

Deep learning has revolutionized the way we solve complex problems, enabling us to achieve higher levels of accuracy and efficiency than ever before. has revolutionized the field of artificial intelligence and has had a significant impact on many areas of research and industry.

Deep learning techniques have been widely used for leaf disease detection. Convolutional Neural Networks (CNNs) are a popular type of deep learning model for image classification tasks, including leaf disease detection.

Leaf disease detection using deep learning is a task of identifying the presence of diseases on plant leaves by analysing images of the leaves using deep learning techniques. The objective of this task is to develop a model that can accurately and efficiently classify healthy and diseased leaves. The model should be able to generalize well on new, unseen images of leaves from different plants and environments.

In our project, we are using the knowledge of deep learning to develop a efficient and sturdy system for detecting leaf disease in crops. We have researched workings of a variety of models to gather knowledge how the models work before making our own model for detecting leaf diseases.

One of the key challenges we faced in this project is developing our model with large image dataset. Since we did not have high performing machines for developing our model using deep learning, the success of our project depended on the availability of relevant datasets and computing resources. We have taken into consideration the limited resources available for our project, such as the limited GPU and other hardware resources to build our model.

In conclusion, our project is a demonstration of how deep learning helps in solving and addressing complex problems and its power of metamorphosing the field of agriculture.

1.2 PROBLEM DEFINATION

Agriculture is the primary and essential source of furnishing national income based on a country's quality and quantity of products, especially crops and plants. According to the report of the Ministry of Agriculture and Farmers Welfare (MAFW) community, 58% of people primarily depend on agriculture for their occupation, mainly in India. Natural disorder factors such as pests, weeds, and diseases account for 15%–25% of crop production losses. [10] Monitoring of disease plays a significant part in the successful development of crops on the farm with the support of efficient farming procedures. Farmers do not expertise in leaf disease, so they produce a lack of production. Leaf disease detection is important because profit and loss depend on production.

Several monitoring-based devices are used in plant disease detection, but this adds complexity and raises hardware costs. Hence, farmers with limited income cannot afford to buy such monitoring devices for detecting diseases. To provide a reliable treatment for the disease, the diagnosis of leaf diseases must be incorporated into the tools.

Therefore, using of deep learning techniques and image processing can be used to detect defective crops. Disease symptoms are often visible on the fruit, stem, and leaves. The plant leaf is taken into consideration for disease identification as it exhibits disease symptoms. Numerous methods have been introduced by researchers to achieve better results in identifying the types of diseases in the leaves.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING SYSTEM

Existing work related to leaf disease detection using CNN to detect and classify leaf disease using image processing techniques can be demonstrated through these following steps:

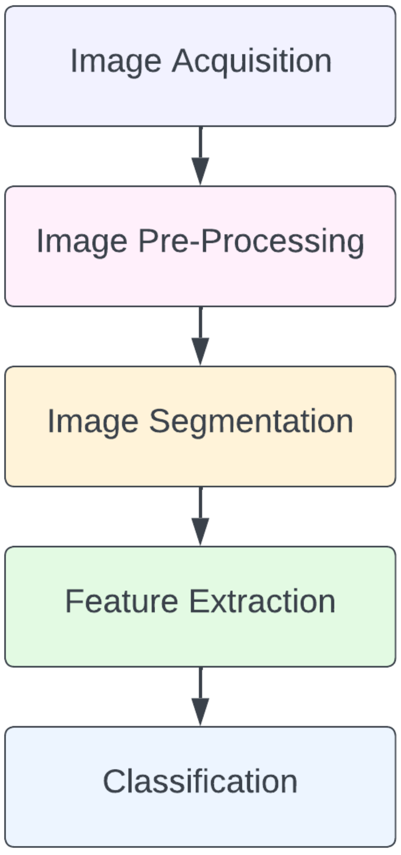


Figure 2.1: Image Classification Steps

Image Acquisition: image acquisition is the first load of the image in the digital picture process and that consists of capturing the image through a digital camera and storing it in digital media for additional MATLAB operations.

Image Pre-processing: The main aim of image pre-processing is to enhance the image information containing unwanted distortions or to reinforce some image features for any processing. The pre-processing technique uses various techniques like dynamic image size and form, filtering of noise, image conversion, enhancing image and morphological operations

Image Segmentation: In the image, segmentation is used K-means cluster technique for partitioning pictures into clusters during which a minimum of one part of the cluster contain an image with a major space of unhealthy part . The k means cluster algorithmic rule is applied to classify the objects into K variety of categories per set of features.

Feature extraction: After clusters are formed texture features are extracted using GLCM.

Classification: In classification is used for testing the leaf disease. The Random Forest classifier is used for classification.

The existing system for solving the problem of identifying plant diseases typically involves visual inspection by human experts, who can identify the signs and symptoms of a variety of diseases by examining the plants' leaves, stems, and roots. However, this process is time-consuming and can be subject to errors, as the symptoms of different diseases can overlap or be difficult to distinguish.

Another approach to the problem involves using chemical analysis to identify the presence of specific pathogens in plant tissues. This method is often more accurate than visual inspection, but it can also be expensive, time consuming, and may require specialized laboratory equipment and expertise.

However, one major drawback of existing machine learning-based approaches is their reliance on large amounts of labelled data. Creating such datasets can be time-consuming and costly, particularly in settings where resources may be limited. Additionally, many existing machine learning models are computationally intensive and may require specialized hardware to run efficiently.

In this section, a survey of various existing plant disease detection techniques along with their publications is presented to better understand their workings and flaws.

- Automated Image Capturing System for Deep Learning-based Tomato Plant Leaf Disease Detection and Recognition.
- Author: Robert G. de Luna, Elmer P. Dadios, Argel A. Bandala. Journal Name: International Conference on Advances in Big Data, Computing and Data Communication Systems
- CNN-based Leaf Disease Identification and Remedy Recommendation System. Publication Year: 2019 Author: Sunku Rohan, Triveni S Pujar, Suma VR Amog Shetty, Rishabh F Tated. Journal Name: IEEE conference paper
- Real-Time Detection of Apple Leaf Diseases Using Deep Learning Approach Based on Improved Convolution Neural Networks. Publication Year: 2019 Author: Bin Liu, Peng Jiang, Yuehan Chen, Dongjian He, Chunquan Liang. Journal Name: IEEE ACCESS
- Identification of plant leaf diseases using a nine-layer deep convolution neural network Publication Year: 2019 Author: Geetha Ramani G., Arun Pandian J. Journal Name: Computers and Electrical Engineering 76 (2019)
- A Segmentation Improved Robust PNN Model for Disease Identification in Different Leaf Publication Year: 2016 Author: Rekha Chahar, Priyanka Soni Journal Name: IEEE International Conference.
- Crop Disease Detection Using Deep Learning Publication Year: 2018 Author: Omkar Kulkarni Journal Name: IEEE Access
- Tomato Leaf Disease Detection using Convolutional Neural Networks. Publication Year: 2019 Author: Prajwala TM, Alla Pranathi, Kandiraju Sai Ashritha, Nagaratna B. Chittaragi, Shashidhar G. Koolagudi Journal Name: Proceedings of 2018 Eleventh International Conference on Contemporary Computing (IC3)

Above Information Collected From: Leaf Disease - Assignment Report, STUDOCUS. [11]

2.2 PROPOSED SYSTEM

Our project aims to detect whether a given plant leaf image is healthy or diseased. This is a binary classification task, where the output is either 0 (healthy) or 1 (diseased). A Convolutional Neural Network (CNN) model is used to classify the images. The system is written in Python and uses deep learning capabilities for image classification tasks.

The dataset used for this project consists of images of plant leaves. The Dataset is collected from the open-source site "Plant Village" and it consists of around 61,486 images of healthy and unhealthy plants divided into 39 different classes. [8]

With the help of a Python code, we have converted 39 classes into 2 classes which consist of all healthy and unhealthy plant leaf images. Then the dataset is pre-processed by resizing the images to a uniform size (256x256) and normalizing the pixel values between 0 and 1.

Images used for a whole range of training and testing were done using the ratio of 90-10(90% of the whole dataset used for the training and 10% for the testing).Our proposed system is an effective solution for detecting leaf diseases in crops using deep learning. The model developed in this project can be used as a foundation for future research and development in the field of leaf disease detection.

***Model Architecture**:*

The CNN model used for this project consists of several layers. The first layer is a 2D convolutional layer with 32 filters and a kernel size of (3,3). This is followed by a max pooling layer with a pool size of (2,2) and a dropout layer with a rate of 0.25.

The same sequence of convolutional, max pooling and dropout layers is repeated twice, with the number of filters increasing to 64 and 128 in the second and third convolutional layers respectively.

The output from the convolutional layers is flattened and passed through two dense layers with 512 and 256 neurons respectively. Each dense layer is followed by a dropout layer with a rate of 0.5. The final layer is a dense layer with a single neuron and a sigmoid activation function, which gives the binary classification output.

Training:

The model is trained using the training dataset with a batch size of 128 for 50 epochs. Early stopping is used as a call back to prevent overfitting. The Adam optimizer is used with a learning rate of 0.001. The model is compiled with binary cross-entropy loss function and accuracy as the evaluation metric.

Evaluation:

The model is evaluated using the validation dataset. The model achieves an accuracy of 0.95 on the validation dataset with a loss of 0.13. The model is then used to predict the classes of new images.

Conclusion:

The CNN model with binary classification can accurately detect whether a given plant leaf image is healthy or diseased. This model can be used in real-world applications to quickly identify diseased plants and take appropriate action to prevent the spread of disease.

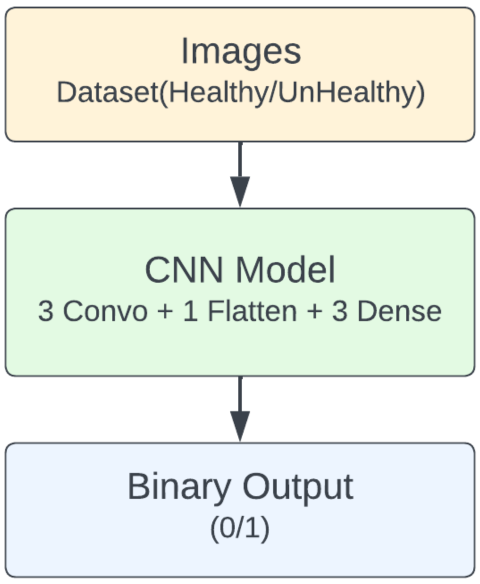


Figure 2.2: Diagrammatic Representation Of The Proposed System

2.3 FEASIBILITY STUDY

The goal of this project is to develop a deep learning model to detect and classify whether the leaf is diseased or not. The project will use a binary classification approach to distinguish between healthy and diseased leaves. The feasibility of this project will be assessed based on various factors such as dataset availability, computational resources, and potential challenges.

Dataset Availability:

One of the most important factors for the success of a deep learning project is the availability and quality of the dataset. In this case, there are several publicly available datasets for leaf disease detection, including the PlantVillage dataset and the Kaggle Plant Pathology dataset. These datasets provide a wide range of images of healthy and diseased leaves that can be used for training and testing the deep learning model. We have used the dataset from PlantVillage for our proposed system.

Computational Resources:

Deep learning models require significant computational resources to train effectively. The feasibility of this project will depend on the availability of adequate computing resources. The project can be trained using cloud-based services such as Amazon AWS or Google Cloud, which offer scalable computing resources. Alternatively, the project can be trained on a high-performance computing cluster, if available. We have used Google colab for our case since we don't have high performing machines.

One potential challenge for this project is the variability of the images due to lighting, camera angles, and other environmental factors. This variability can make it difficult for the deep learning model to accurately classify leaves. To overcome this challenge, data augmentation techniques such as rotation, scaling, and flipping can be used to increase the diversity of the training data.

Another potential challenge is the need for a large amount of training data to develop an accurate deep learning model. This can be mitigated by using pre-trained models or transfer learning techniques to leverage existing models and datasets.

Overall, the project of leaf detection using deep learning with binary classification is feasible with the availability of suitable datasets and computing resources. While there may be potential challenges, like computational resources, datasets, etc., it can be solved by careful planning and execution.

This project has the potential to develop an accurate and effective leaf detection system given the availability of limited hardware resources and a comprehensive dataset. The implementation of the models in Google colab allows for easy transfer of the models to other platforms and applications, making the project more applicable in real-world scenarios.

