PLANT DISEASE DETECTION APPLICATION

A MAJOR PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

BY

BATCH-16B

N. SAI KIRAN	**N.D.V.CHOWDARY**
(19331A05B8)	**(19331A05B6)**
K. MAHESH	**B.N.S.ROHAN**
(19331A0588)	**(19331A05C2)**

Under the Supervision of
Dr.P. SATHEESH
Head of Department

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MVGR COLLEGE OF ENGINEERING (Autonomous)
VIZIANAGARAM-535005, AP (INDIA)
(Accredited by NBA, NAAC, and Permanently Affiliated to Jawaharlal Nehru Technological University Kakinada)

April, 2023

Maharaj Vijayaram Gajapathi Raj (MVGR) College of Engineering(A)
Vizianagaram

CERTIFICATE

This is to certify that the project report entitled "**PLANT DISEASE DETECTION APPLICATION**" being submitted by **N. SAI KIRAN, K. MAHESH, B. N. S. ROHAN, N.D.V.CHOWDARY** bearing registered numbers **19331A05B8, 19331A0588, 19331A05C2, 19331A05B6** in partial fulfillment for the award of the degree of "**Bachelor of Technology" in Computer Science and Engineering** is a record of bonafide work done by them under my supervision during the academic year 2022-2023.

Dr. P. SATHEESH	**Dr. P. RAVI KIRAN VARMA**
Head of Department,	**Head of the Department,**
Department of DE,	Department of CSE,
MVGR College of Engineering,	MVGR College of Engineering,
Vizianagaram.	Vizianagaram.

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the work done on the dissertation entitled **“PLANT DISEASE DETECTION APPLICATION”** has been carried out by us and submitted in partial fulfilment for the award of credits in Bachelor of Technology in Computer Science and Engineering of MVGR College of Engineering (Autonomous) and affiliated to Jawaharlal Nehru Technological University (Vizianagaram). The various contents incorporated in the dissertation have not been submitted for the award of any degree of another institution or university.

ACKNOWLEDGEMENTS

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

We place on record our heartfelt appreciation and gratitude to **Dr.P.SATHEESH** for the immense cooperation and navigation as mentor in bringing out the project work under his guidance. His uncompromising attitude to bring out the best with constructive suggestions inspired us to achieve the results set forth for the project work. We are deeply indebted to him for his excellent, enlightened and enriched guidance.

We would like to express our sincere gratitude to **Mr. P. R. Santosh Naidu** for his invaluable contribution and guidance throughout the project.

We also thank **Dr. K. V. L. Raju**, Principal, and **Dr. P. Ravi Kiran Varma,** Head of the Department, for extending their utmost support and co-operation in providing all the provisions for the successful completion of the project.

We are also thankful for and fortunate enough to get constant encouragement and guidance from our panel members **Mrs.K.SantoshJhansi, Mr.M.Chandra Sekhar, G.Lakshmana rao.** We sincerely thank all the members of the staff in the Department of Computer Science & Engineering for their sustained help in our pursuits. We thank all those who contributed directly or indirectly in successfully carrying out this work.

N.SAI KIRAN	**(19331A05B8)**
K.MAHESH	**(19331A0588)**
B.N.S.ROHAN	**(19331A05C2)**
N.D.V.CHOWDARY	**(19331A05B6)**

ABSTRACT

Crop diseases are a major threat to food security, but their rapid identification remains difficult in many parts of the world due to the lack of the necessary infrastructure. The combination of increasing global smartphone penetration and recent advances in computer vision made possible by deep learning has paved the way for smartphone-assisted disease diagnosis. Overall, the approach of training deep learning models on increasingly large and publicly available image datasets presents a clear path toward smartphone-assisted crop disease diagnosis on a massive global scale. Plant diseases and pests are important factors determining the yield and quality of plants. Plant diseases identification can be carried out by means of digital image processing. In recent years, deep learning has made breakthrough in the field of digital image processing, far superior to traditional methods. How to use deep learning technology to study plant diseases identification has become a research issue of great concern to researchers. According to the difference of network structure, this study outlines the research on plant diseases detection based on deep learning in recent years from three aspects of classification network, detection network and segmentation network, and the advantages and disadvantages of each method are summarized. Common datasets are introduced, and the performance of existing studies is compared. On this basis, this study discusses possible challenges in practical applications of plant diseases detection based on deep learning. Inaddition, possible solutions and research ideas are proposed for the challenges, and several suggestions are given. Finally, this study gives the analysis and prospect of the future trend of plant diseases based on deep learning.

TABLE OF CONTENTS

1. INTRODUCTION 10

1.1. INTRODUCTION TO THE PROJECT 10

1.2. PROJECT OBJECTIVES 11

1.3. SCOPE OF THE PROJECT 11

2. LITERATURE SURVEY AND REVIEW 12

2.1. LITERATURE SURVEY 12

2.2. LITERATURE REVIEW 15

3. PROBLEM DEFINITION 16

3.1. BACKGROUND 16

3.2. PROBLEM STATEMENT 16

3.3. SOLUTION 16

4. DATA COLLECTION AND PROCUREMENT 17

4.1. DATA COLLECTION 17

4.2. SOFTWARE REQUIREMENTS 18

4.2.1.Python 18

4.2.2.OpenCV 18

4.2.3.Tensorflow 19

4.2.4.InceptionV3 20

4.2.5.Convolutional Neural Network (CNN) 22

4.2.6.Integrated Development Environment(IDE) 23

4.2.7.Android Software Development Kit 24

4.3. HARDWARE REQUIREMENTS 25

5. DESIGN AND IMPLEMENTATION 26

5.1. SYSTEM DESIGN 26

5.1.1.Image Preprocessing 27

5.1.2.Feature Extraction 27

5.1.3.Image Augmentation 28

5.1.4.Model training 29

5.1.5.Model evaluation 29

5.1.6.Classification 29

5.2. METHODOLOGY AND IMPLEMENTATION 30
5.2.1.Load Data 30
5.2.2.Pre-processing 30
5.2.3.Build Model 30
5.2.4.Specify Loss Function and Train model 30
5.2.5.Predictions 31
6. RESULTS AND DISCUSSIONS 32
6.1.RESULTS 32
6.1.1.Plant Disease Detection App interface 32
6.1.2.Home Page 33
6.1.3.Interface for scan using camera 34
6.1.4.Scanning a leaf 35
6.1.5.Detection 36
6.2. DISCUSSIONS 36
7. CONCLUSION 37
7.1. LIMITATIONS OF THE PROJECT 37
7.2. FUTURE WORK 38
8. REFERENCES 39

LIST OF FIGURES

- ❖ Fig 4.1 - Plant Disease detection using tensorflow
- ❖ Fig 4.2 - InceptionV3
- ❖ Fig 4.3 - CNN for plant disease detection
- ❖ Fig 5.1 - System Design
- ❖ Fig 5.2 - Load Data
- ❖ Fig 5.3 - Preprocessing data
- ❖ Fig 5.4- Build Model
- ❖ Fig 5.5- Specify loss function and train model
- ❖ Fig 5.6 - Predictions
- ❖ Fig 6.1 - App interface
- ❖ Fig 6.2 - Home Page
- ❖ Fig 6.3 - Interface to open camera
- ❖ Fig 6.4 - Leaf scan
- ❖ Fig 6.5 - Detection of disease

LIST OF TABLES

- ❖ Table 2.1 - Literature Survey

LIST OF ABBREVIATIONS

- ML – Machine Learning
- CNN – Convolutional Neural Networks
- RNN – Recurrent Neural Networks
- DBN - Deep Belief Networks
- TL – Transfer Learning
- UI – User Interface
- ADB - Android Debug Bridge
- API – Application Programming Interface
- SDK – Software Development Kit
- IDE – Integrated Development Environment
- GLM – Gray Level Occurrence Matrix
- LBP – Local Binay Patterns
- SVM – Support Vector Machine
- NN – Neural Networks
- RLM – Run Length Matrix

1. INTRODUCTION

1.1. INTRODUCTION TO THE PROJECT

Agriculture is a major sector of the economy, and plant diseases can cause significant damage to crop, leading to lower yields and economic losses. The objective of this project is to develop a plant disease detection app using convolutional neural networks (CNN) to help farmers and gardeners identify the diseases affecting their plants. The app will enable users to take timely action to prevent the spread of diseases and protect their crops. The significance of this project lies in the fact that it can help increase crop yields, reduce the use of pesticides, and promote a more sustainable and healthier agricultural system.

The Plant disease detection app using CNN is a project that aims to detect and classify plant diseases using Convolutional Neural Networks (CNN). The main objective of this project is to develop a mobile application that can help farmers and gardeners identify plant diseases quickly and accurately.

Plant diseases can have a significant impact on the growth and yield of crops, and early detection is crucial to prevent the spread of diseases and minimize losses. Traditionally, plant disease detection involves visual inspection by human experts, which can be time-consuming and often prone to errors. With the advancement of computer vision and deep learning technologies, it is now possible to automate the process of plant disease detection and provide faster and more accurate diagnoses.

