

“PLANT DISEASE IDENTIFICATION”

A PROJECT REPORT

SUBMITTED BY

DEEP KARMAKAR (27800121046)

ANSHU KUMAR SINGH (27800120009)

RAJA MONDAL (27800121060)

SOUMYADEEP MAJI (27800120001)

Under the guidance of

Dr. Amit Kumar

(Assistant Professor, CSE)

DEPT. OF COMPUTER SCIENCE AND ENGINEERING



SANAKA EDUCATIONAL TRUST'S GROUP OF
INSTITUTIONS
MALANDIGHI, DURGAPUR

In the partial fulfillment for the award of the degree of
BACHELOR OF TECHNOLOGY
Of



MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY,
Salt Lake City, Sector- I, West Bengal-700064, INDIA

ACKNOWLEDGEMENT

We wish to express our thanks and gratitude to our project guide **Dr. AMIT KUMAR (Guide)** for being a constant source of encouragement inspiration and motivation.

Our profound gratitude goes to the **Head of the Department, Dr. SANKAR MUKHERJEE** for his support and guidance throughout the project.

We would also like to thank **Mr. AMIT CHAKRABORTY(Faculty of the C.S.E. department)** for helping us during our project.

Our special thanks goes to the teaching and non-teaching staff of computer science and engineering department for their support.

Finally we would also like to take the opportunity to thank all our friends, classmates who have helped us in gathering all the relevant information regarding this project.

Place:

Durgapur

Project Members

DEEP KARMAKAR
ANSHU SINGH
RAJA MONDAL
SOUMYADEEP MAJI



Sanaka Educational Trust's Group of Institutions
Vill + P.O- Malandighi, P.S-Kanksha, Durgapur-12

CERTIFICATE

This is to certify that the project report entitled **PLANT DISEASE IDENTIFICATION** submitted by
DEEP KARMAKAR (27800121046)
ANSHU KUMAR SINGH (27800120009)
RAJA MONDAL (27800121060)
SOUMYADEEP MAJI (27800120001)
to the Dept. of Computer Science and Engineering, Sanaka Educational Trust's Group of Institutions, has been submitted under the supervision of **Dr. Amit Kumar**. This report is for partial fulfilment of the requirement of VIII Semester, B. Tech in Computer Science and Engineering from SETGOI.

(Signature)

Supervisor

Dept. of CSE

(Signature)

HOD

Dept. of CSE

SETGOI, DURGAPUR

SETGOI, DURGAPUR

CONTENTS

	TOPIC	PAGE NO .
1	ABSTRACT	06
2	INTRODUCTION	07
2.1	GENERAL OVERVIEW OF THE PROBLEM	08
2.2	PROBLEM DEFINITION	09
2.3	ANALYSIS OF THE PROBLEM	09 - 11
2.4	PROPOSED SOLUTION STRATEGY	11 - 12
3	SOFTWARE REQUIREMENT SPECIFICATION	13 - 16
4	DESIGN	17
4.1	DATA FLOW DIAGRAM	18
5	SNAPSHOTS OF CODE	19 – 26
6	CONCLUSION	27
7	REFERENCES	28

ABSTRACT

In this report, we describe the design and implementation of a web-application for Plant Disease Identification for an engineering college. This web-application is named as **DETECT THE DISEASE**, because it can help people to see needful information about various diseases in crop species like potato, tomato, etc.

This project applied software engineering techniques and principles throughout the programming process. We designed and implemented this decision-making program by getting college requirement, based on the information of experience on statistic data analysis, and finally achieve the desired result. This makes the system more intelligent and flexible.

The "**Plant Disease Identification**" project aims to tackle the critical issue of identifying and diagnosing diseases in plants using cutting-edge technology. In the agricultural sector, plant diseases can result in significant crop losses and impact food production, making it crucial to address this challenge effectively.

To address this issue, we are leveraging the power of Python programming language and employing machine learning techniques to develop a robust and efficient disease detection system. Our focus is primarily on detecting diseases in plants based on images of their leaves or other affected parts. Here's an outline of our project's key aspects.

INTRODUCTION

GENERAL OVERVIEW OF THE PROBLEM:

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. Using a public dataset of 2152 images of diseased and healthy plant leaves collected under controlled conditions, we train a deep convolutional neural network to identify 1 crop species and 2 diseases (or absence thereof). The trained model achieves an accuracy of more than 91% on a held-out test set, demonstrating the feasibility of this approach. 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.