CHAPTER 3

EXPERIMENTAL SETUP & ANALYSIS

3.1 HARDWARE SPECIFICATION

- Processor: AMD Ryzen 7 4800H
- Ram: 16GB DDR4 3200MHz
- Graphics: GTX 1650 4GB

3.2 SOFTWARE SPECIFICATION

- Operating System: Windows 11
- Interpreter: Python 3.9
- Google Colab
- Frameworks/Modules/Tools: Tensorflow, Keras, Numpy, Matplotlib, Os, Random, Shutil

3.2.1 OPERATING SYSTEM: WINDOWS

Windows is a popular operating system developed by Microsoft. It was first released in 1985 and has since become the dominant operating system in the PC market. Windows is known for its user-friendly interface, versatility, and compatibility with a wide range of hardware and software. It is used by individuals, businesses, and organizations around the world for various purposes, including productivity, gaming, and entertainment. The latest version of Windows is Windows 11, which was released in October 2021, and it includes several new features and improvements over previous versions.

Windows has a graphical user interface (GUI) that is easy to navigate and use, making it accessible to people of all skill levels. It comes with a suite of productivity tools such as Microsoft Office, which includes Word, Excel, and PowerPoint, that are used for word processing, spreadsheets, and presentations, respectively. Windows also offers a variety of entertainment applications, such as Windows Media Player and Xbox Game Pass, making it a popular choice for gaming enthusiasts.

One of the key advantages of Windows is its customization features. Users can customize their desktops with a variety of themes, backgrounds, and icons, making it a personalized experience. Additionally, Windows has a robust security system that includes features such as Windows Defender, a built-in antivirus program, and BitLocker, which encrypts data to prevent unauthorized access. Overall, Windows is a versatile operating system that offers a range of features and benefits for both personal and professional use. It is constantly evolving, with new updates and versions released regularly to improve its functionality and performance.

3.2.2 INTERPRETER: PYTHON

Python is a high-level, interpreted programming language that is used for a wide range of applications, including web development, data analysis, artificial intelligence, machine learning, scientific computing, and automation. It was created by Guido van Rossum in 1991 and is named after the Monty Python comedy group.

Python is known for its simplicity and readability, which makes it a popular choice for beginners and experts alike. Its syntax is easy to understand and its code is often shorter than other programming languages, making it faster to develop and easier to maintain. Python also has a vast collection of libraries and modules, which allows developers to build complex applications with minimal coding effort.

One of the key advantages of Python is its versatility. It can be used for a wide range of tasks, including web development, desktop applications, scientific computing, data analysis, machine learning, and automation. This versatility, along with its simplicity and readability, has contributed to Python's popularity and widespread use in many industries.

Python is an open-source language, which means that it is free to use, distribute, and modify. It is available for Windows, Linux, and macOS operating systems, making it accessible to developers on a variety of platforms. Additionally, Python has a large and active community of developers who contribute to its development, offer support and resources, and create new libraries and tools.

Overall, Python is a powerful and versatile programming language that is widely used for a variety of applications. Its simplicity, readability, and vast collection of libraries make it a popular choice for both beginners and experts. Its open-source nature and active community ensure that it will continue to evolve and remain relevant in the future.

3.2.3 FRAMEWORKS / MODULES /TOOLS

Frameworks and libraries are essential tools for software development. They are pre-written pieces of code that help developers to perform common tasks quickly and efficiently. These tools are designed to provide developers with the building blocks they need to create high-quality software applications. In this project, we are using several frameworks and libraries to develop our deep learning model. These include TensorFlow, NumPy, Matplotlib, Keras Each of these tools provides specific functionality that helps us to train and evaluate our model.

3.2.3.1 *TENSORFLOW*

TensorFlow is an open-source software library for numerical computation using data flow graphs. Nodes inside the graph represent mathematical formulas, whereas the graph edges represent the multidimensional knowledge arrays (tensors) communicated between them. The versatile architecture permits you to deploy computation to at least one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. Tensor Flow was originally developed by researchers and engineers acting on the Google Brain Team at intervals in Google's Machine Intelligence analysis organization for the needs of conducting machine learning and deep neural networks research, however, the system is general enough to be applicable in a wide range of alternative domains as well. [12]

3.2.3.2 *KERAS*

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with attention to enabling quick experimentation. Having the ability to travel from plan to result with the smallest amount of doable delay is key to doing great research. Keras permits straightforward and quick prototyping (through user-friendliness, modularity, and extensibility). Supports each convolutional network and recurrent network, furthermore as combinations of the two Run seamlessly on CPU and GPU. The library contains numerous implementations of usually used neural network building blocks like layers, objectives, activation functions, optimizers, and several tools to create operating with image and text data easier. The code is hosted on GitHub, and community support forums embody the GitHub issues page, a Glitter channel and a Slack channel. [13]

3.2.3.3 *NUMPY*

NumPy is a Python library that provides scientific and higher-level mathematical abstractions wrapped in Python. It is the core library for scientific computing, that contains a strong n-dimensional array object and provides tools for performing operations on arrays and matrices. It is additionally useful in linear algebra, random number capability etc.

NumPy's array type augments the Python language with an efficient data structure used for numerical work, data analysis, machine learning, scientific computing. NumPy additionally provides basic numerical routines, like tools for locating Eigenvectors.

3.2.3.4 *MATPLOTLIB*

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. Matplotlib comes with a wide variety of plots. Plots help to understand trends, and patterns, and to make correlations. They're typically instruments for reasoning about quantitative information. The matplotlib.pyplot is the collection command style function that makes matplotlib feel like working with MATLAB. The pyplot functions are used to make some changes to the figure such as creating a figure, creating a plotting area in a figure, plotting some lines in a plotting area, decorating the plot including labels, etc. It is good to use when we want to plot something quickly without instantiating any figures or Axes. While working with matplotlib.pyplot, some states are stored across function calls so that it keeps track of the things like current figure and plotting area, and these plotting functions are directed to the current axes. The pyplot module provides the plot() function which is frequently used to plot a graph.

3.2.3.5 *GOOGLE COLAB*

Colab is a free notebook environment that runs entirely in the cloud. It lets you and your team members edit documents, the way you work with Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook. Google is quite aggressive in AI research. Over many years, Google developed an AI framework called TensorFlow and a development tool called Collaboratory.

Today TensorFlow is open-sourced and since 2017, Google made Collaboratory free for public use. Collaboratory is now known as Google Colab or simply Colab. Another attractive feature that Google offers to developers is the use of GPU. Colab supports GPU and it is free. The reasons for making it free for the public could be to make its software a standard in academics for teaching machine learning and data science. It may also have a long-term perspective of building a customer base for Google Cloud APIs which are sold per-use basis. Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

3.2.3.6 *OS*

The os module in Python provides functions for interacting with the operating system. It offers a wide range of capabilities to perform various operating system-related tasks.

The os module provides a convenient way to interact with the underlying operating system, making it easier to perform tasks related to file management, process handling, and environment variables. It is a fundamental module in Python for operating system-related operations.

3.2.3.7 *RANDOM*

The random library in Python is used for generating random numbers and making random selections. It provides functions to generate random floating-point numbers, random integers within a range, and random selections from sequences. You can also shuffle the elements of a sequence or take a random sample from a population.

The library supports generating randomness using seeds, allowing you to initialize the random number generator to produce the same sequence of random numbers. Additionally, it offers functions for generating random numbers from specific distributions, such as the Gaussian distribution.

Overall, the random library is a useful tool for tasks that require randomness, such as simulations, games, and other applications where randomization is needed.

3.2.3.8 *SHUTIL*

The shutil module in Python provides a high-level interface for working with files and directories. It offers functions for file operations, directory operations, and archiving operations.

The shutil module provides convenient functions for common file and directory operations. It abstracts away some of the lower-level complexities of file handling, making it easier to work with files and directories in Python. It is particularly useful when you need to perform tasks such as copying files, moving files, removing directories, or creating archive files.

3.3 DATASET DETAILS

An image dataset is a collection of digital images that are used for various computer vision tasks such as image classification, object detection, segmentation, and recognition. These datasets can be either manually curated or generated through automated methods.

The dataset used for this project consists of images of various plant leaves. The Dataset is collected from the open-source site "PlantVillage" and it consists of around 61,486 images of healthy and unhealthy plants divided into 39 different classes. [8]

Out of 39 classed in our project we have taken only 6 classes with the help of a Python code, we have converted 6 classes into 2 classes which consist of shuffled healthy and unhealthy plant leaf images. Then the dataset is pre-processed by resizing the images to a uniform size (256x256) and normalizing the pixel values between 0 and 1.

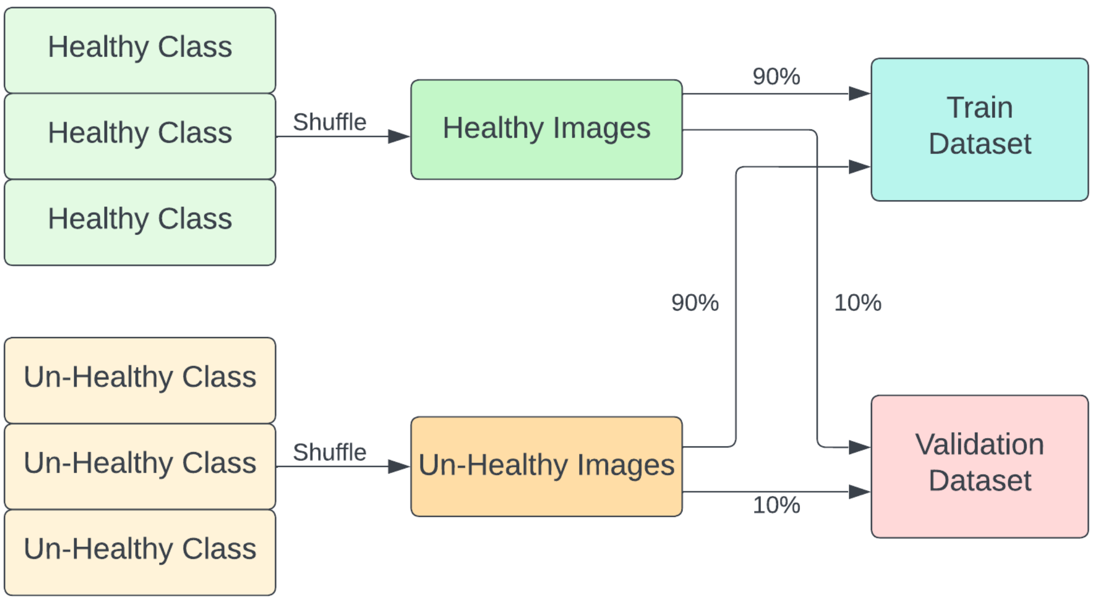


Figure 3.1: Dataset creation flowchart

Images used for a whole range of training and testing were done using the ratio of 90-10 (90% of the whole dataset used for the training and 10% for the testing).

Train Dataset: This is the group of our dataset used to train the neural network directly. Training data refers to the dataset partition exposed to the neural network during training.

Validation Dataset: This group of the dataset is utilized during training to assess the performance of the network at various iterations. We can also use this database for our future testing.