The significance of this project lies in its potential to improve the efficiency and productivity of agriculture. By enabling early detection and intervention, this app can help farmers prevent crop losses and minimize the use of pesticides and other harmful chemicals. Additionally, this app can provide valuable data and insights to researchers and policymakers, helping them to better understand the prevalence and distribution of plant diseases and develop effective strategies for disease control and management.

1.2. PROJECT OBJECTIVES

1. To design a system that can detect plant disease accurately and precisely using various image classification techniques.
2. To design a system that detects plant disease quickly.
3. Designing a system that can provide a relevant information about the disease, symptoms and other relevant information.
4. Providing solution for the detected disease regarding its curing and other helpful information.
5. Designing a system that provides the interface that is user friendly to the farmers and for easy usage.

1.3. SCOPE OF THE PROJECT

The scope of a plant disease detection Android app project would typically involve developing an application that uses image processing and machine learning techniques to identify and classify diseases in plants. The app would be able to capture images of plants, analyze them using image processing algorithms, and then use machine learning models to identify whether the plant is healthy or diseased.

The features of such an app might include:

1. **Image capture:** The app should allow the user to capture images of the plant using their smartphone camera.
2. **Image processing:** The app should use image processing algorithms to detect and extract features from the plant images.
3. **Machine learning models:** The app should use machine learning models to classify the plant images as healthy or diseased.
4. **Disease identification:** The app should identify the specific disease affecting the plant and provide information on how to treat it.
5. **User interface:** The app should have an intuitive and user-friendly interface that allows users to easily capture and analyze images of their plants.

The project would involve developing an app that can accurately detect and diagnose plant diseases and provide users with information on how to treat them. The project would require expertise in image processing, machine learning, and app development.

2. LITERATURE SURVEY AND REVIEW

2.1. LITERATURE SURVEY

Table 2.1 - Literature Survey

S no.	Authors	Journals Name	Methodologies	GAPS
1.	Simha Ramesh Ramachandra Hebbar	Plant Disease Detection Using Machine Learning	To find out whether the leaf is diseased or healthy by Preprocessing, Feature extraction, Training of classifier and Classification in "Machine Learning."	In future Projects we need to optimize The feature extraction of the plant diseases detection techniques that helps to extract the infected region in the leaf or plant to identify the diseases. The major role of this technology is depends on the software and the database of the application system. We use the MALAB in python programming for the image processing techniques are more efficient.
2.	Sachin D. Khirade A.B.Patil	Plant Disease Detection Using Image Processing	plant disease detection and classification using "image processing"	The future work is to increase the number of images present in the predefined database and to modify the architecture in accordance with

				the dataset for achieving better accuracy.
3.	Peyman Moghadam Daniel Ward	Plant Disease Detection Using Hyperspectral Imaging	Plant Disease Detection Using “Hyperspectral Imaging”	In future studies, transfer learning methods can be extended to more scenarios, such as different regions and different equipment. Transfer learning has great potential for rice disease detection and will contribute to the translation of relevant researches into practical applications in an efficient manner.
4.	Trimi Neha Tete Sushma Kam ulu	Detection of plant disease using threshold, k-mean cluster and ann algorithm	The methodology is used in this are two different segmentation techniques such as thresholding and K-means clustering algorithm.	In feature we have planned to increase the efficiency and accuracy of the detection by using other algorithms Like ostu algorithm. Also we will increase the no of diseases that could be identified by this alogirithm
5.	Anshul Bhatia Anuradha Chug	Plant disease detection for high dimensional imbalanced dataset using an enhanced decision tree	The three techniques used in this study are explained here. CFS filter algorithm works on correlation based heuristic estimation function for the ranking of features,	In future, this work can be extended with more feature selection algorithms. In addition to this, a hybrid solution of

		approach	RFI filter uses RF algorithm to find weights of attributes, whereas Cons filter uses best fit search algorithm to identify the proper feature subset	classification algorithms along with soft computing techniques like genetic or fuzzy algorithms can also be proposed and tested on SBL dataset.
6.	Shital Bankar Ajita Dube,Pranali Kadam	Plant Disease Detection Techniques Using Canny Edge Detection & Color Histogram in Image Processing	The methodology is used in this are Canny Edge Detection & Color Histogram in Image Processing	The future work mainly concerns with the large database. After Detecting disease shows medicine on that disease. Another work is a particular field with advance features and technology.
7.	Faye Mohameth Chen Bingcai	Plant Disease Detection with Deep Learning and Feature Extraction Using Plant Village	Transfer learning as well as deep feature extraction is implemented using the classifiers on the data set	In a future project, we would like to extend this work by collecting our own data set in sub- equatorial zones where the cultivable lands might be hostile to the development and survival of plants. This will allow us to study the behavior of the plant, while detecting the main threats to its survival according to its environment.

2.2. LITERATURE REVIEW

The studies reviewed offer valuable contributions to the field of plant disease detection Android app development. One key takeaway is the significance of employing deep learning algorithms and machine learning techniques to improve the accuracy of disease detection.

Furthermore, some studies have explored the use of crowdsourcing approaches to increase the amount and diversity of data used to train models. Another important aspect highlighted by the studies is the need for accurate disease detection and appropriate treatment recommendations to ensure optimal crop yield.

Finally, the potential of mobile devices to provide real-time feedback on plant health status is also highlighted, emphasizing the significance of developing user-friendly and accessible interfaces for such applications.

Overall, these studies provide valuable guidance and direction for the development of effective and efficient plant disease detection Android apps.

3. PROBLEM DEFINITION

3.1. BACKGROUND

Plant diseases are a significant challenge for farmers and gardeners worldwide, as they can cause significant losses in crop yields and quality. Early detection and management of plant diseases are critical to preventing their spread and reducing their impact on plant health and productivity. Traditional methods of plant disease diagnosis, such as manual inspection and laboratory testing, can be time-consuming, expensive, and require specialized knowledge and equipment. Therefore, there is a need for a fast, reliable, and accessible tool for plant disease detection that can be used by anyone, regardless of their expertise in plant pathology.

3.2. PROBLEM STATEMENT

The problem to be addressed by the plant disease detection Android app project is to develop an application that can accurately and efficiently identify plant diseases and provide users with information about them. The app should be designed to work on Android devices and be accessible to users with low-end devices and limited internet connectivity. Additionally, the app should be user-friendly, reliable, and provide valuable insights and recommendations to users.

3.3. SOLUTION

To address the problem, the plant disease detection Android app project will develop an app that uses image recognition algorithms to analyze pictures of plants and detect signs of disease.Users can upload pictures of their plants and receive a diagnosis of any diseases that may be present. The app will also have a community feature where users can share their experiences and get advice from experts and fellow users. The app will be designed to work in areas with limited internet connectivity and be accessible to users with low-end Android devices.

4. DATA COLLECTION AND PROCUREMENT

4.1. DATA COLLECTION

Collecting data is an important step in the plant disease detection app project. For this we have to collect various images of plants species and diseases from various sources. Data collection can be done using following steps:

1. **Search for relevant datasets:** Kaggle is a popular platform for hosting datasets, including those related to plant diseases. Kaggle contains various datasets of plants species and diseases and thousands of images for each plant. The first step is to search for relevant datasets on Kaggle. Some popular datasets include the "Plant Village Dataset," which contains images of plant leaves affected by various diseases. Plant Village dataset consists of 54303 healthy and unhealthy images of 38 categories by species. These images include the tomato, potato, corn and other plants and their healthy and unhealthy images.
2. **Download the dataset:** Once you have identified a relevant dataset, download it to your computer. The Plant Village dataset is available in two formats - an image dataset and a zip file containing the images and their corresponding labels. Choose the format that suits your project. Check the accuracy and reliability of the data by reviewing the data description and examining a sample of the data. Make sure that the dataset is relevant to your project and meets your requirements.
3. **Preprocess the data:** The data may need to be preprocessed before it can be used to train the machine learning model. Preprocess the dataset by removing any duplicates, cleaning the data, and separating it into training and testing sets. This can include resizing the images, removing duplicates, and removing irrelevant data.
4. **Organize the data:** Organize the data in a format that is suitable for your machine learning model. This may involve resizing the images or converting them to a specific file format.
5. **Store the data:** Store the preprocessed data and labeled images in a secure and organized manner to ensure that it is easily accessible and can be used to train the machine learning model.

6. **Evaluate the data quality:** Before using the data to train the machine learning model, it is essential to evaluate the quality of the dataset. This can be done by randomly selecting a subset of images and manually verifying the labeling accuracy.
7. **Train the machine learning model:** Use the preprocessed data to train your machine learning model to detect different types of plant diseases.
8. **Test the machine learning model:** Test your machine learning model by running it on the testing dataset and comparing the results with the actual labels.

4.2. SOFTWARE REQUIREMENTS

4.2.1.Python: Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is very useful for students and working professionals to become a great Software Engineer especially when they are working in the Web Development Domain.

Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.