Modern technologies have given human society the ability to produce enough food to meet the demand of more than 7 billion people. However, food security remains threatened by a number of factors including climate change (Tai et al., 2014), the decline in pollinators (Report of the Plenary of the Intergovernmental Science-Policy Platform on Biodiversity Ecosystem and Services on the work of its fourth session, 2016), plant diseases (Strange and Scott, 2005), and others. Plant diseases are not only a threat to food security at the global scale, but can also have disastrous consequences for smallholder farmers whose livelihoods depend on healthy crops. In the developing world, more than 80 percent of the agricultural production is generated by smallholder farmers (UNEP, 2013), and reports of yield loss of more than 50% due to pests and diseases are common (Harvey et al., 2014). Furthermore, the largest fraction of hungry people (50%) live in smallholder farming households, making smallholder farmers a group that's particularly vulnerable to pathogen-derived disruptions in food supply.

Various efforts have been developed to prevent crop loss due to diseases. Historical approaches of widespread application of pesticides have in the past decade increasingly been supplemented by integrated pest management (IPM) approaches. Independent of the approach, identifying a disease correctly when it first appears is a crucial step for efficient disease management.

PROBLEM DEFINITION:

The primary challenge in plant disease detection is to create reliable systems that can accurately differentiate between healthy plants and those affected by various diseases. This entails developing techniques that can analyze visual or sensor data to identify symptoms, patterns, or anomalies associated with specific diseases. These methods must be adaptable to different plant species, environmental conditions, and types of diseases. Types Of Classification: i. Healthy ii. Unhealthy

We are predicting the accuracy of the above two. We are initiating with Potato Disease Detection

Early Blight



Late Blight



Challenges in Plant Disease Detection:

Early Detection: Identifying diseases at an early stage is crucial for effective management. Traditional methods may not detect subtle symptoms until they become severe.

Variability: Plant diseases can manifest differently based on factors such as environmental conditions, pathogens, and plant species, making accurate detection challenging.

Labor Intensiveness: Manual visual inspection is timeconsuming, requires skilled personnel, and may not be feasible for large-scale cultivation.

ANALYSIS OF THE PROBLEM:

Historically, disease identification has been supported by agricultural extension organizations or other institutions, such as local plant clinics. In more recent times, such efforts have additionally been supported by providing information for disease diagnosis online, leveraging the increasing Internet penetration worldwide. Even more recently, tools based on mobile phones have proliferated, taking advantage of the historically

unparalleled rapid uptake of mobile phone technology in all parts of the world (ITU, 2015).

Smartphones in particular offer very novel approaches to help identify diseases because of their computing power, high-resolution displays, and extensive built-in sets of accessories, such as advanced HD cameras. It is widely estimated that there will be between 5 and 6 billion smartphones on the globe by 2020. At the end of 2015, already 69% of the world's population had access to mobile broadband coverage, and mobile broadband penetration reached 47% in 2015, a 12-fold increase since 2007 (ITU, 2015).

The analysis of the problem of plant disease detection involves a comprehensive understanding of the challenges, methodologies, and implications associated with developing effective disease detection systems. Here, we'll delve into the key aspects of this analysis: 1. Importance of Early Detection: Early detection of plant diseases is crucial to prevent widespread outbreaks and minimize crop losses. Delayed or inaccurate detection can lead to reduced yields, decreased quality of produce, and economic losses for farmers and the agricultural industry. 2. Complexity and Variability: Plant diseases can exhibit diverse symptoms, making accurate detection challenging. The appearance of symptoms can vary based on factors such as plant species, environmental conditions, and the specific pathogen causing the disease. 3. Traditional vs. Modern Approaches: Historically, plant diseases were identified through manual visual inspection by experienced agronomists. However, this approach is time-consuming, labor-intensive, and subjective. Modern technology, including computer vision, machine learning, and sensor networks, provides more efficient and objective alternatives. Computer vision, and object recognition in particular, has made tremendous advances in the past few years. The PASCAL VOC Challenge (Everingham et al., 2010), and more recently the Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015) based on the ImageNet dataset (Deng et al., 2009) have been widely used as benchmarks for numerous visualization-related problems in computer vision, including object classification. In 2012, a large, deep convolutional neural network achieved a top-5 error of 16.4% for the classification of images into 1000 possible categories (Krizhevsky et al., 2012). In the following 3 years, various advances in deep convolutional neural networks lowered the error rate to 3.57% (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Zeiler and Fergus, 2014; He et al., 2015; Szegedy et al., 2015). While training large neural networks can be very time-consuming, the trained models can classify images very quickly, which makes them also suitable for consumer applications on smartphones.

Deep neural networks have recently been successfully applied in many diverse domains as examples of end to end learning. Neural networks provide a mapping between an input—such as an image of a diseased plant—to an output—such as a crop~disease pair. The nodes in a neural network are mathematical functions that take numerical inputs from the incoming edges, and provide a numerical output as an outgoing edge. Deep neural networks are simply mapping the input layer to the output layer over a series of stacked layers of nodes. The challenge is to create a deep network in such a way that both the structure of the network as well as the functions (nodes) and edge weights correctly map the input to the output. Deep neural networks are trained by tuning the network parameters in such a way that the mapping improves during the training process. This process is computationally challenging and has in recent times been improved dramatically by a number of both conceptual and engineering breakthroughs.