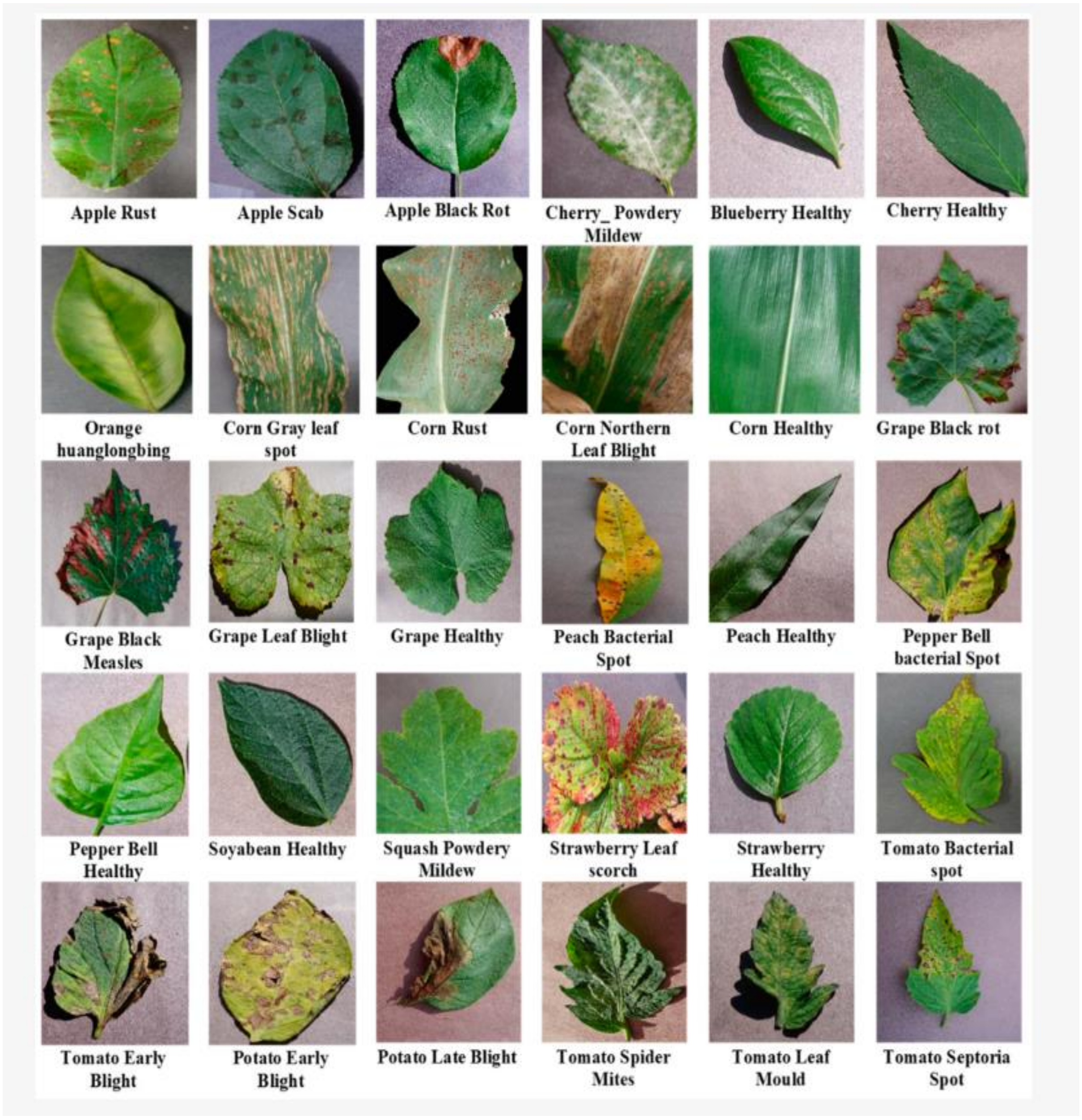


Figure 3.2: Representation of Dataset

CHAPTER 4

SYSTEM ANALYSIS & DESIGN

4.1 REQUIREMENT SPECIFICATION

The requirements can be further divided into two parts for an abstract understanding i.e., system requirement and project requirement.

Functional Requirements

- The system should be able to detect different types of leaf diseases with a high level of accuracy.
- The system should be able to train and evaluate.
- The system should be able to accept input images in various formats and sizes.
- The system should be able to give the output of the classification results for each image as well as visual representations of the classification.
- The system should be able to handle large datasets for training and testing.

Non-Functional Requirements

- The system should have a user-friendly interface for inputting and displaying images and results.
- The system should be efficient and not take an excessive amount of time to train and classify images.
- The system should have high availability and uptime.

Hardware and Software Requirements

- The system should be compatible with Windows operating system.
- The system should require a minimum of 8GB of RAM and 1GB of free disk space and a minimum of 4GB graphics.
- The system should use GPU for training deep learning models.
- The system should require Python 3.6 or above and the necessary libraries, including TensorFlow, Keras, NumPy, Matplotlib.

Performance Requirements

- The system should achieve an accuracy of at least 90% in classifying leaf diseases.
- The system should be able to train model in a reasonable amount of time, not taking longer than 24 hours.
- The system should be able to classify images within a reasonable amount of time, with no individual image taking longer than 5 minutes to classify.

4.2 PROJECT STEPS

We divided the whole coding part into some parts to make it easy to understand and modify later. These steps are elaborated below with all code explanations.

PREREQUISITES OF THE MODEL

This code is a Python script for running a deep learning model using the TensorFlow library and Keras API in Google Colaboratory, a cloud-based platform for machine learning and data analysis. The code begins by importing the necessary modules and libraries, including the "drive" module from the "google.colab" library, which allows the user to mount their Google Drive to the Colaboratory notebook. This is useful for accessing and saving data files, model weights, and other resources.

The next few lines import the TensorFlow library and the Keras API, which is a high-level interface for building and training deep learning models. Specifically, the "keras" module is imported from the "tensorflow" library, which provides a set of pre-built functions and classes for constructing neural networks.

The script then defines a deep learning model using the "Sequential" class from Keras. This class allows the user to build a linear stack of layers for the neural network. In this case, the layers include 2D convolutional layers (Conv2D), max pooling layers (MaxPooling2D), fully connected layers (Dense), batch normalization layers (BatchNormalization), and dropout layers (Dropout). These layers are commonly used in image classification tasks, such as object detection, where the goal is to identify and classify objects in an image.

Finally, the code prints the version of TensorFlow that is currently being used. This is useful for debugging and ensuring that the correct version of the library is being used for the specific model being built. Overall, this code provides a simple and effective way to build and train a deep learning model for image classification tasks in Google Colaboratory

- This code is mounting the Google Drive storage to access files in Colab. “Google.colab” is a package that provides a set of tools for working with Colab, including file I/O, visualization, and more. Then the drive is imported from “google.colab” to mount the Google Drive storage.

- drive.mount('/content/drive/') is mounting the root directory of Google Drive to the local directory /content/drive/. After running this code, a link will be generated. Clicking on the link will prompt the user to sign in to their Google account and provide an authorization code. Once the authorization code is entered, the Google Drive will be mounted to the Colab notebook
- This code is importing the required libraries for building a neural network using Keras with TensorFlow as its backend.Tensorflow is being imported as tf, and keras is being imported from tensorflow. Sequential is being imported from keras to build a sequential neural network model.Dense, Conv2D, MaxPooling2D, Flatten, BatchNormalization, and Dropout are also being imported from keras.layers to define the layers of the neural network. print(tf._version_) is simply printing the version of TensorFlow that is currently installed
- This code block uses the image_dataset_from_directory() function from the keras.utils module to create two image datasets - one for training and one for validation. image_dataset_from_directory() is a utility function that reads images from a directory and creates a tf.data.Dataset object that can be used for training or evaluating a machine learning model.
- The function takes several arguments:
  1. directory: The path to the directory containing the image files.
  2. labels: Specifies how to label the data. If set to "inferred", the function will infer the labels from the subdirectory names.
  3. label_mode: Specifies the type of label data to use. If set to "int", the labels will be integers.
  4. batch_size: The number of images to include in each batch.
  5. image_size: The size to which all images will be resized.
  6. shuffle: Whether or not to shuffle the data.
- In this specific code block, the image_dataset_from_directory() function is used to create two datasets: train_ds and validation_ds. The train_ds dataset is created using the images in the directory '/content/drive/MyDrive/Dataset/train' and the validation_ds dataset is created using the images in the directory '/content/drive/MyDrive/Dataset/test'.

- Both datasets are labeled by inferring the labels from the subdirectory names and using integer labels. The batch size is set to 128, the image size is set to (256,256), and the data is being shuffled.

MAKING OF THE CNN MODEL

This code defines a deep learning model for image classification using the Keras API. The model is built using the Sequential class, which allows for a linear stack of layers. The layers that are used include convolutional layers, max pooling layers, dropout layers, and dense layers.

The first line imports the "layers" module from the Keras API. The next few lines define the layers for the model. The input layer is a convolutional layer with 32 filters, each with a kernel size of 3x3, an activation function of ReLU, and an input shape of 256x256 pixels with 3 color channels (RGB). This is followed by a max pooling layer with a pool size of 2x2 and stride of 2, which helps to reduce the spatial dimensions of the output. A dropout layer is also added with a rate of 0.25 to reduce overfitting.

The next two hidden layers are similar to the first layer, but with increasing numbers of filters (64 and 128, respectively) and without an input shape. The same max pooling and dropout layers are added after each convolutional layer.

The output from the convolutional layers is then flattened into a one-dimensional array using the Flatten layer. This allows the output to be passed through fully connected dense layers. The first dense layer has 512 units and uses the ReLU activation function, followed by a dropout layer with a rate of 0.5. The second dense layer has 256 units and also uses ReLU activation and a dropout rate of 0.5.

Finally, the output layer has a single unit with a sigmoid activation function, which is appropriate for binary classification tasks. This layer is used to predict the probability of the input image belonging to the positive class.

The code then prints a summary of the model using the summary() method, which displays the shape of each layer and the total number of parameters in the model. This can be useful for debugging and optimizing the model architecture.

- We have developed a code for building a convolutional neural network(CNN) using Keras, which is a high-level neural network API written in python. The network is designed for image classification tasks with binary output(0 or 1). Here's a breakdown of what each part of the code does:
- All the steps of our model are explained as follows:
  1. From keras import layers: This imports the layers module from keras , which provides a collection of neural network layers.
  2. Model =keras.sequential([…..]): This creates a new sequential model object, which is a linear stack of layers. The layers are added to the model in the order they are listed in the list passed to keras.Sequential(), which defined inside the square brackets.
  3. Layers.Conv2D(32,(3,3),activation = "relu",input_shape =(256,256,3)): This adds a 2D convolutional layer with 32 filters,each with a size of 3*3. The activation function used in Rectified Linear Unit(ReLU),which is a commonly used activation function in neural networks. The input_shape argument specifies the shape of the input data, which is a 256*256 RGB image.
  4. Layers.MaxPooling2D(pool_size=(2,2),strides = 2): This adds a max pooling layer, which reduces the spatial size of the output from the previous layer by taking the maximum value in each pooling region. The pool_size argument specifies the size of the pooling window, and the strides argument specifies the stride length.
  5. layers.Dropout(0.25): This adds a dropout layer, which randomly sets a fraction of the input units to 0 during training, to prevent overfitting.
  6. The next three blocks of code (lines 8-13, 15-20, and 22-27) add more convolutional, pooling, and dropout layers to the model, with increasing numbers of filters.
  7. layers.Flatten(): This adds a layer that flattens the output from the convolutional layers into a 1D array, which can be input to a fully connected layer.
  8. layers.Dense(512, activation='relu'): This adds a fully connected layer with 512 units and ReLU activation.
  9. layers.Dropout(0.5): This adds another dropout layer with a higher dropout rate.
  10. layers.Dense(256, activation='relu'): This adds another fully connected layer with 256 units and ReLU activation.
  11. layers.Dropout(0.5): This adds yet another dropout layer.

12. layers.Dense(1, activation='sigmoid'): This adds the output layer with a single neuron and sigmoid activation function, which outputs a probability value between 0 and 1.

13. We have then used keras.optimizers.RMSprop() which is an optimizer that implements the RMSprop algorithm. It takes a learning rate as input, which is set to 0.001 in this case. RMSprop is a gradient descent algorithm that helps converge to the minimum faster, especially in the presence of sparse gradients.
14. After that we implemented model.compile() which is a method that compiles the Keras model with the specified optimizer, loss function, and evaluation metric.
15. Here, the optimizer parameter is set to the RMSprop optimizer object created above, loss is set to 'binary_crossentropy', which is a binary loss function suitable for binary classification problems, and metrics is set to ['accuracy'] to evaluate the model's accuracy during training and testing.

- Overall, this model consists of several convolutional layers followed by fully connected layers, with dropout layers to prevent overfitting. The output layer uses sigmoid activation function to output a binary classification result. This code block is compiling the model with a specified optimizer, loss function and evaluation metric

TRAINING OF THE CNN MODEL

- This code imports the EarlyStopping callback from Keras, which is a function that stops the training of a neural network model when a monitored metric, such as validation loss, has stopped improving for a certain number of epochs (patience).
- Here's a breakdown of the arguments passed to EarlyStopping():
  1. monitor='val_loss': This specifies the metric to monitor for early stopping, which is the validation loss in this case.
  2. min_delta=0.00001: This sets the minimum change in the monitored metric to be considered an improvement, which is set to a small value of 0.00001 in this case.
  3. patience=10: This sets the number of epochs to wait before stopping the training if the monitored metric does not improve, which is set to 10 in this case.
  4. mode='auto': This specifies the direction of the monitored metric to be considered as an improvement, which is set to 'auto' in this case to automatically infer the direction based on the metric being monitored.
  5. verbose=1: This sets the level of verbosity during training, which is set to 1 to print a message when the training is stopped due to early stopping.
  6. baseline=None: This sets a baseline value for the monitored metric. If the monitored metric does not improve beyond this baseline value, training will stop.
  7. restore_best_weights=False: This specifies whether to restore the weights of the model to the best weights observed during training, which is set to False in this case, meaning that the final weights will be those obtained at the end of training rather than the best weights observed during training.
- Overall, the EarlyStopping callback is a useful tool for avoiding overfitting and saving computational resources by stopping the training of a neural network model before it has fully converged if there is no improvement in the monitored metric.
- This code trains a Keras model using the fit() method and passes in some arguments, including the training and validation datasets, number of epochs, and a callback for early stopping.
- Here's a breakdown of the arguments passed to fit():
  1. train_ds: This is the training dataset that is used to train the model.
  2. validation_data=validation_ds: This specifies the validation dataset to be used during training, which is used to evaluate the performance of the model after each epoch.