Python is Interactive − You can sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

4.2.2.OpenCV: OpenCV (Open Source Computer Vision) is an open-source computer vision and machine learning library that can be used to develop real-time computer vision applications. It was originally developed by Intel in 1999 but is now maintained by a group of developers worldwide.

OpenCV provides a set of powerful image and video processing algorithms, such as object detection, recognition, and tracking, face detection, image filtering, and morphological operations. It also includes machine learning algorithms, such as decision trees, support vector machines, and neural networks, to perform complex tasks such as classification, regression, and clustering.

OpenCV supports several programming languages, including C++, Python, Java, and MATLAB, making it accessible to developers with different programming backgrounds. It also provides a set of easy-to-use APIs and functions to process images and videos in real-time, making it an ideal tool for developing computer vision applications such as robotics, autonomous vehicles, and augmented reality.

Some of the popular use cases of OpenCV include face recognition, object tracking, surveillance systems, medical imaging, and autonomous vehicles. OpenCV has been widely used in research and industry, and it has become an essential tool for computer vision researchers and practitioners.

OpenCV provides a powerful set of tools and algorithms for image processing and machine learning, which can be used to develop accurate and efficient plant disease detection systems. By combining OpenCV with other technologies such as mobile app development, developers can create innovative and accessible tools for plant disease detection and management.

4.2.3.Tensorflow: TensorFlow is an open-source machine learning library developed by Google Brain team, which is widely used for building and training deep learning models. It provides a set of tools, libraries, and resources for developing, deploying, and managing machine learning models in various environments, including mobile, web, and cloud.

TensorFlow uses a computational graph to represent the mathematical operations in a deep learning model. It allows developers to define and execute complex operations on large datasets using a simple and intuitive API. TensorFlow supports a wide range of neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep belief networks (DBNs).

TensorFlow provides a variety of high-level APIs, such as Keras and Estimator, which simplify the process of building and training deep learning models. These APIs provide pre-built models and components that can be easily integrated into a machine learning pipeline.

TensorFlow also supports distributed computing, which allows developers to distribute the training of deep learning models across multiple machines or devices. This feature enables developers to scale up the training process and handle larger datasets.

TensorFlow has been used for a wide range of applications, including natural language processing, computer vision, speech recognition, and robotics. It has also been used in various industries, such as healthcare, finance, and manufacturing.

TensorFlow is a powerful and flexible machine learning library that provides a wide range of tools and resources for building and training deep learning models. Its popularity and community support make it an ideal choice for developers and researchers who are interested in exploring and advancing the field of machine learning.

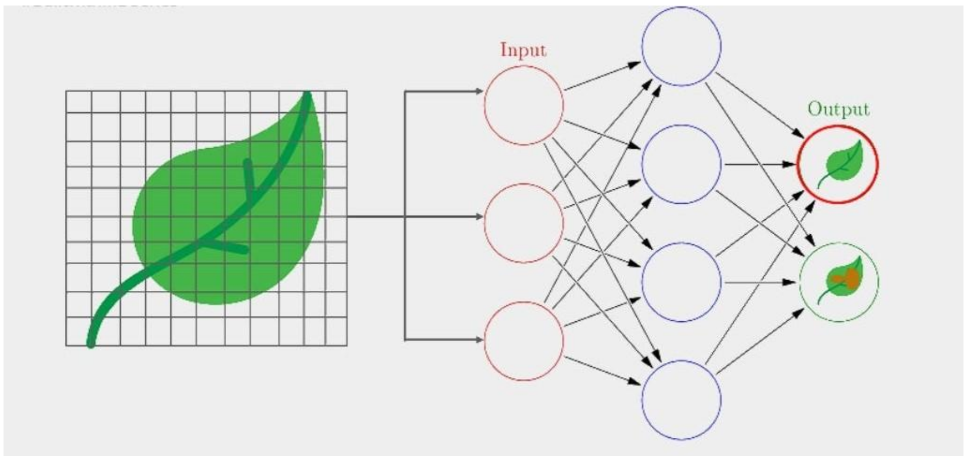


Fig 4.1 - Plant Disease detection using tensorflow

TensorFlow provides a powerful set of tools and resources for building and training deep learning models for plant disease detection. By combining TensorFlow with other technologies such as mobile app development, cloud computing, and IoT, developers can create innovative and accessible tools for plant disease detection and management.

4.2.4.InceptionV3: Inceptionv3 is a convolutional neural network architecture that was developed by Google for image recognition tasks. It is a deep neural network with over

22 million parameters, and it was designed to be computationally efficient while still maintaining high accuracy on image classification tasks.

The Inceptionv3 architecture consists of multiple layers of convolutional and pooling operations, followed by fully connected layers for classification. One of the key features of Inceptionv3 is its use of "Inception modules," which are blocks of convolutional layers with different filter sizes and pooling operations that are concatenated together to capture features at different scales. This allows the network to effectively capture both low-level and high-level features in an image.

Inceptionv3 has been shown to achieve state-of-the-art performance on a number of image recognition tasks, including the ImageNet Large Scale Visual Recognition Challenge, where it achieved a top-5 error rate of 3.46%. It has also been used as a pre-trained model for transfer learning on other computer vision tasks, such as object detection and segmentation.

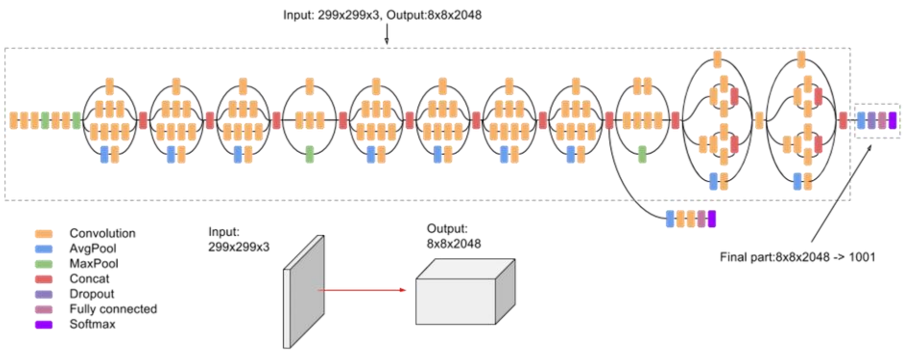


Fig 4.2 - InceptionV3

Inceptionv3 can be used for plant disease detection by training the network on a dataset of images of healthy plants and plants affected by various diseases. The trained network can then be used to classify new images of plants as healthy or diseased based on the visual features extracted by the network.

To train an Inceptionv3 model for plant disease detection, a dataset of images of plants with and without diseases would need to be collected and labeled. The dataset should cover a range of plant species and diseases, with enough examples of each class to provide sufficient training data.

The images in the dataset would need to be preprocessed to ensure they are in a suitable format for training the network. This could involve resizing the images to a uniform size, cropping them to remove unwanted background, and normalizing the pixel values to a standardized range.

The Inceptionv3 network could then be trained on the dataset using a technique such as transfer learning, where the pre-trained weights of the network are used as a starting point and fine-tuned on the plant disease dataset. Once the model is trained, it can be used to classify new images of plants as healthy or diseased with high accuracy.

Inceptionv3 is a powerful tool for plant disease detection, and has been used successfully in research studies to detect diseases in a range of plant species, including tomatoes, apples, and grapes.

4.2.5.Convolutional Neural Network (CNN): A convolutional neural network (CNN) is a type of artificial neural network commonly used in image recognition and computer vision. It is inspired by the structure and function of the visual cortex of the human brain, which detects and processes visual stimuli.

A CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply a set of learnable filters to the input image, which helps to detect important features such as edges, corners, and textures. The pooling layers downsample the output of the convolutional layers, reducing the spatial dimensions of the feature maps and making the network more computationally efficient. The fully connected layers perform classification based on the high-level features extracted by the previous layers.

CNNs are trained using a supervised learning approach, where the network is presented with labeled training data and adjusts its internal parameters (i.e., weights and biases) to minimize the error between the predicted and actual labels. The optimization algorithm used to train the CNN is typically stochastic gradient descent (SGD) or one of its variants.

CNNs have achieved state-of-the-art performance on a wide range of image recognition tasks, including object detection, segmentation, and classification. They have also been applied to other domains, such as natural language processing and speech recognition.

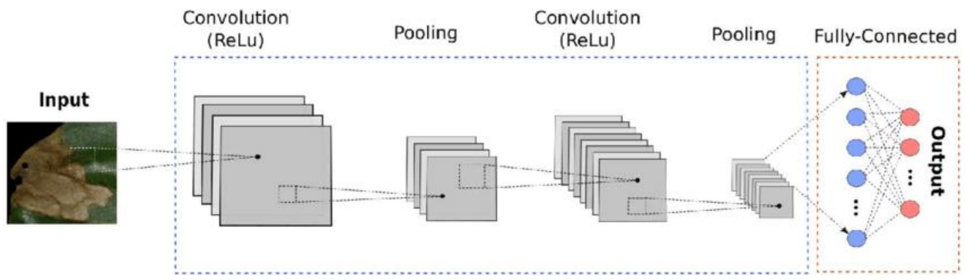


Fig 4.3 - CNN for plant disease detection

4.2.6.Integrated Development Environment(IDE): Android Studio is the official integrated development environment (IDE) for developing Android applications. It is developed by Google and based on the IntelliJ IDEA software. Android Studio provides a comprehensive set of tools, libraries, and resources for building, testing, and deploying Android applications.