PROPOSED SOLUTION AND STRATEGY

The proposed solution strategy for plant disease detection involves a combination of advanced technologies, data-driven methodologies, and userfriendly interfaces to address the challenges of timely and accurate disease identification in crops. Here, we outline the key components of the solution strategy:

1. Data Collection and Curation: Gather a diverse dataset of plant images representing various species, diseases, and growth stages. Ensure high-quality images by addressing issues such as lighting variations, angles, and image resolution. Annotate the images with accurate disease labels to create a well-labeled training dataset.

2. Data Preprocessing and Augmentation: Apply preprocessing techniques to clean, normalize, and enhance the dataset. Augment the dataset by generating variations of images through techniques like rotation, flipping, and scaling. Balance the dataset to prevent biases toward certain classes of diseases.

3. Deep Learning Model Development: Design a Convolutional Neural Network (CNN) architecture suitable for image classification tasks. Utilize pre-trained CNN models as a starting point for transfer learning, leveraging their learned features. Customize the architecture for the specific problem of

plant disease detection, including the number of layers, filter sizes, and activation functions.

4. Training and Optimization: Split the dataset into training, validation, and testing subsets. Train the CNN model using the training data and finetune hyperparameters to achieve optimal performance. Use validation data to monitor the model's progress and prevent overfitting by adjusting regularization techniques.

5. Real-time Detection Application: Develop a user-friendly mobile application that allows users (farmers, agronomists) to capture plant images using their smartphones. Integrate the trained CNN model into the application to perform real-time disease detection and classification. Provide instant feedback on the health status of the plant and recommend appropriate actions.

In addition, traditional approaches to disease classification via machine learning typically focus on a small number of classes usually within a single crop. Examples include a feature extraction and classification pipeline using thermal stereo images. Our approach is based on supervised training using a deep convolutional neural network architecture is a practical possibility even for image classification problems with a very large number of classes, beating the traditional approaches using hand-engineered features by a substantial margin in standard benchmarks. Using the deep convolutional neural network architecture, we trained a model on images of plant leaves with the goal of classifying the presence and identity of disease on images that the model had not seen before. Within the PlantVillage data set of 2152 images containing 1 crop species and 2 diseases (or absence thereof), this goal has been achieved as demonstrated by the top accuracy of 91.00%.

SOFTWARE REQUIREMENT SPECIFICATIONS

This SRS contains a user level description of the project, along with a detailed list of prioritized requirements.

SYSTEM INTERFACES

FAST API : FastAPI is a modern, high-performance web framework for building APIs with Python. It's designed to be easy to use, highly efficient, and type-safe. FastAPI leverages Python 3.7+'s type hints for automatic validation, serialization, and documentation of API endpoints. It's built on top of Starlette for the web parts and Pydantic for data validation and serialization, providing fast execution speed due to its asynchronous nature. FastAPI's auto-generated interactive API documentation using Swagger UI or ReDoc makes it convenient for developers to understand and test APIs.

REACT JS : React.js is an open-source JavaScript library used for building user interfaces, particularly for web applications. Developed by Facebook, React allows developers to create reusable UI components that manage their own state and can be combined to build complex interfaces. Its key feature is the ability to efficiently update and render components when data changes, optimizing performance by using a virtual DOM to minimize actual DOM manipulation. React uses a declarative approach, where developers describe how the UI should look based on the application's state, making it easier to manage and debug large applications.

SOFTWARE INTERFACES

VS CODE: Visual Studio Code (VSCode) is a free, open-source code editor developed by Microsoft. It's known for its versatility, speed, and wide range of features that cater to developers across various programming languages and frameworks. VSCode offers an intuitive user interface with powerful editing capabilities, including syntax highlighting, code completion, debugging support, and an extensive library of extensions that can be easily installed to enhance its functionality. It's highly customizable, allowing users to personalize their workspace with themes, settings, and extensions to suit their specific workflow preferences. Its built-in Git integration and collaboration tools further contribute to its popularity among developers for building and managing different types of projects.

PYCHARM: PyCharm is a powerful integrated development environment (IDE) specifically designed for Python programming. Created by JetBrains, it offers a robust set of features to streamline the development process. PyCharm provides a user-friendly interface with code analysis, debugging, and intelligent code completion capabilities. It supports various frameworks, such as Django, Flask, and scientific libraries like NumPy and pandas. The IDE also includes version control integration, extensive plugin support,

and tools for testing, profiling, and web development, making it a popular choice among Python developers.

DOCKER: Docker is a platform used for containerization, which allows us to package applications and their dependencies into standardized units called containers. These containers are isolated environments that run on any machine, making it easier to build, deploy, and manage applications across different computing environments, such as development, testing, and production. Docker simplifies the process of software development by ensuring consistency between environments and enabling greater efficiency in deploying and scaling applications.