3. epochs=50: This sets the number of training epochs to be performed.
4. callbacks=[callback]: This specifies a list of callbacks to be used during training, which includes the EarlyStopping callback defined earlier.
5. The fit() method returns a History object that contains information about the training process, such as the training and validation loss and accuracy for each epoch. This object is assigned to the history variable in the code.

- Overall, this code trains a Keras model using the specified training and validation datasets, with early stopping used to prevent overfitting of the model on the training data.
- This code uses the matplotlib library to plot the training and validation accuracy of a Keras model that was previously trained using the fit() method.
- Here's a breakdown of the code:

1. plt.plot(history.history['accuracy'],color='red',label='Train accuracy'): This plots the training accuracy of the model on the y-axis against the number of epochs on the x-axis. The data for the plot is obtained from the history object returned by the fit() method. The color of the plot is set to red and a label of "Train accuracy" is added to the plot.
2. plt.plot(history.history['val_accuracy'],color='green',label='Validation accuracy'): This plots the validation accuracy of the model on the y-axis against the number of epochs on the x-axis. The data for the plot is obtained from the history object returned by the fit() method. The color of the plot is set to green and a label of "Validation accuracy" is added to the plot.
3. plt.legend(): This adds a legend to the plot based on the labels assigned to each plot line.
4. plt.show(): This displays the plot on the screen.

- Overall, this code is used to visualize the training and validation accuracy of a Keras model, which can be helpful in determining whether the model is overfitting or underfitting
- This code uses the matplotlib library to plot the training and validation loss of a Keras model that was previously trained using the fit() method.
- Here's a breakdown of the code:

1. plt.plot(history.history['loss'],color='red',label='Train loss'): This plots the training loss of the model on the y-axis against the number of epochs on the x-axis. The data

for the plot is obtained from the history object returned by the fit() method. The color of the plot is set to red and a label of "Train loss" is added to the plot.

2. plt.plot(history.history['val_loss'],color='green',label='Validation loss'): This plots the validation loss of the model on the y-axis against the number of epochs on the x-axis. The data for the plot is obtained from the history object returned by the fit() method. The color of the plot is set to green and a label of "Validation loss" is added to the plot.
3. plt.legend(): This adds a legend to the plot based on the labels assigned to each plot line.
4. plt.show(): This displays the plot on the screen.

- Overall, this code is used to visualize the training and validation loss of a Keras model, which can be helpful in determining whether the model is overfitting or underfitting

TESTING THE CNN MODEL

- This code uses the OpenCV and Matplotlib libraries to load an image file and pass it through a Keras model to make a prediction about its health status.
- Here's a breakdown of the code:
  1. filepath = input("Enter image file path: "): This line prompts the user to enter the file path of the image they want to test.
  2. test_img = cv2.imread(filepath): This uses the OpenCV imread() function to read in the image from the specified file path.
  3. plt.imshow(test_img): This displays the loaded image using the imshow() function from the Matplotlib library.
  4. test_img = cv2.resize(test_img,(256,256)): This resizes the loaded image to 256x256 pixels, which matches the input shape of the Keras model.
  5. test_input = test_img.reshape((1,256,256,3)): This reshapes the resized image into a 4-dimensional tensor with a shape of (1,256,256,3), which is the expected input shape of the Keras model.
  6. result = model.predict(test_input)[0][0]: This uses the trained Keras model to make a prediction on the test input, and extracts the prediction score from the resulting 2-dimensional array.
  7. if result == 0.0: ... else: ...: This code checks the prediction score and prints the predicted health status of the image to the console.
- Overall, this code can be used to quickly test the performance of a trained Keras model on a single input image. Note that for this code to work correctly, you'll need to make sure that the Keras model has been trained on a dataset that includes similar types of images to the one you're testing.

4.2.1 TRAINING PROCESS OF THE MODEL

INITIAL ANALYSIS:

Initially, we trained around 300 images with 10 epochs with a batch size of 32. We have tried and implemented various combinations to get higher accuracy.

The best accuracy that we got was 94% but the accuracy was fluctuating sometimes so we tried to train and test more images to increase the accuracy.

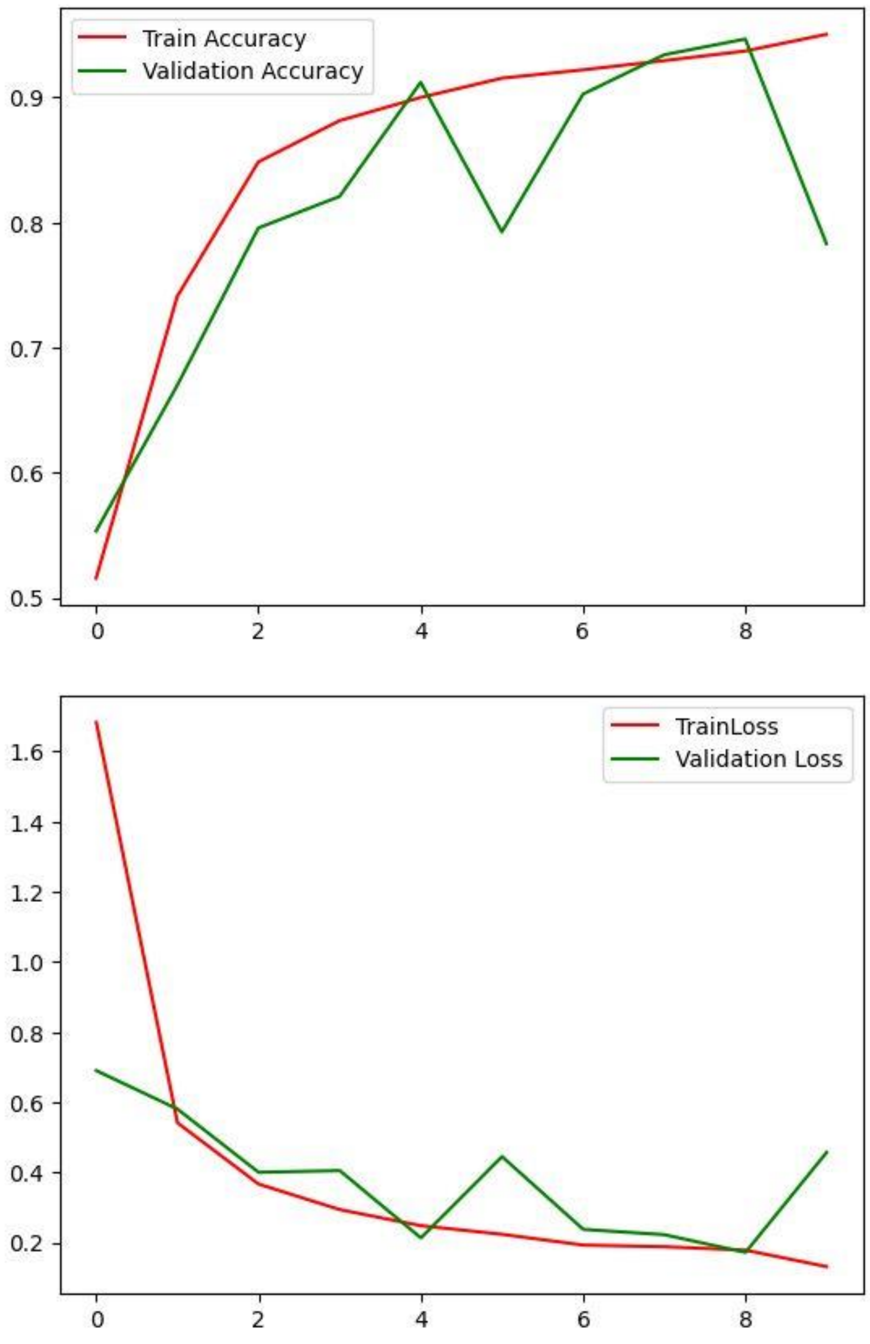


Figure 4.1: Initial Phase Training Analysis

SECOND ANALYSIS:

After the success of the initial phase, we increased the number of images to 3000 and also increased the number of epochs and batch size to 50 and 128.

We have also tried various combinations and implemented various functions and layers to check and improve accuracy. In this phase, the highest accuracy we got is 96%.

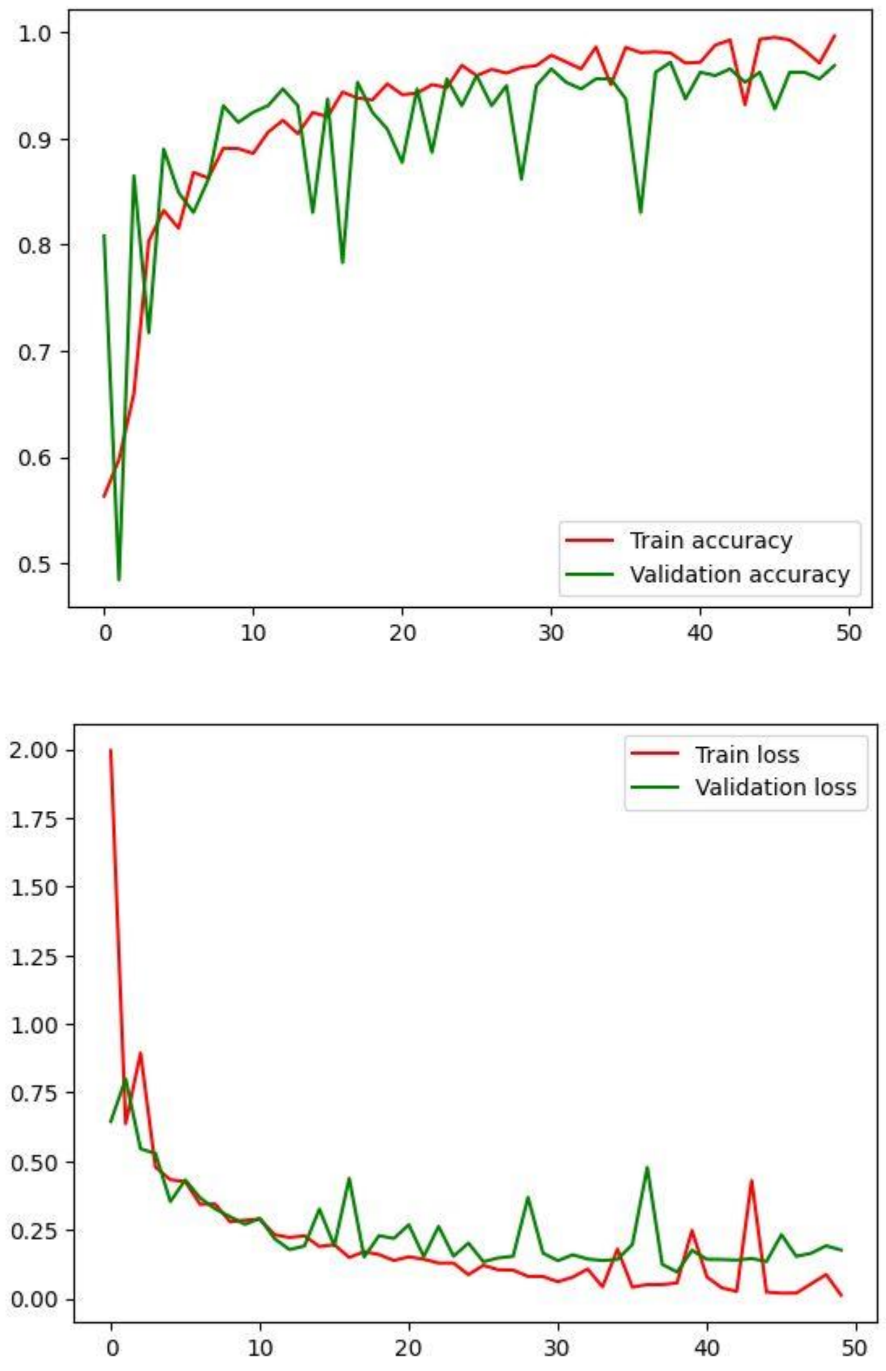


Figure 4.2: Second Phase Training Analysis

THIRD ANALYSIS:

In the latest phase of our project, we have increased the number of images to around 5000 and we have kept the number of epochs to 50 and batch size to 128 like the earlier phase.

The highest accuracy we got so far in this phase is 98%, which sometimes fluctuates.