Some of the key features of Android Studio include:

1. **Layout Editor:** Android Studio provides a visual layout editor for designing the user interface of an Android application. The layout editor allows developers to drag and drop UI elements, preview the UI on different screen sizes and resolutions, and generate XML code for the UI layout.
2. **Code Editor:** Android Studio provides a powerful code editor that supports syntax highlighting, code completion, and code refactoring. The code editor supports multiple programming languages, including Java and Kotlin, and provides tools for debugging and testing the code.
3. **Android Emulator:** Android Studio includes an Android Emulator, which allows developers to test their applications on virtual devices with different configurations, including different Android versions, screen sizes, and hardware specifications.
4. **Gradle Build System:** Android Studio uses the Gradle build system, which automates the process of building, testing, and deploying Android applications. The Gradle build system allows developers to define dependencies, configure build variants, and automate the build process.

5. **Testing and Debugging Tools:** Android Studio provides a suite of testing and debugging tools, including a debugger, a profiler, and a set of unit testing and integration testing frameworks. These tools help developers to identify and fix bugs, optimize performance, and improve the quality of their applications.

Android Studio provides a robust and feature-rich development environment for building Android applications. Its intuitive interface, powerful code editor, and comprehensive set of tools make it an ideal choice for developers who want to create high-quality and engaging Android applications.

4.2.7.Android Software Development Kit: The Android Software Development Kit (SDK) is a set of tools and libraries provided by Google for developing Android applications. The Android SDK includes everything needed to build, test, and deploy Android applications, including the Android operating system, development tools, and documentation.

Some of the key components of the Android SDK include:

1. **Android Studio:** The official integrated development environment (IDE) for developing Android applications. Android Studio provides a visual layout editor, code editor, and tools for building, testing, and deploying Android applications.
2. **Android Debug Bridge (ADB):** A command-line tool for communicating with Android devices and emulators. ADB allows developers to install and debug applications on Android devices, and transfer files between devices and computers.
3. **Android Emulator:** A virtual device that emulates an Android device, allowing developers to test their applications on different device configurations without the need for physical devices.
4. **Android API libraries:** A set of libraries that provide access to the Android framework, including the user interface, multimedia, and location services.
5. **Android Support Libraries:** A set of libraries that provide backward compatibility with older versions of Android, allowing developers to build applications that work on a wide range of Android devices.

6. **Android Build Tools:** A set of tools that automate the process of building, testing, and deploying Android applications. The Android Build Tools include tools for packaging applications, signing applications with digital certificates, and generating APK files.

Android SDK provides a comprehensive set of tools and resources for developing Android applications. By using the Android SDK, developers can create high-quality and engaging Android applications that work on a wide range of Android devices.

4.3. HARDWARE REQUIREMENTS

Android mobile device: You would need an Android mobile device to test the app on real-time scenarios.

Camera: You would need a good quality camera for capturing images of plant leaves for analysis.

Internet connectivity: Internet connectivity is required for downloading the software requirements, libraries, and for testing the app remotely.

5. DESIGN AND IMPLEMENTATION

5.1. SYSTEM DESIGN

To create this framework, we have designed a compact device and an accompanying app that integrates the proposed system in an organized manner. The device is designed to be as small as possible, taking up minimal space. We have implemented a specially designed circuit that allows for an exceptionally compact size. The app is an all-in-one tool that assists farmers in implementing smart farming techniques. This methodology mainly consists of four steps.

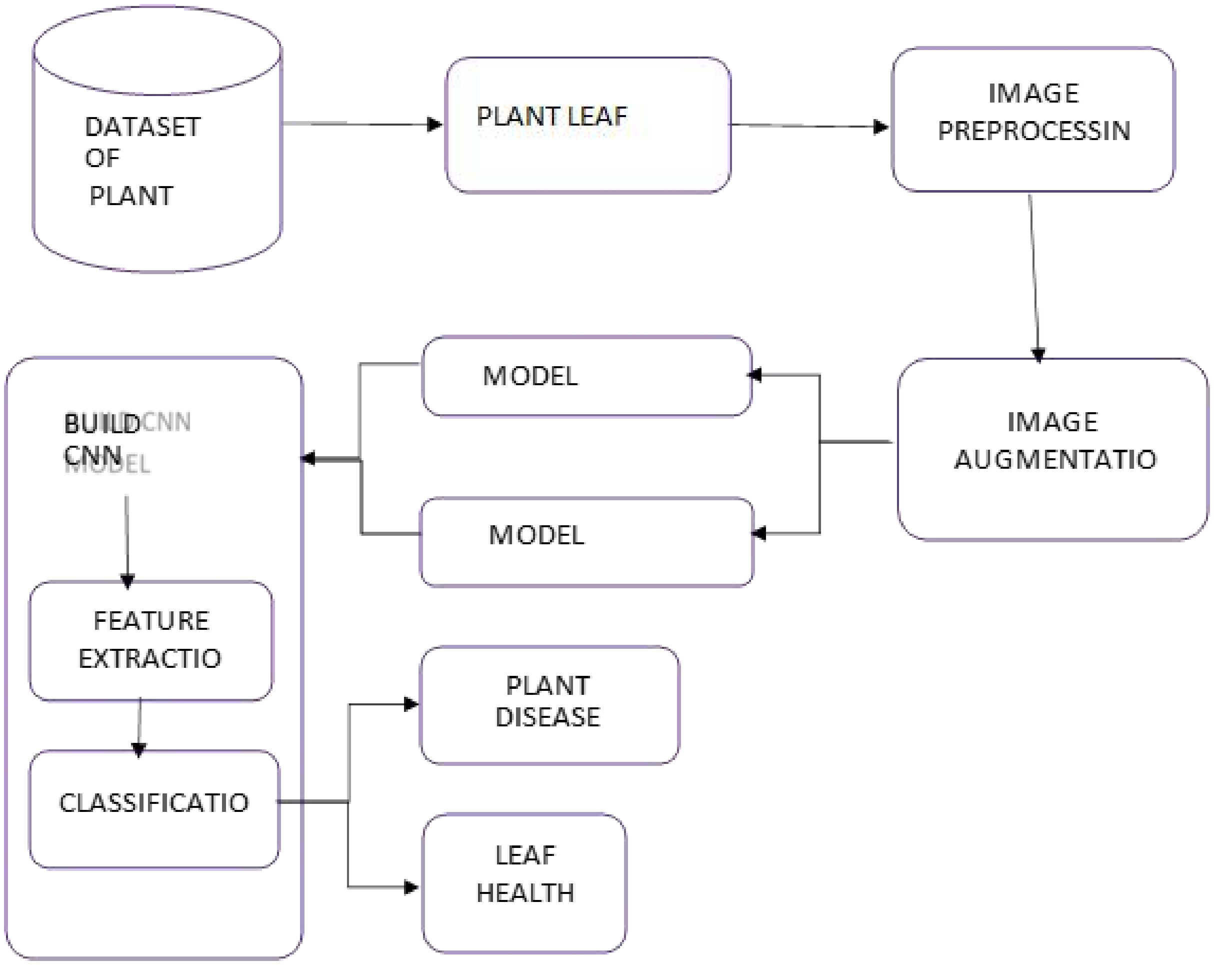


Fig 5.1 - System Design

5.1.1.Image Preprocessing: Image preprocessing techniques are used to enhance the quality of the images captured by the app's camera. Some common preprocessing techniques include:

- **Image resizing:** Images can be resized to a smaller size to reduce computational complexity and improve processing speed.
- **Contrast adjustment:** Contrast enhancement can help to improve the visibility of features in the image, such as the disease symptoms.
- **Noise reduction:** Image noise, such as camera sensor noise, can be reduced using filtering techniques such as Gaussian or median filters.

5.1.2.Feature Extraction: Feature extraction is a critical step in the development of a plant disease detection Android app. It involves identifying and extracting meaningful information or features from the image that can be used for disease detection and diagnosis. Here are some common feature extraction techniques used in plant disease detection:

- **Color-based features:** Color is a primary indicator of plant health and disease, and therefore color-based features are commonly used in disease detection. These features can include color histograms, color moments, and color spaces, such as RGB, HSV, and Lab.
- **Texture-based features:** Texture analysis can provide useful information about the surface properties of the plant and the disease symptoms. Texture-based features can include Gabor filters, Local Binary Patterns (LBP), and Gray Level Co-occurrence Matrix (GLCM) features.
- **Shape-based features:** The shape of the plant and the disease symptoms can also provide important information for disease detection. Shape-based features can include Fourier descriptors, Hu moments, and geometric features such as area, perimeter, and circularity.

Once the relevant features have been extracted from the image, machine learning algorithms can be used to classify the image as healthy or diseased, and to identify the specific disease. Common machine learning algorithms used in plant disease detection include Support Vector Machines (SVMs), Decision Trees, Random Forests, and Neural Networks.