FLUTTER: Flutter is a framework for building natively compiled applications for mobile, web, and desktop from a single codebase. Here's a brief overview of how Flutter is used to develop an application:

- 1. Setup:** Install Flutter SDK and set up an editor like Visual Studio Code or Android Studio, along with necessary plugins.
- 2. Project Creation:** Use the Flutter command-line tool to create a new project with `flutter create project_name`.
- 3. Development:** Write Dart code to define the application's UI and behavior. Flutter uses widgets as the building blocks for the UI. You can use pre-designed widgets or create custom ones.
- 4. Hot Reload:** Use the hot reload feature to instantly see changes in the app's UI during development, making the process more efficient.
- 5. State Management:** Manage the state of the application using various approaches like `setState`, `Provider`, `Riverpod`, or `Bloc`.
- 6. Testing:** Write unit, widget, and integration tests to ensure the application works correctly.
- 7. Building:** Compile the application for the desired platform(s). Flutter supports building for iOS, Android, web, Windows, macOS, and Linux.
- 8. Deployment:** Deploy the app to app stores, web servers, or other platforms as needed. Flutter's single codebase approach streamlines development and maintenance across multiple platforms, making it a popular choice for developers.

HOW FLUTTER HAS BEEN USED HERE

1. Setup:

- Install Flutter SDK: Download and install the Flutter SDK from the official Flutter website (<https://flutter.dev/>).
- Set up Editor: Install an Integrated Development Environment (IDE) such as Visual Studio Code or Android Studio, and add the Flutter and Dart plugins.
- Environment Configuration: Ensure that the `PATH` variable includes the Flutter SDK.

2. Project Creation:

- Create New Project: Open a terminal and run the command `flutter create project_name`. This generates a new Flutter project with a predefined directory structure.
- Project Structure: Familiarize yourself with the key folders and files such as `lib/main.dart` (the entry point for the app), `pubspec.yaml` (for dependencies), and `test` (for tests).

3. Development:

- Writing Code: Use Dart to write your application code. Dart is the programming language used by Flutter.
- UI Design: Build the UI using Flutter's rich set of pre-designed widgets. Widgets are the basic building blocks of a Flutter app. They can be combined to create complex UIs.
- Custom Widgets: Create custom widgets by extending existing ones or composing multiple widgets together.

4. Hot Reload:

- Instant Feedback: Use Flutter's hot reload feature to instantly see the results of code changes without restarting the app. This greatly speeds up the development process.

5. State Management:

- Simple State Management: For simple state management, use `setState()` to update the UI.
- Advanced State Management: For more complex state management, use packages like `Provider`, `Riverpod`, `Bloc`, or `Redux`. These help manage state across the entire application efficiently.

6. Testing:

- Unit Tests: Write unit tests to test individual functions or classes.
- Widget Tests: Write widget tests to verify the UI components.
- Integration Tests: Write integration tests to test the app as a whole, simulating user interactions and ensuring the app works end-to-end.

7. Building:

- Build for Development: Use `flutter run` to build and run the app on an emulator or physical device for development.
- Build for Production: Use `flutter build` followed by the target platform (e.g., `flutter build apk` for Android, `flutter build ios` for iOS) to generate the production-ready build of the app.

8. Deployment:

- Deploy to App Stores: Follow the platform-specific guidelines to deploy your app to the Google Play Store or Apple App Store.
- Deploy to Web: Use `flutter build web` to create a web version of your app and deploy it to a web server.
- Deploy to Desktop: Flutter also supports building for desktop platforms like Windows, macOS, and Linux. Use `flutter build` commands for these platforms to create desktop apps.

Throughout the development process, Flutter provides a rich ecosystem of packages and plugins available on pub.dev (<https://pub.dev/>), which can be used to extend functionality such as accessing device features, implementing navigation, and more. This makes Flutter a versatile and powerful framework for cross-platform application development.

MEMORY CONSTRAINTS

This mainly depends on the operating system and web browser in use; however, the average memory needed for low-end systems should be around 2 GB RAM and 256 GB Hardisk.