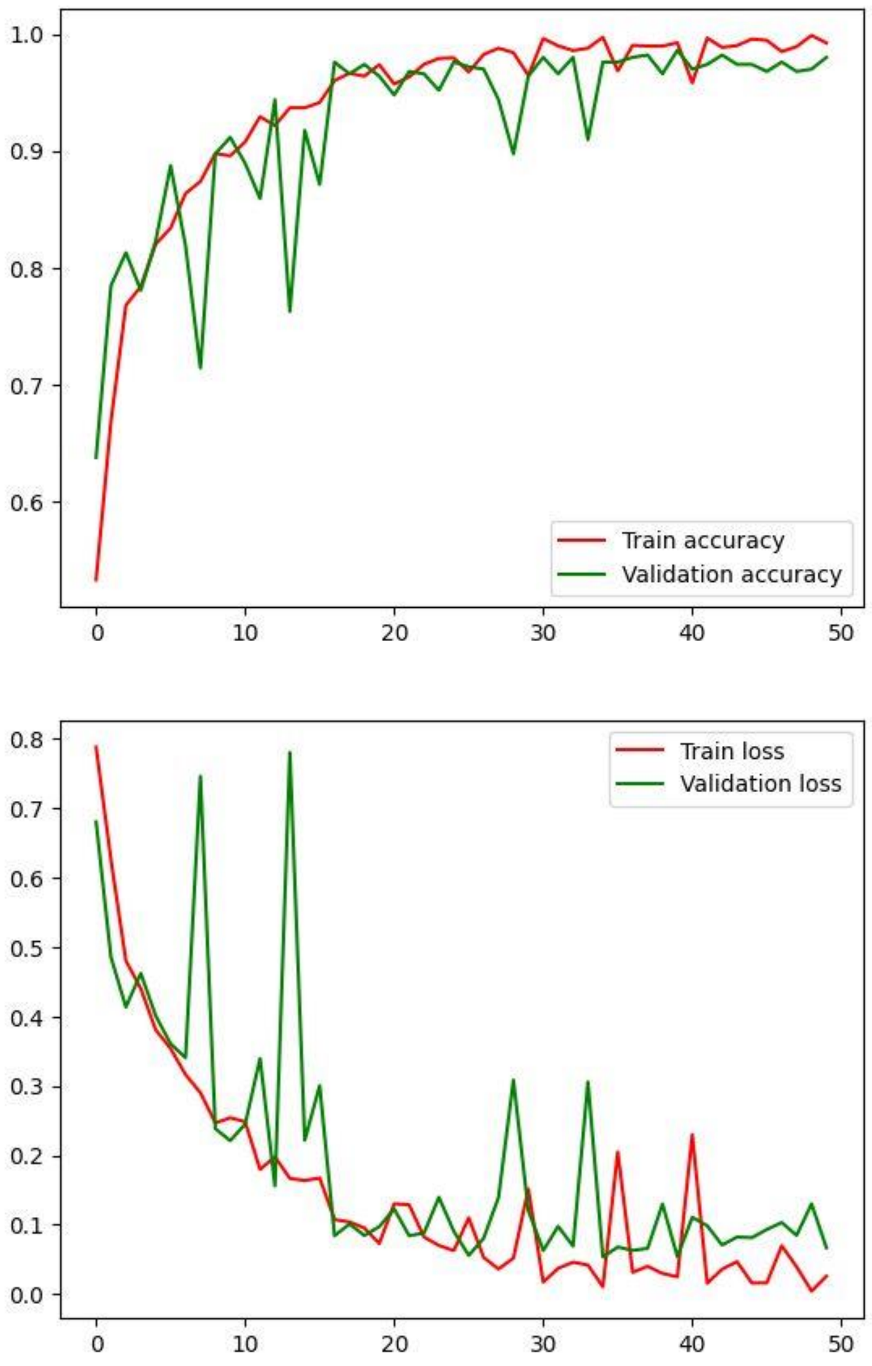


Figure 4.3: Third Phase Training Analysis

LATEST ANALYSIS:

After the success of the third phase, we increased the number of images to 8000 and also increased the number of epochs from 50 to 100 and batch size from 128 to 256.

We have also tried various combinations and implemented functions like skip connection to the layers to check and improve accuracy. In this phase, the highest accuracy we got is 98.4%.

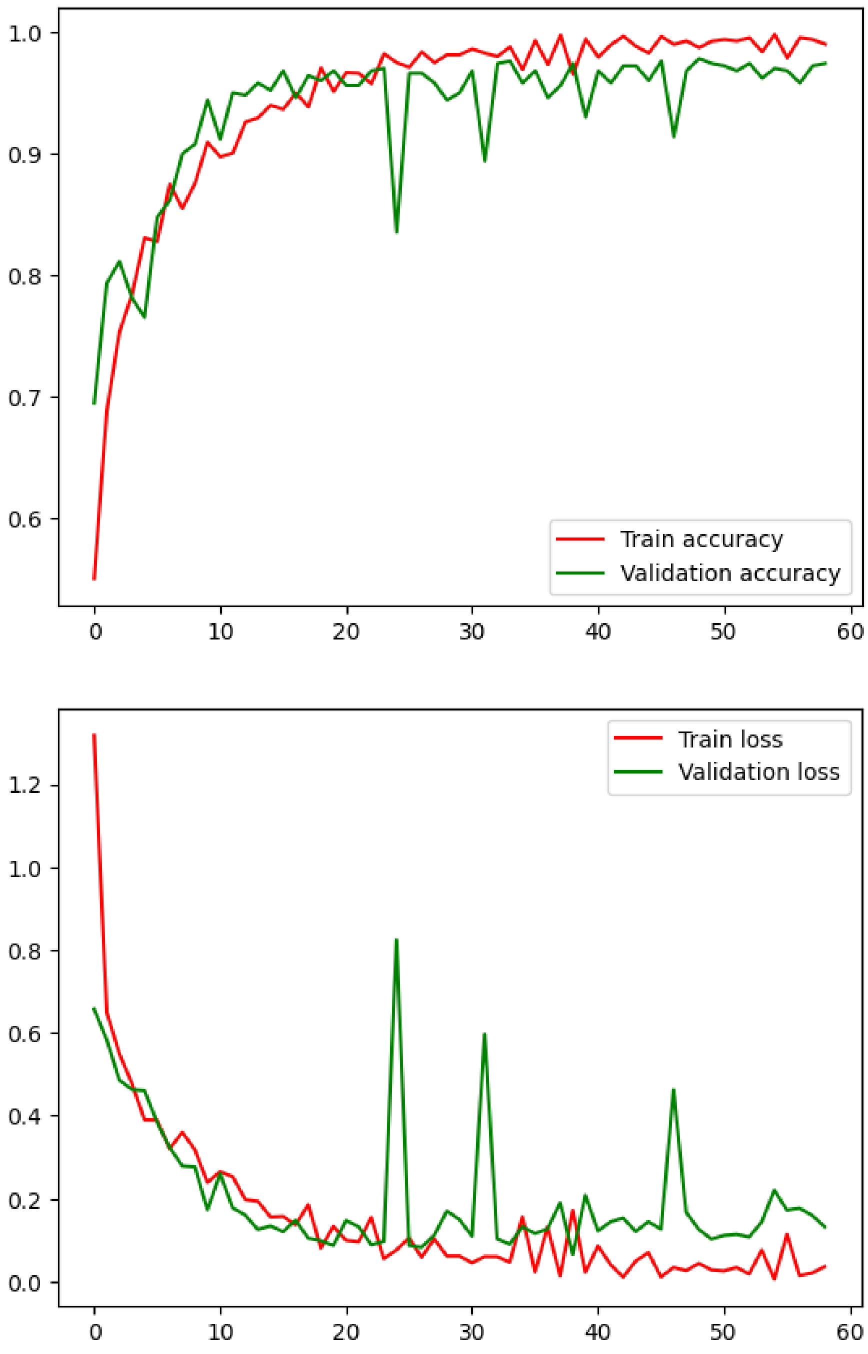


Figure 4.4: Latest Phase Training Analysis

4.2.2 TESTING PROCESS

- The code is used to classify an image as either healthy or unhealthy using a pre-trained machine learning model.
- It prompts the user to enter the file path of an image, reads the image from the file, and displays it using Matplotlib.
- Then it resizes and reshapes the image, passes it to the model to obtain a prediction, and prints a message indicating whether the image is predicted to be healthy (0) or unhealthy (1) based on the value of the prediction.
- When we get a satisfactory result from the testing, we save the trained model to a h5 file which can be retrieved later on.
- If we don’t get a satisfactory result or if the accuracy fluctuates then we again try some layers combinational and property changes and re-test the model for better accuracy.

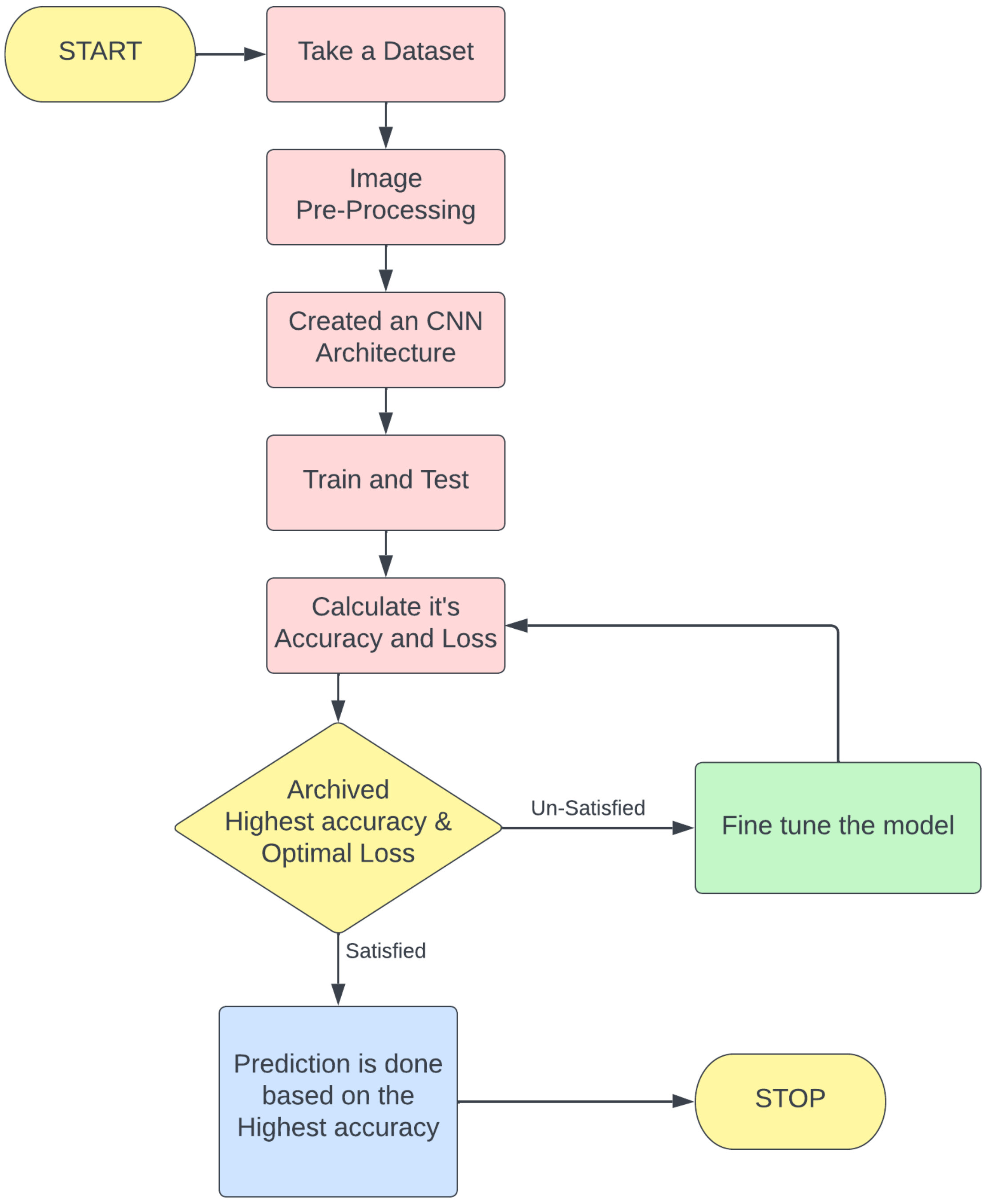

START
Take a Dataset
Image Pre-Processing
Created an CNN Architecture
Train and Test
Calculate it's Accuracy and Loss
Archived Highest accuracy & Optimal Loss
Un-Satisfied
Fine tune the model
Satisfied
Prediction is done based on the Highest accuracy
STOP