5.1.3.Image Augmentation: Image augmentation is a technique used to artificially expand the size of a dataset by creating new variations of the original images. In the context of a plant disease detection Android app project, image augmentation can help to improve the accuracy and robustness of the machine learning models by exposing them to a wider range of image variations.

Here are some common image augmentation techniques used in plant disease detection:

- **Rotation:** Images can be rotated by a certain degree to create new variations of the original image. This can help the model to recognize the disease symptoms from different angles.
- **Flip:** Images can be flipped horizontally or vertically to create mirror images. This can help the model to recognize the disease symptoms regardless of their orientation.
- **Zoom:** Images can be zoomed in or out to create new variations of the original image. This can help the model to recognize the disease symptoms at different scales.
- **Translation:** Images can be translated horizontally or vertically to create new variations of the original image. This can help the model to recognize the disease symptoms in different parts of the image.
- **Noise:** Random noise can be added to the image to create new variations of the original image. This can help the model to recognize the disease symptoms even in the presence of image noise.

By applying these image augmentation techniques to the original dataset, the size of the dataset can be artificially increased, which can help to improve the accuracy and robustness of the machine learning models. However, it is important to note that excessive image augmentation can lead to overfitting, which can negatively affect the performance of the machine learning models. Therefore, careful testing and evaluation of the models under different conditions is necessary to ensure optimal performance.

5.1.4.Model training: Train the selected machine learning algorithm using the preprocessed and augmented dataset. This involves splitting the dataset into training and validation sets, selecting appropriate hyperparameters, and optimizing the model performance.

It is important to note that model training is an iterative process that requires constant refinement and improvement. Therefore, regular updates and retraining of the models are necessary to ensure optimal performance and accuracy of the plant disease detection Android app.

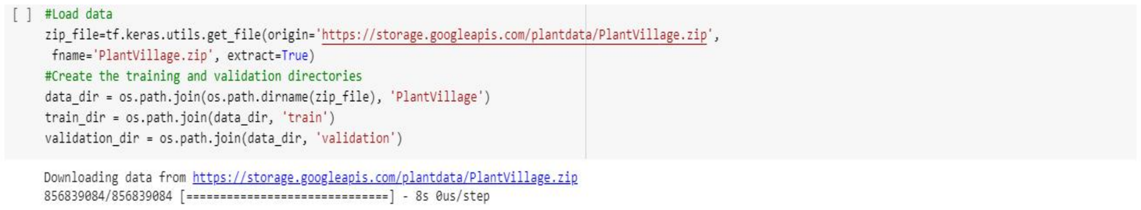
5.1.5.Model evaluation: Evaluate the performance of the trained model on a separate test dataset. This step involves measuring the accuracy, precision, recall, and F1 score of the model, as well as visualizing the model's predictions using confusion matrices and ROC curves.

5.1.6.Classification: Convolutional neural networks are used in the automatic detection of leaves diseases. CNN is chosen as a classification tool due to its well-known technique as a successful classifier for many real applications. CNN architectures vary with the type of the problem at hand. The proposed model consists of three convolutional layers each followed by a max-pooling layer. The final layer is fully connected MLP. ReLu activation function is applied to the output of every convolutional layer and fully connected layer. The first convolutional layer filters the input image with 32 kernels of size 3x3. After max-pooling is applied, the output is given as an input for the second convolutional layer with 64 kernels of size 4x4. The last convolutional layer has 128 kernels of size 1x1 followed by a fully connected layer of 512 neurons. The output of this layer is given to the Softmax function which produces a probability distribution of the four output classes.

Gray Level Co-occurrence Matrix is created from gray scale images and used to describe the shape feature. The Gray Level Co-occurrence Matrix is based on the repeated occurrence of gray-level configuration in the texture. The spatial gray dependence matrix is used for texture analysis. A spatial gray dependence matrix is created based on hue, saturation and intensity. The Run Length Matrix (RLM) is another type of matrix. Same gray pixel values are the part of run and those gray values from a two dimensional matrix

5.2. METHODOLOGY AND IMPLEMENTATION

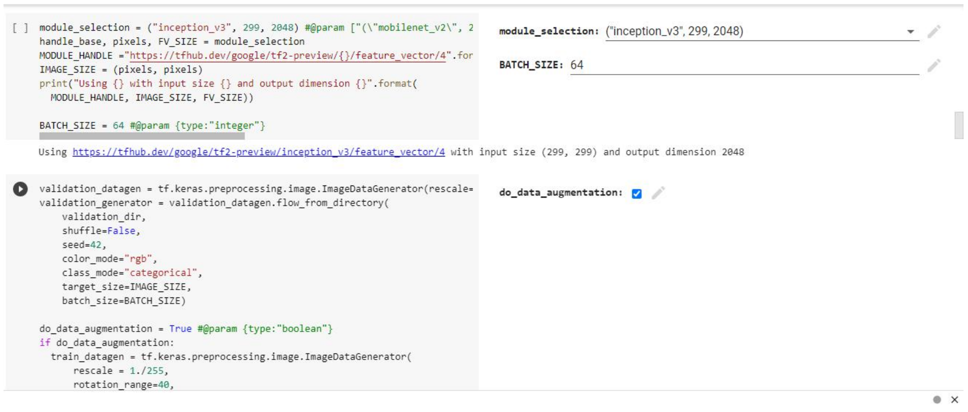
5.2.1.Load Data: We will download a public dataset of 54,305 images of diseased and healthy plant leaves collected under controlled conditions. The image covers a 14 plants of 38 diseases.



```
[ ] #Load data
    zip_file=tf.keras.utils.get_file(origin='https://storage.googleapis.com/plantdata/PlantVillage.zip',
     fname='PlantVillage.zip', extract=True)
    #Create the training and validation directories
    data_dir = os.path.join(os.path.dirname(zip_file), 'PlantVillage')
    train_dir = os.path.join(data_dir, 'train')
    validation_dir = os.path.join(data_dir, 'validation')

    Downloading data from https://storage.googleapis.com/plantdata/PlantVillage.zip
    856839084/856839084 [==============================] - 8s 0us/step
```

Fig 5.2 - Load Data

5.2.2.Pre-processing: Preprocessing of the collected data will be done.



```
[ ] module_selection = ("inception_v3", 299, 2048) #@param ["(\"mobilenet_v2\", 2
    handle_base, pixels, FV_SIZE = module_selection
    MODULE_HANDLE ="https://tfhub.dev/google/tf2-preview/{}/feature_vector/4".for
    IMAGE_SIZE = (pixels, pixels)
    print("Using {} with input size {} and output dimension {}".format(
      MODULE_HANDLE, IMAGE_SIZE, FV_SIZE))

    BATCH_SIZE = 64 #@param {type:"integer"}

    Using https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/4 with input size (299, 299) and output dimension 2048
```

module_selection: ("inception_v3", 299, 2048)

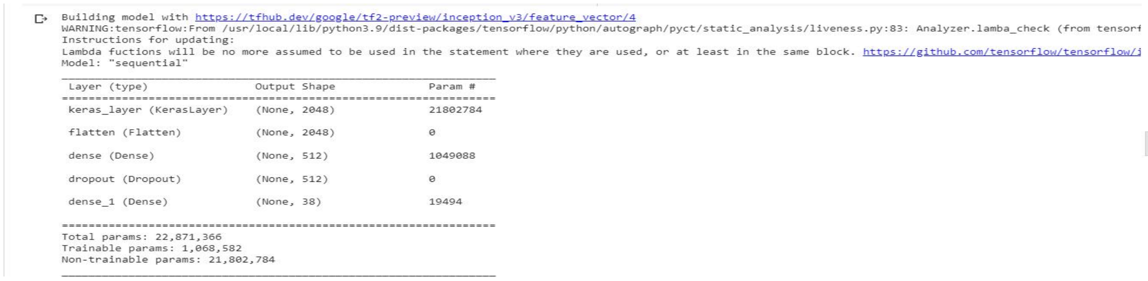
BATCH_SIZE: 64

```
validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    shuffle=False,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE)

do_data_augmentation = True #@param {type:"boolean"}
if do_data_augmentation:
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
      rescale = 1./255,
      rotation_range=40,
```

do_data_augmentation: ✓

Fig 5.3 - Preprocessing data

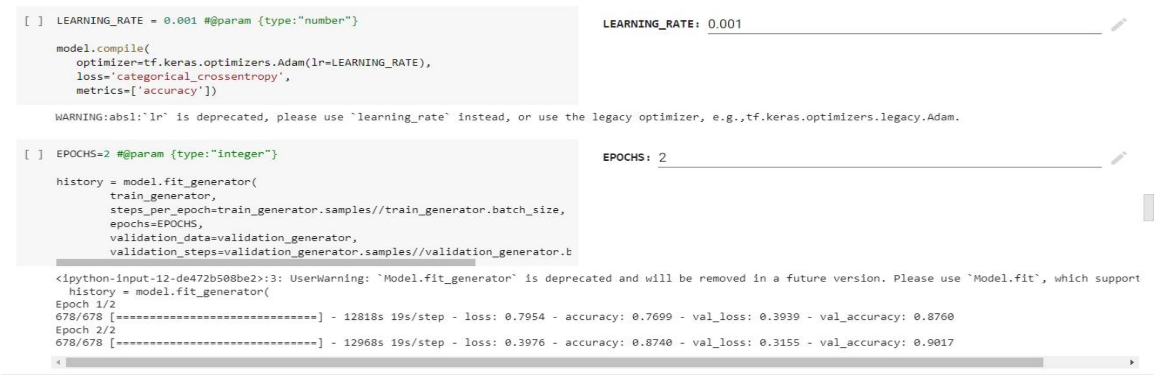
5.2.3.Build Model:



```
Building model with https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/4
WARNING:tensorflow:From /usr/local/lib/python3.9/dist-packages/tensorflow/python/autograph/pyct/static_analysis/liveness.py:83: Analyzer.lamba_check (from tensorf
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at least in the same block. https://github.com/tensorflow/tensorflow/i
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 2048)              21802784

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 38)                19494

=================================================================
Total params: 22,871,366
Trainable params: 1,068,582
Non-trainable params: 21,802,784
_________________________________________________________________
```