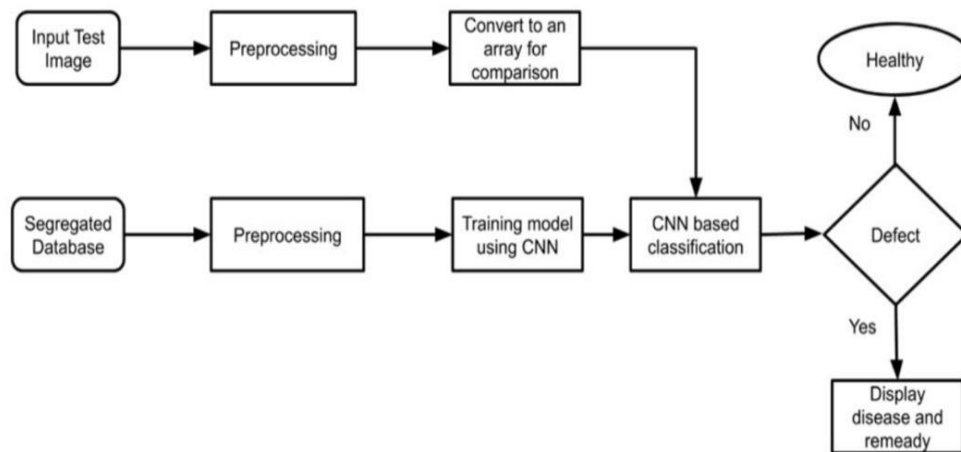




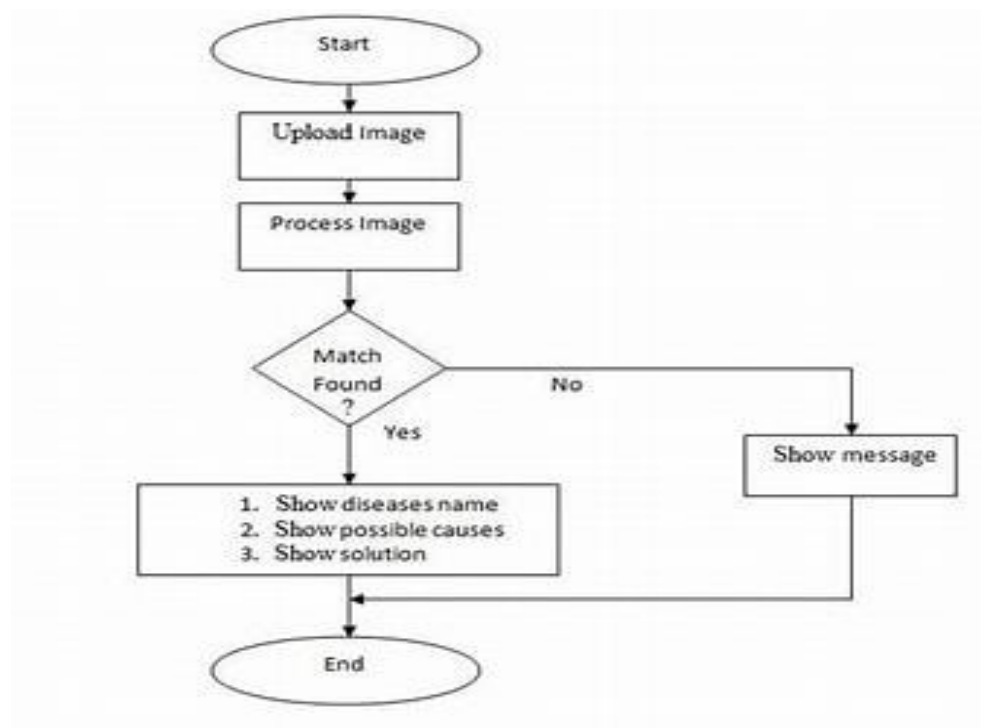
Design

DATA FLOW DIAGRAM

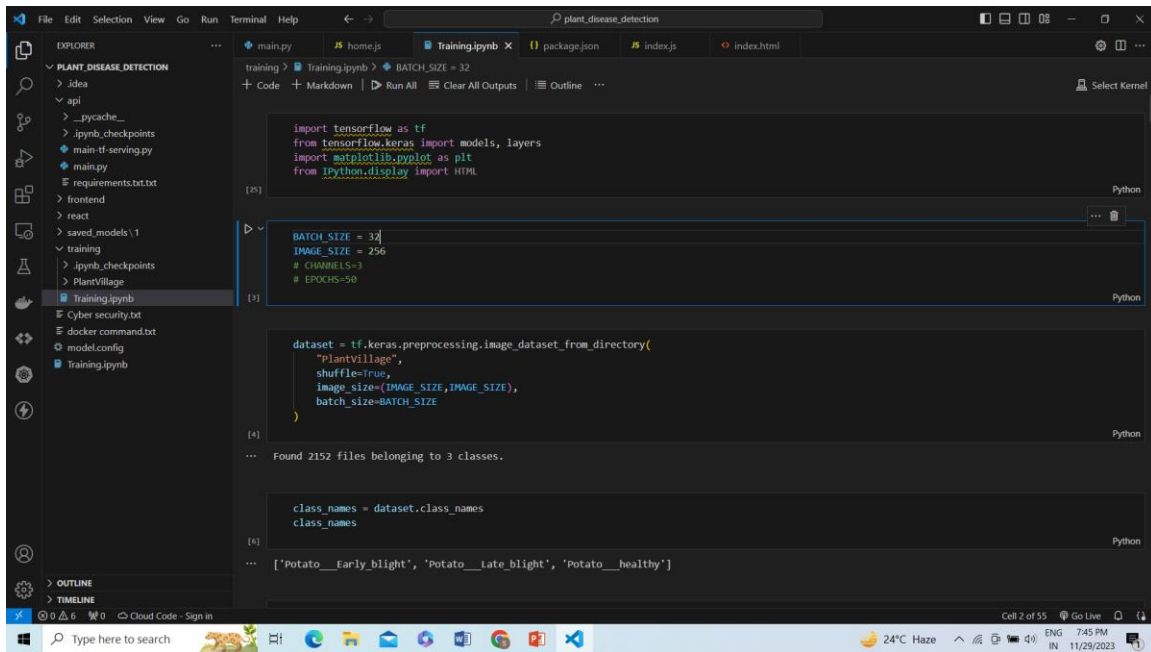
CONTEXT LEVEL DIAGRAM



Dig. 2 Flow chart diagram



SNAPSHOTS OF PROJECT



```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML

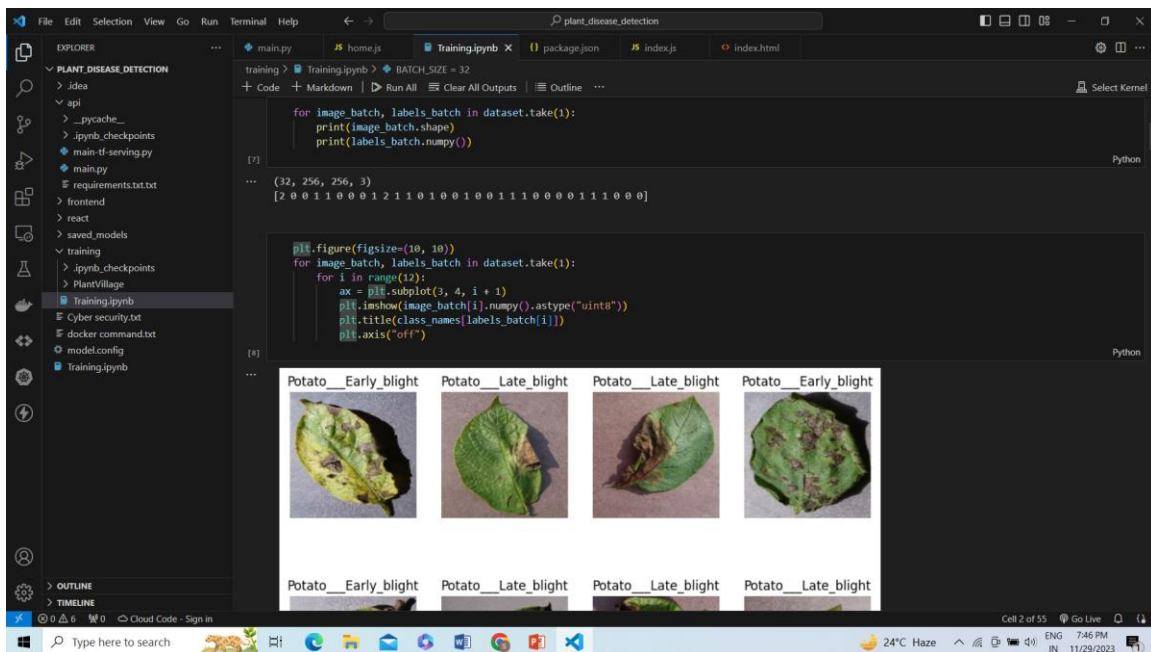
BATCH_SIZE = 32
IMAGE_SIZE = 256
# CHANNELS=3
# EPOCHS=50

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

Found 2152 files belonging to 3 classes.

class_names = dataset.class_names
class_names

['Potato__Early_blight', 'Potato__Late_blight', 'Potato__healthy']
```