Figure 4.5: Project Flowchart

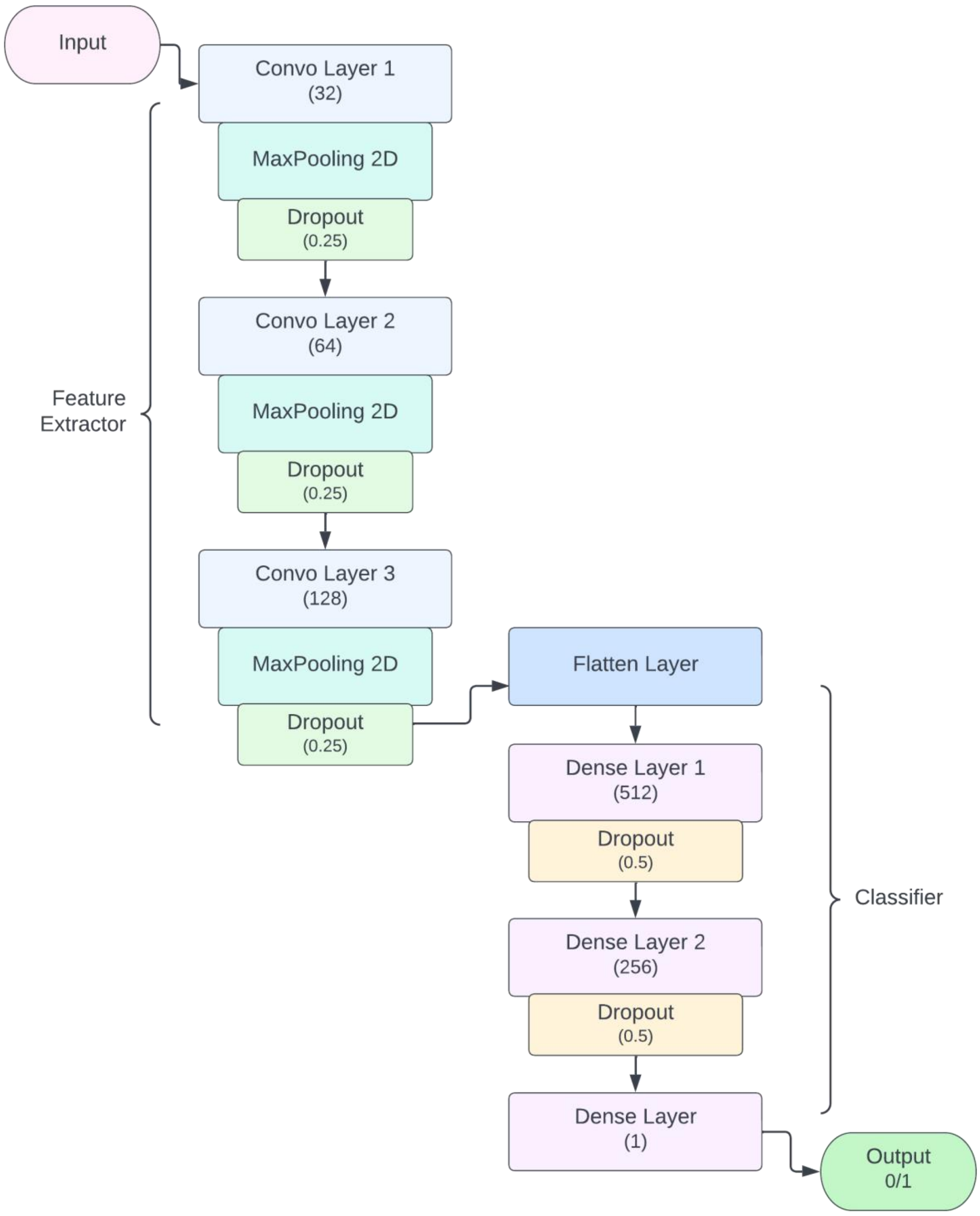


Figure 4.6: Pictorial Representation of Our Model

4.5 DESIGN AND CODE

The codebase for our project is a collection of scripts and modules written in Python. It is designed to implement deep learning model for the classification (whether healthy or not) of plant diseases using leaf images. The codebase is structured in a modular and scalable manner, allowing for the model for further improvement in the future. The overall design of the codebase prioritizes flexibility and efficiency, with a focus on the seamless integration of different libraries and frameworks. In this section, we will discuss the design principles and patterns used in the development of our codebase.

Code for preparing Dataset

```
import os
import random
import shutil

# Created new Folder
new_paths = [
    r"Dataset\train\healthy",
    r"Dataset\train\unhealthy",
    r"Dataset\test\healthy",
    r"Dataset\test\unhealthy",
]

for path in new_paths:
    if not os.path.exists(path):
        os.makedirs(path)
        print("New Folder created !")

from_path = r'SAMPLE DATA (LEAF IMAGE)'
sub_folders = os.listdir(from_path)

healthy_image_list = []
test_healthy_image_list = []
unhealthy_image_list = []
test_unhealthy_image_list = []

# Moving images to destination
for folder in sub_folders:
    healthy = False
    if "healthy" in folder:
        healthy = True
    images = os.listdir(f"{from_path}\\{folder}")
    train_quantity = round(len(sub_folders) / 100 * 90)
    random.shuffle(images)
    current_image = 0

    for image in images:
        old_image_path = f"{from_path}\\{folder}\\{image}"
        if healthy:
            if current_image <= train_quantity:
                new_image_path = f"{new_paths[2]}\\{image}"
            else:
                new_image_path = f"{new_paths[0]}\\{image}"
        else:
            if current_image <= train_quantity:
                new_image_path = f"{new_paths[3]}\\{image}"
            else:
                new_image_path = f"{new_paths[1]}\\{image}"
        shutil.copy(old_image_path, new_image_path)
        current_image += 1

print(len(healthy_image_list), len(unhealthy_image_list))
```

Imported Dataset from Google Drive

```
from google.colab import drive

drive.mount('/content/drive/')
```

Imported all required libraries

```
import tensorflow as tf

from tensorflow import keras

from keras import Sequential

from keras.layers import

Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropout


print(tf.__version__)
```

Generated Data

```
train_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/drive/MyDrive/Dataset/train',
    labels = 'inferred',
    label_mode = 'int',
    batch_size = 128,
    image_size = (256,256),
    shuffle=True,
)

validation_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/drive/MyDrive/Dataset/test',
    labels = 'inferred',
    label_mode = 'int',
    batch_size = 128,
    image_size = (256,256),
    shuffle=True,
)
```

Normalized data

```
def process(image,label):
  image = tf.cast(image/255. ,tf.float32)
  return image, label

train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)
```

Creating the CNN Model

```
from keras import layers

model = keras.Sequential([

    # Input layer
    layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (256, 256, 3)),
    layers.MaxPooling2D(pool_size = (2, 2), strides = 2),
    layers.Dropout(0.25),

    # Hidden layer 1
    layers.Conv2D(64, (3,3), activation = 'relu'),
    layers.MaxPooling2D(pool_size = (2, 2), strides = 2),
    layers.Dropout(0.25),
    # Hidden layer 2
    layers.Conv2D(128, (3,3), activation = 'relu'),
    layers.MaxPooling2D(pool_size=(2, 2), strides = 2),
    layers.Dropout(0.25),

    # Flatten the output from convolutional layers
    layers.Flatten(),

    # Dense layer 1
    layers.Dense(512, activation = 'relu'),
    layers.Dropout(0.5),
    # Dense layer 2
    layers.Dense(256, activation = 'relu'),
    layers.Dropout(0.5),

    # Output layer
    layers.Dense(1, activation = 'sigmoid')
])
```

Printing Model Summary

```
model.summary()
```

Creating checkpoint for Early Stopping

```
from keras.callbacks import EarlyStopping

callback = EarlyStopping(
    monitor = 'val_loss',
    min_delta = 0.00001,
    patience = 15,
    mode = 'auto',
    verbose = 1,
    baseline = None,
    restore_best_weights = False
)
```

Training the model

```
history = model.fit(
    train_ds,
    validation_data = validation_ds,
    epochs = 50,
    callbacks = [callback]
  )
```

Printing Accuracy Graph

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], color = 'red', label = 'Train accuracy')
plt.plot(history.history['val_accuracy'], color = 'green', label = 'Validation accuracy')

plt.legend()
plt.show()
```

Printing Loss Graph

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], color = 'red', label = 'Train loss')
plt.plot(history.history['val_loss'], color = 'green', label = 'Validation loss')

plt.legend()
plt.show()
```

Saving the model to file

```
filename = input("Enter a filename: ")

model.save(f'/content/drive/MyDrive/Dataset/{filename}.h5')
```

Loading model from saved file

```
from keras.models
import Model, load_model

model = load_model('/content/drive/MyDrive/Dataset/only_apple.h5')
```

Testing the model

```
import cv2
import matplotlib.pyplot as plt

filepath = input("Enter image file path: ")

test_img = cv2.imread(filepath)
```

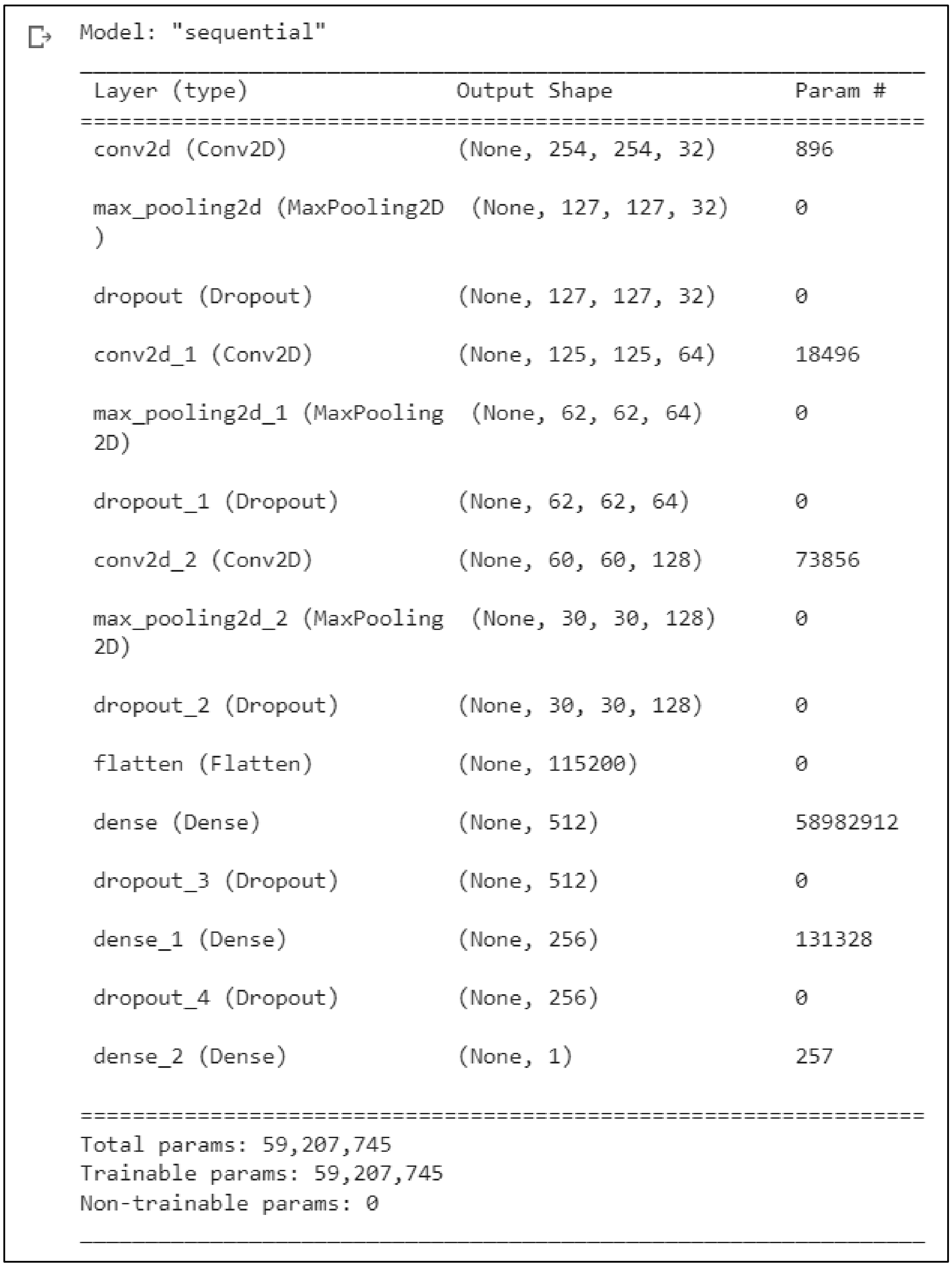
```
plt.imshow(test_img)


test_img = cv2.resize(test_img,(256,256))


test_input = test_img.reshape((1,256,256,3))


result = model.predict(test_input)[0][0]


if result == 0.0:
  print(f'Predicted: Healthy [{result}]')
else:
  print(f'Predicted: Un-Healthy [{result}]')
```