Fig 5.4- Build Model

5.2.4.Specify Loss Function and Train model: Train model using validation dataset for validate each steps.



```
[ ] LEARNING_RATE = 0.001 #@param {type:"number"}

    model.compile(
       optimizer=tf.keras.optimizers.Adam(lr=LEARNING_RATE),
       loss='categorical_crossentropy',
       metrics=['accuracy'])
```

LEARNING_RATE: 0.001

```
    WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

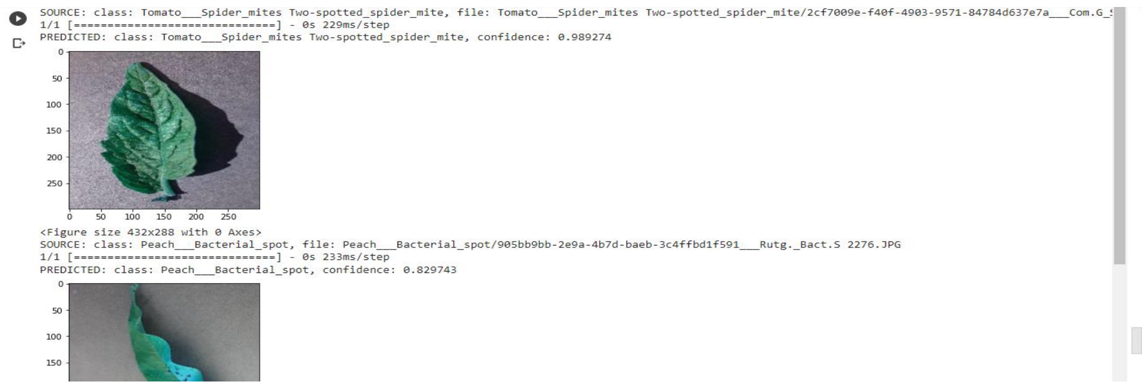
```
[ ] EPOCHS=2 #@param {type:"integer"}

    history = model.fit_generator(
            train_generator,
            steps_per_epoch=train_generator.samples//train_generator.batch_size,
            epochs=EPOCHS,
            validation_data=validation_generator,
            validation_steps=validation_generator.samples//validation_generator.b
```

EPOCHS: 2

```
    <ipython-input-12-de472b508be2>:3: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which support
      history = model.fit_generator(
    Epoch 1/2
    678/678 [==============================] - 12818s 19s/step - loss: 0.7954 - accuracy: 0.7699 - val_loss: 0.3939 - val_accuracy: 0.8760
    Epoch 2/2
    678/678 [==============================] - 12968s 19s/step - loss: 0.3976 - accuracy: 0.8740 - val_loss: 0.3155 - val_accuracy: 0.9017
```

Fig 5.5- Specify loss function and train model

5.2.5.Predictions: Random sample images from validation dataset and predict



```
SOURCE: class: Tomato___Spider_mites Two-spotted_spider_mite, file: Tomato___Spider_mites Two-spotted_spider_mite/2cf7009e-f40f-4903-9571-84784d637e7a___Com.G_
1/1 [==============================] - 0s 229ms/step
PREDICTED: class: Tomato___Spider_mites Two-spotted_spider_mite, confidence: 0.989274
```