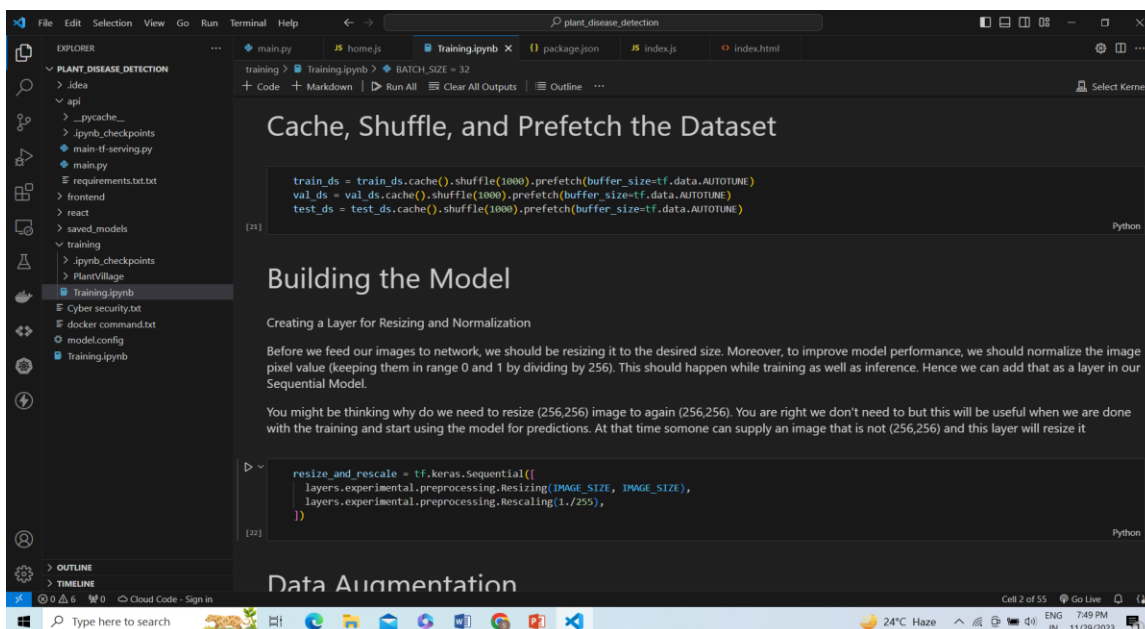
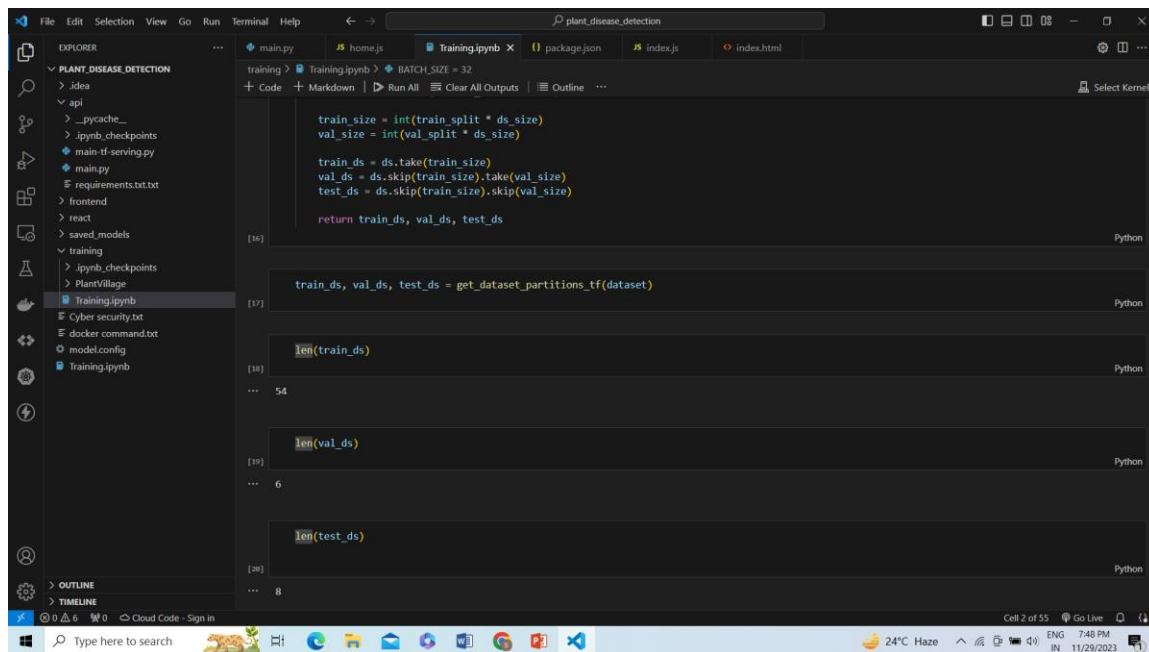


```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())

(32, 256, 256, 3)
[2 0 0 1 1 0 0 0 1 2 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0]

plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")

Potato__Early_blight Potato__Late_blight Potato__Late_blight Potato__Early_blight
Potato__Early_blight Potato__Late_blight Potato__Late_blight Potato__Late_blight
```



The screenshot shows a Jupyter Notebook in a web browser. The left sidebar contains a file explorer for a project named 'PLANT_DISEASE_DETECTION'. The main area displays the 'Data Augmentation' section of the notebook. It includes a text block explaining that data augmentation is needed for accuracy, followed by a code cell defining a data augmentation pipeline using Keras layers for random flip and rotation. A second code cell shows how to apply this augmentation to the training dataset. The bottom status bar indicates 'Cell 2 of 55' and the current time is 7:49 PM on 11/29/2023.