CHAPTER 5

RESULTS

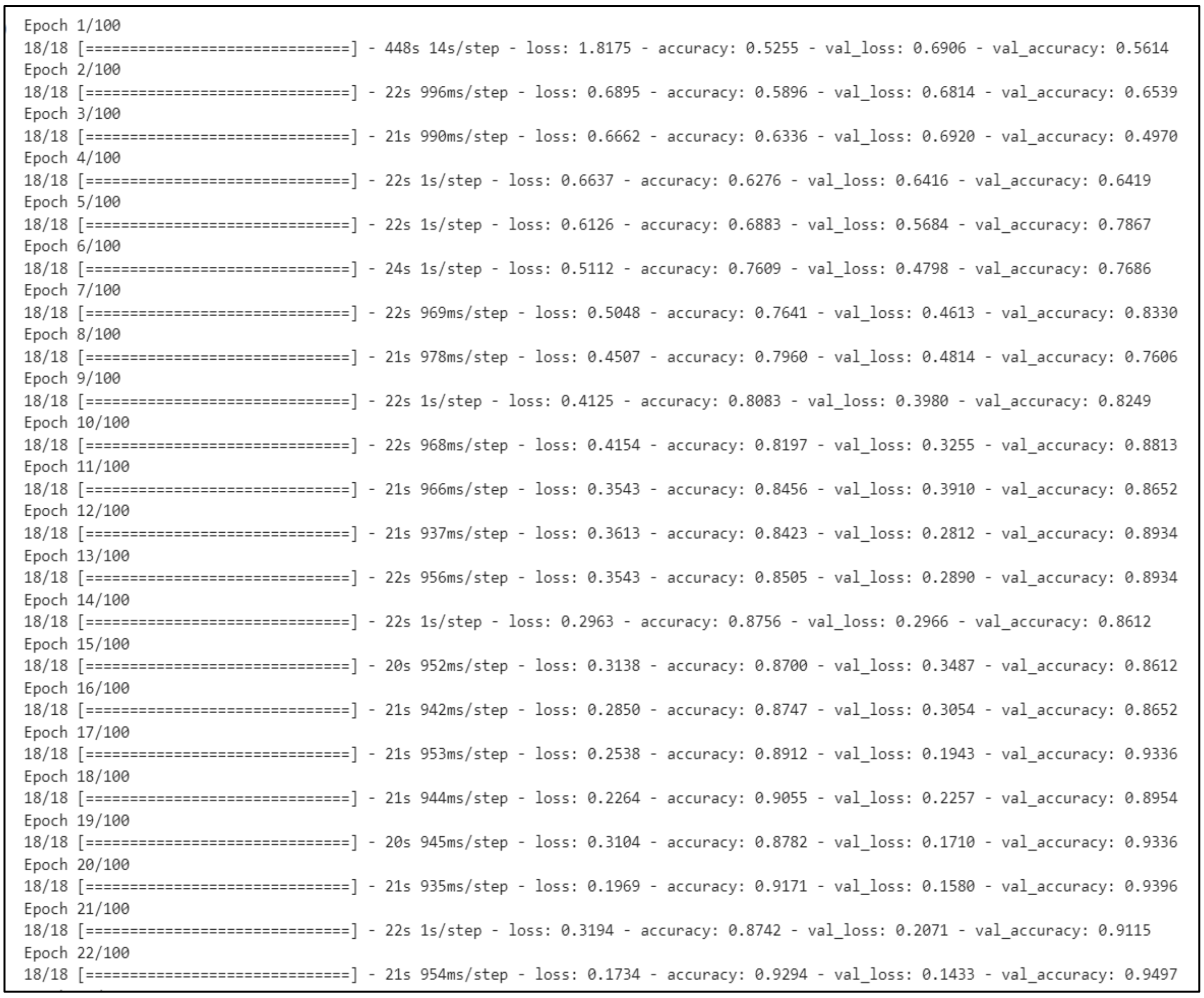
5.1 OUTPUT OF MODEL & TRAINING

We have also presented the summary table, epoch table and outputs.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 dropout (Dropout)           (None, 127, 127, 32)      0

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 62, 62, 64)        0

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 128)      0
 2D)

 dropout_2 (Dropout)         (None, 30, 30, 128)       0

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 512)               58982912

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_4 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 1)                 257

=================================================================
Total params: 59,207,745
Trainable params: 59,207,745
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.1: Model Summary Table

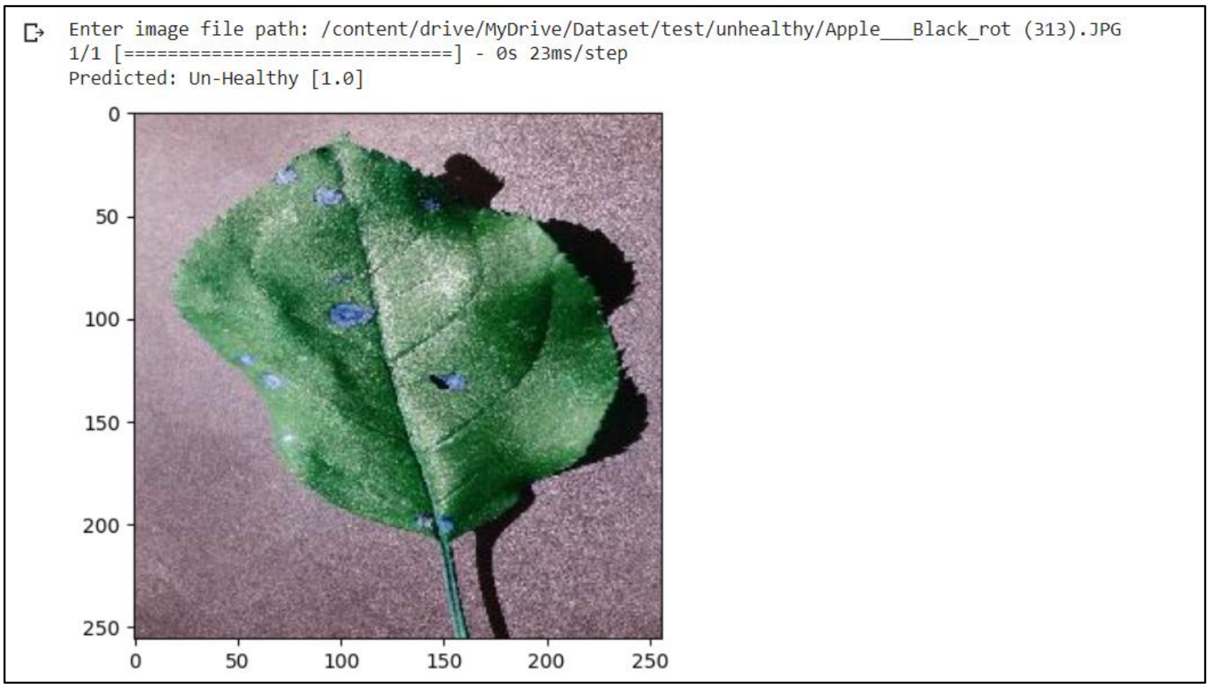


```
Epoch 1/100
18/18 [==============================] - 448s 14s/step - loss: 1.8175 - accuracy: 0.5255 - val_loss: 0.6906 - val_accuracy: 0.5614
Epoch 2/100
18/18 [==============================] - 22s 996ms/step - loss: 0.6895 - accuracy: 0.5896 - val_loss: 0.6814 - val_accuracy: 0.6539
Epoch 3/100
18/18 [==============================] - 21s 990ms/step - loss: 0.6662 - accuracy: 0.6336 - val_loss: 0.6920 - val_accuracy: 0.4970
Epoch 4/100
18/18 [==============================] - 22s 1s/step - loss: 0.6637 - accuracy: 0.6276 - val_loss: 0.6416 - val_accuracy: 0.6419
Epoch 5/100
18/18 [==============================] - 22s 1s/step - loss: 0.6126 - accuracy: 0.6883 - val_loss: 0.5684 - val_accuracy: 0.7867
Epoch 6/100
18/18 [==============================] - 24s 1s/step - loss: 0.5112 - accuracy: 0.7609 - val_loss: 0.4798 - val_accuracy: 0.7686
Epoch 7/100
18/18 [==============================] - 22s 969ms/step - loss: 0.5048 - accuracy: 0.7641 - val_loss: 0.4613 - val_accuracy: 0.8330
Epoch 8/100
18/18 [==============================] - 21s 978ms/step - loss: 0.4507 - accuracy: 0.7960 - val_loss: 0.4814 - val_accuracy: 0.7606
Epoch 9/100
18/18 [==============================] - 22s 1s/step - loss: 0.4125 - accuracy: 0.8083 - val_loss: 0.3980 - val_accuracy: 0.8249
Epoch 10/100
18/18 [==============================] - 22s 968ms/step - loss: 0.4154 - accuracy: 0.8197 - val_loss: 0.3255 - val_accuracy: 0.8813
Epoch 11/100
18/18 [==============================] - 21s 966ms/step - loss: 0.3543 - accuracy: 0.8456 - val_loss: 0.3910 - val_accuracy: 0.8652
Epoch 12/100
18/18 [==============================] - 21s 937ms/step - loss: 0.3613 - accuracy: 0.8423 - val_loss: 0.2812 - val_accuracy: 0.8934
Epoch 13/100
18/18 [==============================] - 22s 956ms/step - loss: 0.3543 - accuracy: 0.8505 - val_loss: 0.2890 - val_accuracy: 0.8934
Epoch 14/100
18/18 [==============================] - 22s 1s/step - loss: 0.2963 - accuracy: 0.8756 - val_loss: 0.2966 - val_accuracy: 0.8612
Epoch 15/100
18/18 [==============================] - 20s 952ms/step - loss: 0.3138 - accuracy: 0.8700 - val_loss: 0.3487 - val_accuracy: 0.8612
Epoch 16/100
18/18 [==============================] - 21s 942ms/step - loss: 0.2850 - accuracy: 0.8747 - val_loss: 0.3054 - val_accuracy: 0.8652
Epoch 17/100
18/18 [==============================] - 21s 953ms/step - loss: 0.2538 - accuracy: 0.8912 - val_loss: 0.1943 - val_accuracy: 0.9336
Epoch 18/100
18/18 [==============================] - 21s 944ms/step - loss: 0.2264 - accuracy: 0.9055 - val_loss: 0.2257 - val_accuracy: 0.8954
Epoch 19/100
18/18 [==============================] - 20s 945ms/step - loss: 0.3104 - accuracy: 0.8782 - val_loss: 0.1710 - val_accuracy: 0.9336
Epoch 20/100
18/18 [==============================] - 21s 935ms/step - loss: 0.1969 - accuracy: 0.9171 - val_loss: 0.1580 - val_accuracy: 0.9396
Epoch 21/100
18/18 [==============================] - 22s 1s/step - loss: 0.3194 - accuracy: 0.8742 - val_loss: 0.2071 - val_accuracy: 0.9115
Epoch 22/100
18/18 [==============================] - 21s 954ms/step - loss: 0.1734 - accuracy: 0.9294 - val_loss: 0.1433 - val_accuracy: 0.9497
```