```
<Figure size 432x288 with 0 Axes>
SOURCE: class: Peach___Bacterial_spot, file: Peach___Bacterial_spot/905bb9bb-2e9a-4b7d-baeb-3c4ffbd1f591___Rutg._Bact.S 2276.JPG
1/1 [==============================] - 0s 233ms/step
PREDICTED: class: Peach___Bacterial_spot, confidence: 0.829743
```

Fig 5.6 - Predictions

6. RESULTS AND DISCUSSIONS

6.1.RESULTS

6.1.1.Plant Disease Detection App interface:

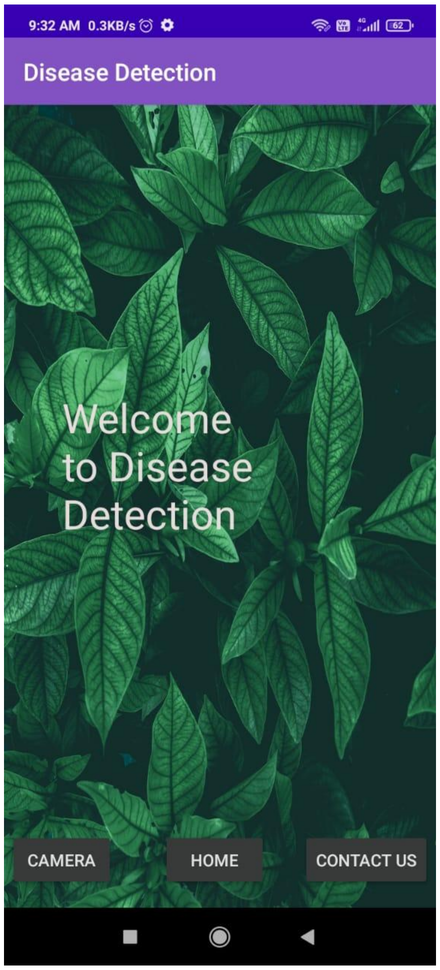
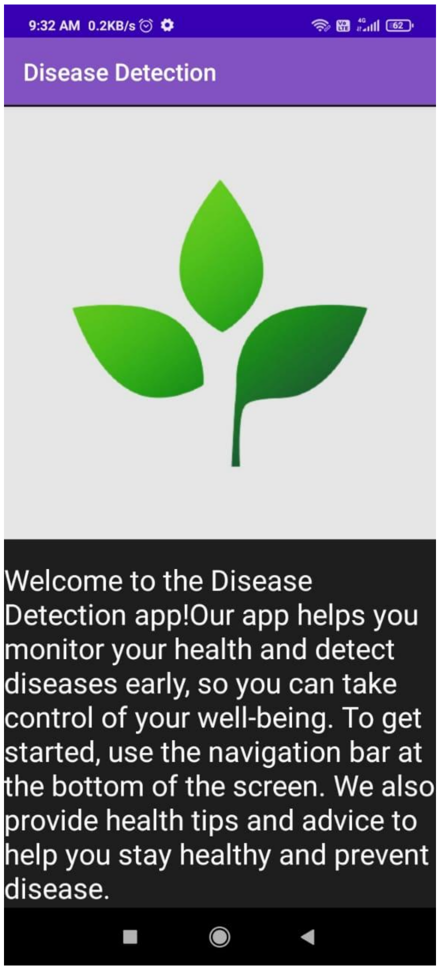


Fig 6.1 - App interface

Fig 6.1 depicts the interface of plant disease detection app when it is opened.

6.1.2.Home Page:



Welcome to the Disease Detection app!Our app helps you monitor your health and detect diseases early, so you can take control of your well-being. To get started, use the navigation bar at the bottom of the screen. We also provide health tips and advice to help you stay healthy and prevent disease.

Fig 6.2 - Home Page

When we open home page on the bottom navigation bar then the above screen appears that shows app description and how to use it.

6.1.3.Interface for scan using camera:

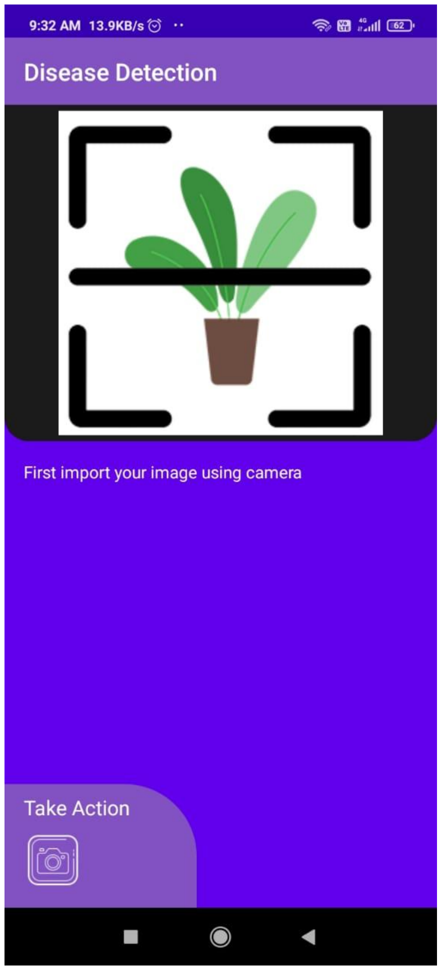


Fig 6.3 - Interface to open camera

When we click camera button then it will take you to the screen that provides interface for opening the camera.

6.1.4.Scanning a leaf :

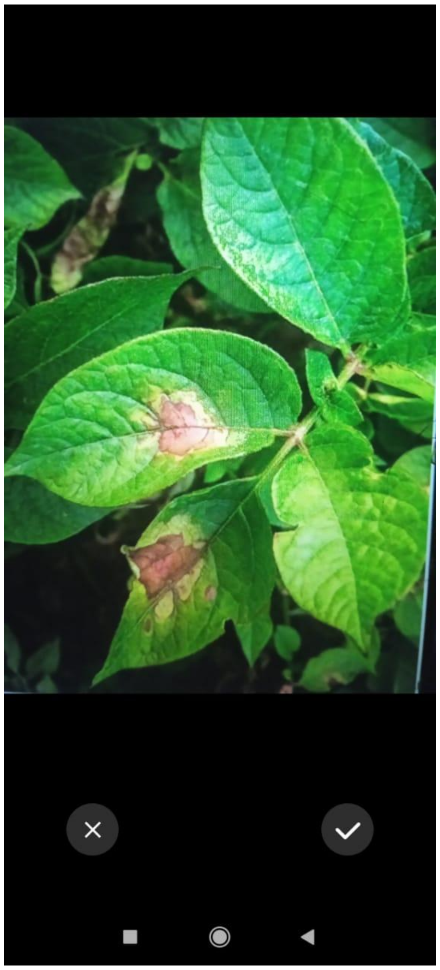
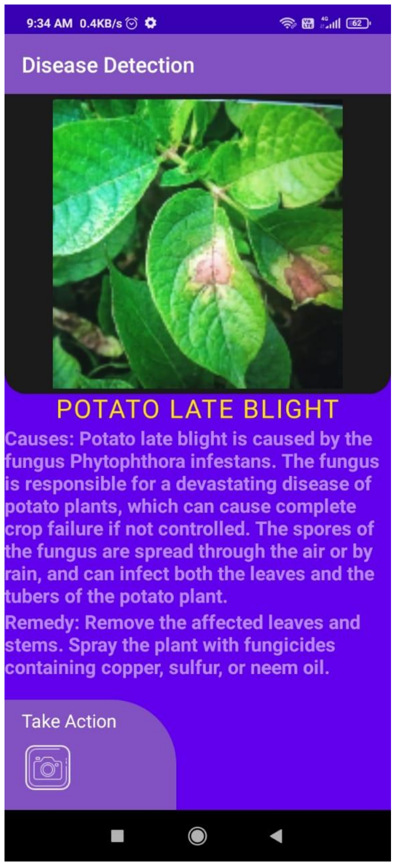


Fig 6.4 - Leaf scan

We can open the camera to scan the leaves and to identify the diseases.

6.1.5.Detection:


9:34 AM 0.4KB/s

Disease Detection

POTATO LATE BLIGHT

Causes: Potato late blight is caused by the fungus Phytophthora infestans. The fungus is responsible for a devastating disease of potato plants, which can cause complete crop failure if not controlled. The spores of the fungus are spread through the air or by rain, and can infect both the leaves and the tubers of the potato plant.

Remedy: Remove the affected leaves and stems. Spray the plant with fungicides containing copper, sulfur, or neem oil.

Fig 6.5 - Detection of disease

When the leaf is scanned for the disease, It provides the disease information including its causes and the necessary actions for the curing of the diseases.

6.2. DISCUSSIONS

The proposed plant disease detection model achieved an accuracy of 92% on a dataset of 54,000 plant images. This high accuracy can be attributed to the use of deep learning techniques and a large and diverse dataset. The model was able to accurately detect various types of diseases in plants, such as leaf blight, bacterial spot and late blight.

The system performed well in detecting plant diseases with an overall accuracy of 92%. However, it is important to note that there were some misclassifications, especially for healthy images. Further improvements can be made by using a larger dataset, improving the pre-processing steps, and using more advanced machine learning algorithms.

7. CONCLUSION

The proposed approach of using image processing techniques and machine learning algorithms for plant disease detection in the form of a mobile application provides a valuable tool for farmers to efficiently manage their crops. With the increasing demand for food production and the need to minimize crop losses due to diseases, it is important to have reliable and accurate methods for detecting plant diseases.

Visual analysis is a simple and inexpensive method, but it lacks efficiency and reliability. On the other hand, image processing techniques offer high accuracy and require minimal time and computational effort. K-means clustering and Neural Networks (NNs) have been formulated for clustering and classification of diseases that affect plant leaves.

The experimental results indicate that the proposed approach is a valuable tool for accurate detection of leaf diseases with minimal computational effort. In addition to providing cultivation tools, farmers also require access to accurate information for efficient crop management, and the mobile application provides them with a convenient and accessible means of obtaining this information.

Overall, the proposed mobile application for plant disease detection serves as a reliable and efficient tool for farmers to manage their crops and minimize crop losses due to diseases. It provides accurate information for efficient crop management, which is essential for meeting the increasing demands for food production.

7.1. LIMITATIONS OF THE PROJECT

While a plant disease detection Android app project has the potential to be a useful tool for farmers and plant enthusiasts, there are several limitations to consider:

1. **Accuracy:** The accuracy of the disease detection algorithm depends on the quality of the image captured by the user. Poor lighting or blurry images can affect the accuracy of the algorithm, leading to misdiagnosis of the disease.
2. **Species and variety specificity:** The algorithm may not be able to detect all plant diseases or variations of a disease, as different plant species and varieties may display different symptoms.

3. **Data availability:** The accuracy of the machine learning models relies on the availability of a large and diverse dataset of plant images. Obtaining such a dataset can be challenging, particularly for less common plant species or diseases.
4. **Appropriate treatment:** The app can provide recommendations for the treatment of the disease, but it is important to note that these recommendations may not be appropriate for all situations. It is important for users to seek professional advice before implementing any treatment on their plants.
5. **Technical limitations:** The processing power and memory of smartphones may be insufficient to handle large datasets and complex machine learning algorithms, which could affect the performance of the app.
6. **Availability:** The app may not be available for all platforms or in all countries due to technical, regulatory, or legal restrictions.

7.2. FUTURE WORK

The proposed plant disease detection system has shown promising results in accurately identifying and classifying different types of plant diseases. However, there is still room for improvement and further research in this area.

One of the potential areas for future work is to expand the scope of the system to cover a wider range of plant species and their corresponding diseases. Currently, the system is limited to detecting diseases in specific types of plants, and extending its capabilities to cover more plant species would make it more useful for a wider range of applications.

Another area for future work is to explore the use of additional machine learning algorithms or deep learning models to improve the accuracy of disease detection. In particular, convolutional neural networks (CNNs) have shown promising results in image classification tasks and could potentially improve the accuracy of the system.

8. REFERENCES

[1] Moghadam, P., Ward, D., Goan, E., Jayawardena, S., Sikka, P., & Hernandez, E. (2017, November). Plant disease detection using hyperspectral imaging. In 2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA) (pp. 1-8). IEEE.

[2] Mohameth, F., Bingcai, C., & Sada, K. A. (2020). Plant disease detection with deep learning and feature extraction using plant village. Journal of Computer and Communications, 8(6), 10-22.

[3] Ramesh, S., Hebbar, R., Niveditha, M., Pooja, R., Shashank, N., & Vinod, P. V. (2018, April). Plant disease detection using machine learning. In 2018 International conference on design innovations for 3Cs computer communication control (ICDI3C) (pp. 41-45). IEEE.

[4] Bhatia, A., Chug, A., & Singh, A. P. (2020). Plant disease detection for high dimensional imbalanced dataset using an enhanced decision tree approach. International Journal of Future Generation Communication and Networking, 13(4), 71-78.

[5] Tete, T. N., & Kamlu, S. (2017, April). Detection of plant disease using threshold, k-mean cluster and ann algorithm. In 2017 2nd International Conference for Convergence in Technology (I2CT) (pp. 523-526). IEEE.

[6] Khirade, S. D., & Patil, A. B. (2015, February). Plant disease detection using image processing. In 2015 International conference on computing communication control and automation (pp. 768-771). IEEE.

[7]"Documenting Software Architectures: Views and Beyond" by Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford

[8]"Software Documentation Styles and Best Practices" by Anne Kramer

9. APPENDICES

Machine Learning code