Figure 5.2: Epoch Runtime

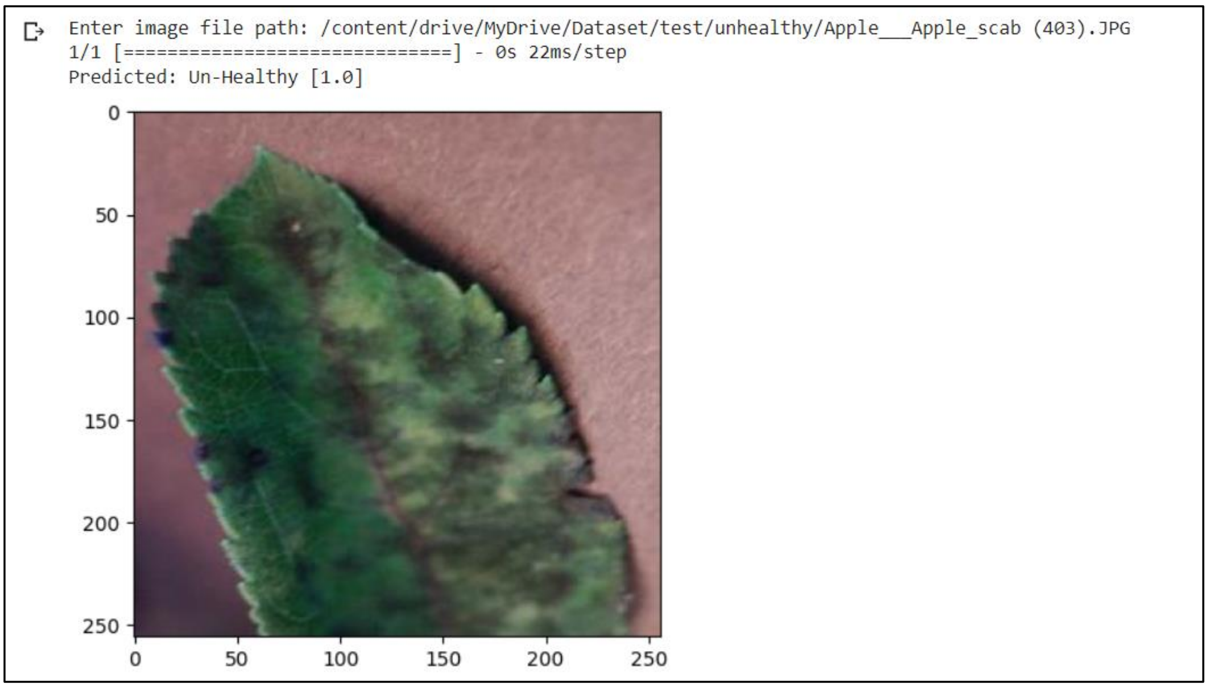
5.2 OUTPUTS OF TESTING LEAVES

Here are some outputs of the leaf images that we took randomly and tested in our model to demonstrate the working of our model.



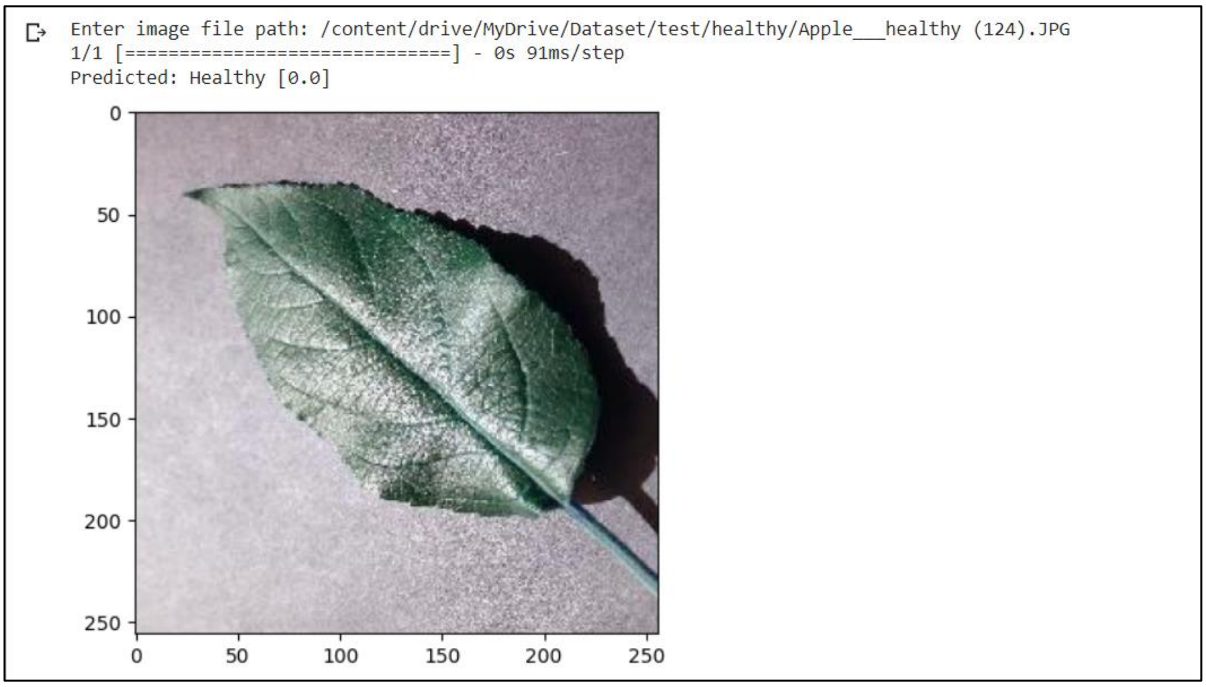
```
Enter image file path: /content/drive/MyDrive/Dataset/test/unhealthy/Apple___Black_rot (313).JPG
1/1 [==============================] - 0s 23ms/step
Predicted: Un-Healthy [1.0]
```

Figure 5.3: Testing of a Un-Healthy Leaf Image



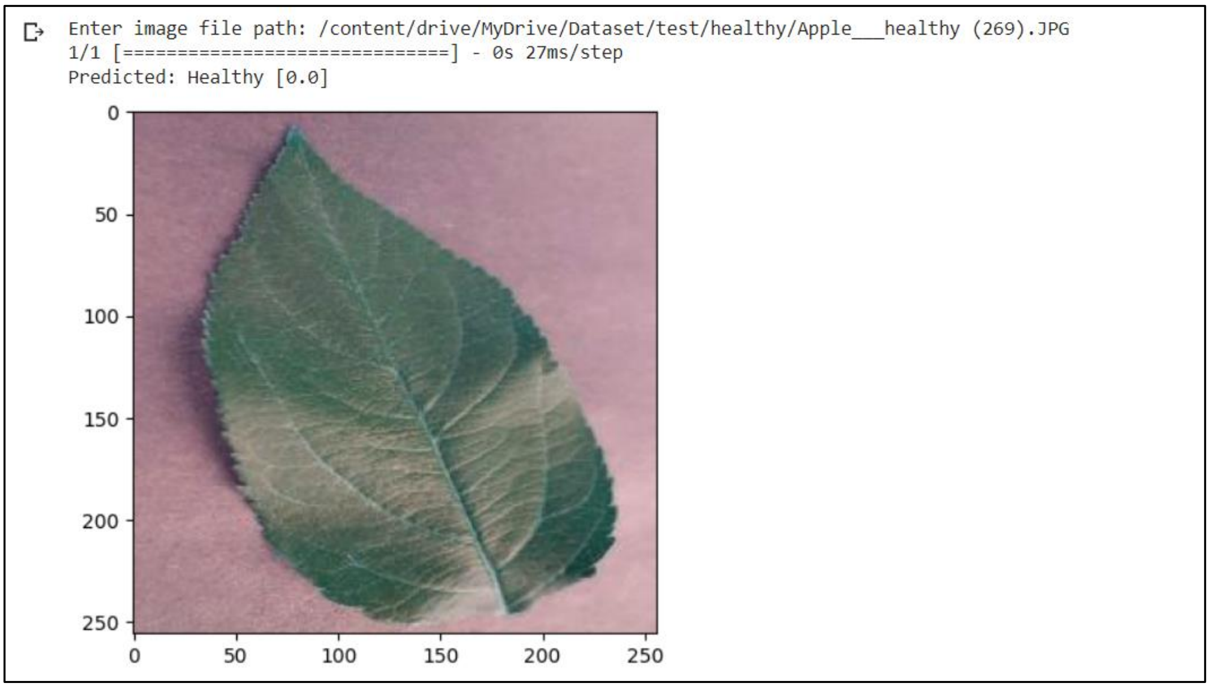
```
Enter image file path: /content/drive/MyDrive/Dataset/test/unhealthy/Apple___Apple_scab (403).JPG
1/1 [==============================] - 0s 22ms/step
Predicted: Un-Healthy [1.0]
```

Figure 5.4: Testing of another Un-Healthy Leaf Image



```
Enter image file path: /content/drive/MyDrive/Dataset/test/healthy/Apple___healthy (124).JPG
1/1 [==============================] - 0s 91ms/step
Predicted: Healthy [0.0]
```

Figure 5.5: Testing of a Healthy Leaf Image



```
Enter image file path: /content/drive/MyDrive/Dataset/test/healthy/Apple___healthy (269).JPG
1/1 [==============================] - 0s 27ms/step
Predicted: Healthy [0.0]
```

Figure 5.6: Testing of another Healthy Leaf Image

CHAPTER 6

FUTURE PLANS

Machine learning comes with huge possibilities and we have discovered just a part of it till now, in future we will explore and try new features and functionalities in our CNN model and try to improve it.

Machine learning requires a lot of computational resources, thus high performing machines are must for developing such type of model. We were also limited because of our machine limitations. For now, we have used a cloud service Google Colab as our main platform, but as it is a free service it also comes with many limitations. In future if we can arrange a high performing machine then we will be able to train our model with a large number of dataset and also run large number of epochs which will definitely improve our model’s accuracy.

CHAPTER 7

CONCLUSION

We have studied the basic features of machine learning, neural networks, convolutional network, their layers, binary classification and other features. We have also learnt a bit about image processing techniques like image segmentation, feature extraction, and classifications.

In conclusion, the use of deep learning with binary classification is a feasible approach for the detection of leaf diseases. With the availability of suitable datasets and adequate computing resources, a deep learning model can be trained to accurately distinguish between healthy and diseased leaves. However, there may be potential challenges due to the variability of the images and the need for a large amount of training data. These challenges can be addressed through the use of data augmentation techniques and transfer learning. With careful planning and execution, a leaf disease detection system based on deep learning can provide an effective and accurate solution for the detection of plant diseases.

To ensure the feasibility of our project, we used Python as our programming language and TensorFlow as our deep learning framework. These tools have proven to be effective in developing deep learning models and are widely used in the research community.

We also conducted a feasibility study to determine the resources required for our project. Our system is designed to work on limited resources, making it feasible for small-scale farmers and researchers who may not have access to high-end hardware. In terms of the design of our system, we developed data flow diagrams to illustrate the flow of data through our system. The diagrams show how our system can take input images of plant leaves and output whether it is diseased or not.

We have made our own CNN model with various layers and the best accuracy we got is more than 94%. In future, we will add more classes of leaves and also disease classifications.

Thus, the leaf disease detection project using deep learning with binary classification is a promising approach for identifying and classifying plant diseases. With a suitable dataset and adequate computing resources, a deep learning model can be trained to accurately distinguish between healthy and diseased leaves. However, the variability of the images due to environmental factors and the requirement of a large training dataset can pose challenges. These challenges can be addressed using data augmentation techniques and transfer learning.

Overall, the success of this project relies on careful planning and execution, but it has the potential to provide an effective and accurate solution for plant disease detection.

In conclusion, our project is a significant step towards the development of an accurate and efficient system for plant disease detection. The use of deep learning techniques and our proposed system design makes it feasible for small-scale farmers and researchers to use, which can have a significant impact on the agriculture industry. Moving forward, there is still a lot of potential for our project to improve.

For instance, we can continue to add new deep learning algorithms and techniques and improve the accuracy of our existing model by using more extensive datasets.   Our project provides a promising solution to the problem of plant disease detection, and we believe that it has the potential to make a significant impact on the agriculture industry.

REFRENCES

1. Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. International Journal of Computer Vision, 88(2), 303-338. https://link.springer.com/article/10.1007/s11263-009-0275-4
2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25 (NIPS 2012). https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924 a68c45b-Abstract.html
3. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556
4. Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In European Conference on Computer Vision (pp. 818-833). https://doi.org/10.1007/978-3-319-10590-1_53
5. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385
6. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. https://arxiv.org/abs/1512.00567
7. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., … & FeiFei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), 211-252. https://doi.org/10.1007/s11263-015-0816-y

8. PlantVillage Dataset
https://paperswithcode.com/dataset/plantvillage
9. Hughes, D. P., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics
https://arxiv.org/abs/1511.08060
10. Arunangshu Pal, Vinay Kumar(2023). AgriDet:Pant Leaf Disease severity classification using agriculture detection framework.
https://www.sciencedirect.com/science/article/abs/pii/S0952197622007448
11. Leaf Disease - Assignment Report, STUDOCUS
https://www.studocu.com/in/document/andhra-university/linear-ics-applications/leaf-disease-assignment-report/56897762
12. TensorFlow Documentation
https://www.tensorflow.org
13. KERAS Documentation
URL: https://keras.io/getting_started
14. YouTube Channel: CampusX
https://www.youtube.com/@campusx-official
15. YouTube Channel: Coding Lane
https://www.youtube.com/@CodingLane