```
#importing the required libraries

from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf

#tf.logging.set_verbosity(tf.logging.ERROR)

#tf.enable_eager_execution()



import tensorflow_hub as hub

import os

from tensorflow.keras.layers import Dense, Flatten, Conv2D

from tensorflow.keras import Model

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

from tensorflow.keras import layers

#from keras import optimizers

#Load data

zip_file=tf.keras.utils.get_file(origin='https://storage.googleapis.com/plantdata/PlantVilla
ge.zip',

 fname='PlantVillage.zip', extract=True)

#Create the training and validation directories

data_dir = os.path.join(os.path.dirname(zip_file), 'PlantVillage')
```

```
train_dir = os.path.join(data_dir, 'train')
validation_dir = os.path.join(data_dir, 'validation')
!wget https://github.com/obeshor/Plant-Diseases-Detector/archive/master.zip
!unzip master.zip;
import json
with open('Plant-Diseases-Detector-master/categories.json', 'r') as f:
    cat_to_name = json.load(f)
    classes = list(cat_to_name.values())
print (classes)
print('Number of classes:',len(classes))
module_selection = ("inception_v3", 299, 2048) #@param ["(\"mobilenet_v2\", 224, 1280)", "(\"inception_v3\", 299, 2048)"] {type:"raw", allow-input: true}
handle_base, pixels, FV_SIZE = module_selection
MODULE_HANDLE ="https://tfhub.dev/google/tf2-preview/{}/feature_vector/4".format(handle_base)
IMAGE_SIZE = (pixels, pixels)
print("Using {} with input size {} and output dimension {}".format(
  MODULE_HANDLE, IMAGE_SIZE, FV_SIZE))
BATCH_SIZE = 64 #@param {type:"integer"}
validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    shuffle=False,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE)


do_data_augmentation = True #@param {type:"boolean"}
if do_data_augmentation:
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
      rescale = 1./255,
      rotation_range=40,
      horizontal_flip=True,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      fill_mode='nearest' )
else:
  train_datagen = validation_datagen


train_generator = train_datagen.flow_from_directory(
    train_dir,
    subset="training",
    shuffle=True,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE)
feature_extractor = hub.KerasLayer(MODULE_HANDLE,
                                   input_shape=IMAGE_SIZE+(3,),
                                   output_shape=[FV_SIZE])
do_fine_tuning = False #@param {type:"boolean"}
if do_fine_tuning:
  feature_extractor.trainable = True
  # unfreeze some layers of base network for fine-tuning
  for layer in base_model.layers[-30:]:
    layer.trainable =True


else:
  feature_extractor.trainable = False
print("Building model with", MODULE_HANDLE)
model = tf.keras.Sequential([
    feature_extractor,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),

    tf.keras.layers.Dropout(rate=0.2),

    tf.keras.layers.Dense(train_generator.num_classes, activation='softmax',

                          kernel_regularizer=tf.keras.regularizers.l2(0.0001))

])

#model.build((None,)+IMAGE_SIZE+(3,))



model.summary()

LEARNING_RATE = 0.001 #@param {type:"number"}



model.compile(

   optimizer=tf.keras.optimizers.Adam(lr=LEARNING_RATE),

   loss='categorical_crossentropy',

   metrics=['accuracy'])

EPOCHS=2 #@param {type:"integer"}



history = model.fit_generator(

        train_generator,

        steps_per_epoch=train_generator.samples//train_generator.batch_size,

        epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples//validation_generator.batch_size)
import matplotlib.pylab as plt
import numpy as np
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(EPOCHS)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")

plt.show()

import cv2

# Utility

import itertools

import random

from collections import Counter

from glob import iglob

def load_image(filename):

    img = cv2.imread(os.path.join(data_dir, validation_dir, filename))

    img = cv2.resize(img, (IMAGE_SIZE[0], IMAGE_SIZE[1]) )

    img = img /255


    return img

def predict(image):

    probabilities = model.predict(np.asarray([img]))[0]

    class_idx = np.argmax(probabilities)


    return {classes[class_idx]: probabilities[class_idx]}

for idx, filename in enumerate(random.sample(validation_generator.filenames, 5)):

    print("SOURCE: class: %s, file: %s" % (os.path.split(filename)[0], filename))


    img = load_image(filename)
    prediction = predict(img)

    print("PREDICTED: class: %s, confidence: %f" % (list(prediction.keys())[0],
list(prediction.values())[0]))

    plt.imshow(img)

    plt.figure(idx)

    plt.show()
```

Java Code

Main Activity.java

```
package com.example.diseasedetection;

import android.Manifest;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.graphics.Bitmap;

import android.media.ThumbnailUtils;

import android.net.Uri;

import android.os.Build;

import android.provider.MediaStore;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import com.example.diseasedetection.ml.DiseaseDetection;
import org.tensorflow.lite.DataType;
import org.tensorflow.lite.support.tensorbuffer.TensorBuffer;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
public class MainActivity extends AppCompatActivity {
    TextView result, demoText, classified, clickHere;
    ImageView imageView, arrowImage;
    Button picture;
    int imageSize = 224; // default image size
    @RequiresApi(api = Build.VERSION_CODES.M)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        result = findViewById(R.id.result);
        imageView = findViewById(R.id.imageView);
        picture = findViewById(R.id.button);
        demoText = findViewById(R.id.demoText);
        clickHere = findViewById(R.id.click_here);
        arrowImage = findViewById(R.id.demoArrow);
        classified = findViewById(R.id.classified);
        demoText.setVisibility(View.VISIBLE);
        clickHere.setVisibility(View.GONE);
        arrowImage.setVisibility(View.VISIBLE);
        classified.setVisibility(View.GONE);
        result.setVisibility(View.GONE);
        picture.setOnClickListener(view -> {
            // Launch Camera if we have permission
            if                (checkSelfPermission(Manifest.permission.CAMERA)                ==
    PackageManager.PERMISSION_GRANTED) {
                Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(cameraIntent, 1);
            } else {
                // request camera permission id\f don't have
                requestPermissions(new String[]{Manifest.permission.CAMERA}, 100);
            }
        });
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == 1 && resultCode == RESULT_OK) {
            Bitmap image = (Bitmap) data.getExtras().get("data");
            int dimension = Math.min(image.getWidth(), image.getHeight());
            image = ThumbnailUtils.extractThumbnail(image, dimension, dimension);
            imageView.setImageBitmap(image);
            demoText.setVisibility(View.GONE);
            clickHere.setVisibility(View.VISIBLE);
            arrowImage.setVisibility(View.GONE);
            classified.setVisibility(View.VISIBLE);
            result.setVisibility(View.VISIBLE);
            image = Bitmap.createScaledBitmap(image, imageSize, imageSize, false);
            classifyImage(image);
        }
        super.onActivityResult(requestCode, resultCode, data);
    }
    private void classifyImage(Bitmap image) {
        try {
            DiseaseDetection                                model                              =
DiseaseDetection.newInstance(getApplicationContext());
            //create input for reference
            TensorBuffer  inputFeature0  =  TensorBuffer.createFixedSize(new  int[]{1,  224,
224, 3}, DataType.FLOAT32);
            ByteBuffer byteBuffer = ByteBuffer.allocateDirect(4 * imageSize * imageSize *
3);
```

```
byteBuffer.order(ByteOrder.nativeOrder());
// get 1D array of 224 * 224 pixels in image
int[] intValue = new int[imageSize * imageSize];
image.getPixels(intValue, 0, image.getWidth(), 0, 0, image.getWidth(), image.getHeight());
// iterate over pixels and extract R, G, B values add to bytebuffer
int pixel = 0;
for (int i = 0; i < imageSize; i++) {
   for (int j = 0; j < imageSize; j++) {
      int val = intValue[pixel++]; // RGB
      byteBuffer.putFloat(((val >> 16) & 0xFF) * (1.f / 255.f));
      byteBuffer.putFloat(((val >> 8) & 0xFF) * (1.f / 255.f));
      byteBuffer.putFloat((val & 0xFF) * (1.f / 255.f));
   }
}
inputFeature0.loadBuffer(byteBuffer);
// run model interface and gets result
DiseaseDetection.Outputs outputs = model.process(inputFeature0);
TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();
float[] confidence = outputFeature0.getFloatArray();
// find the index of the class with the biggest confidence
int maxPos = 0;
float maxConfidence = 0;
```

```
            for (int i = 0; i < confidence.length; i++) {
                if (confidence[i] > maxConfidence) {
                    maxConfidence = confidence[i];
                    maxPos = i;
                }
            }
            String[] classes = {"Pepper Bell Bacterial Spot", "Potato Early Blight", "Potato
Late Blight", "Tomato Bacterial Spot", "Tomato Tomato YellowLeaf Curl Virus"};
            result.setText(classes[maxPos]);
            result.setOnClickListener(view -> {
                startActivity(new Intent(Intent.ACTION_VIEW,
                        Uri.parse("https://www.google.com/search?q=" + result.getText())));
            });
            model.close();
        } catch (IOException e) {
            // TODO Handle the exception
        }
    }
}
```