Data Augmentation

Data Augmentation is needed when we have less data, this boosts the accuracy of our model by augmenting the data.

```
[23]: data_augmentation = tf.keras.Sequential([
      layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
      layers.experimental.preprocessing.RandomRotation(0.2),
    ])
Python
```

Applying Data Augmentation to Train Dataset

```
[24]: train_ds = train_ds.map(
      lambda x, y: (data_augmentation(x, training=True), y)
    ).prefetch(buffer_size=tf.data.AUTOTUNE)
Python
```

Model Architecture

We use a CNN coupled with a Softmax activation in the output layer. We also add the initial layers for resizing, normalization and Data Augmentation

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3
```

This screenshot shows the 'Model Architecture' section of the same Jupyter Notebook. It contains a text block describing the CNN structure and a code cell that defines the model architecture. The architecture starts with a sequential model, followed by a 'resize_and_rescale' layer, and then a series of convolutional and pooling layers. The final layers are a dense layer with 64 units and a softmax output layer. The code cell also includes a call to 'model.summary()' at the bottom. The status bar at the bottom shows 'Cell 2 of 55' and the time is 7:50 PM on 11/29/2023.

Model Architecture

We use a CNN coupled with a Softmax activation in the output layer. We also add the initial layers for resizing, normalization and Data Augmentation

```
[27]: input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
      n_classes = 3

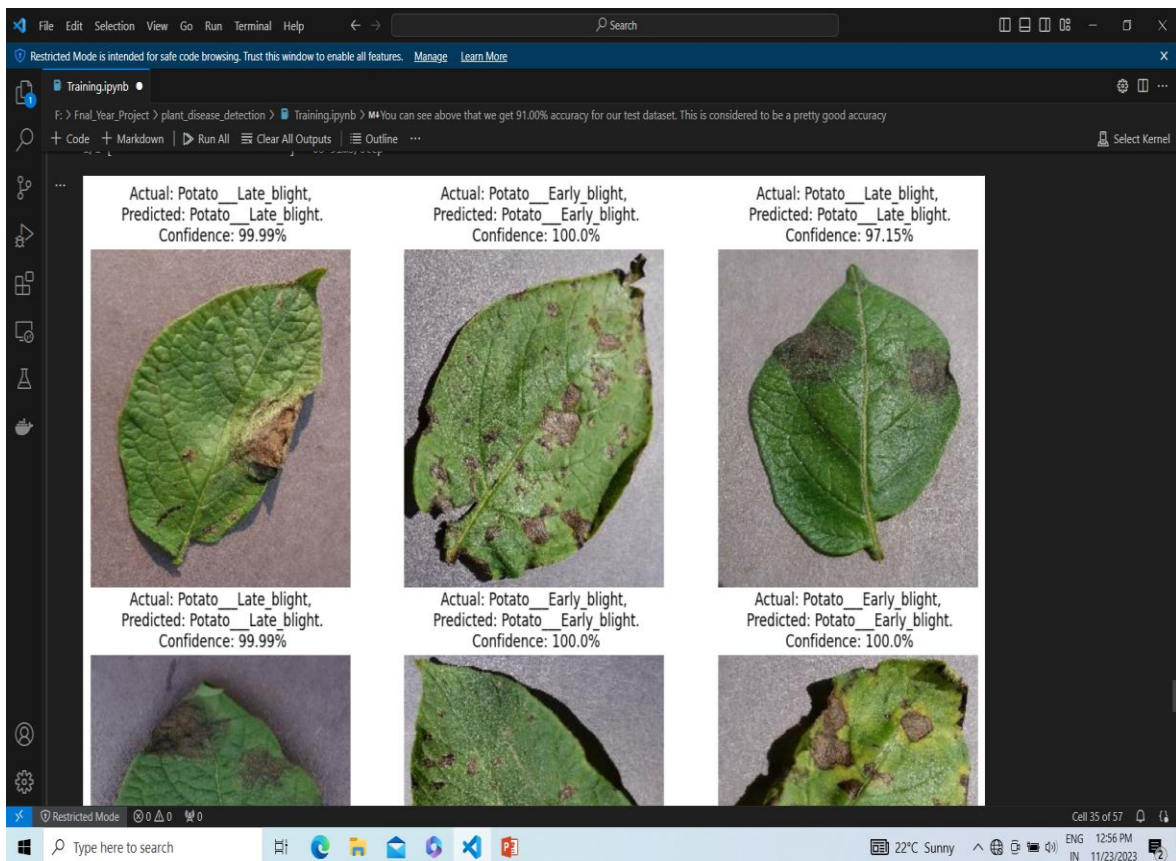
      model = models.Sequential([
        resize_and_rescale,
        layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(2, 2),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(n_classes, activation='softmax'),
      ])

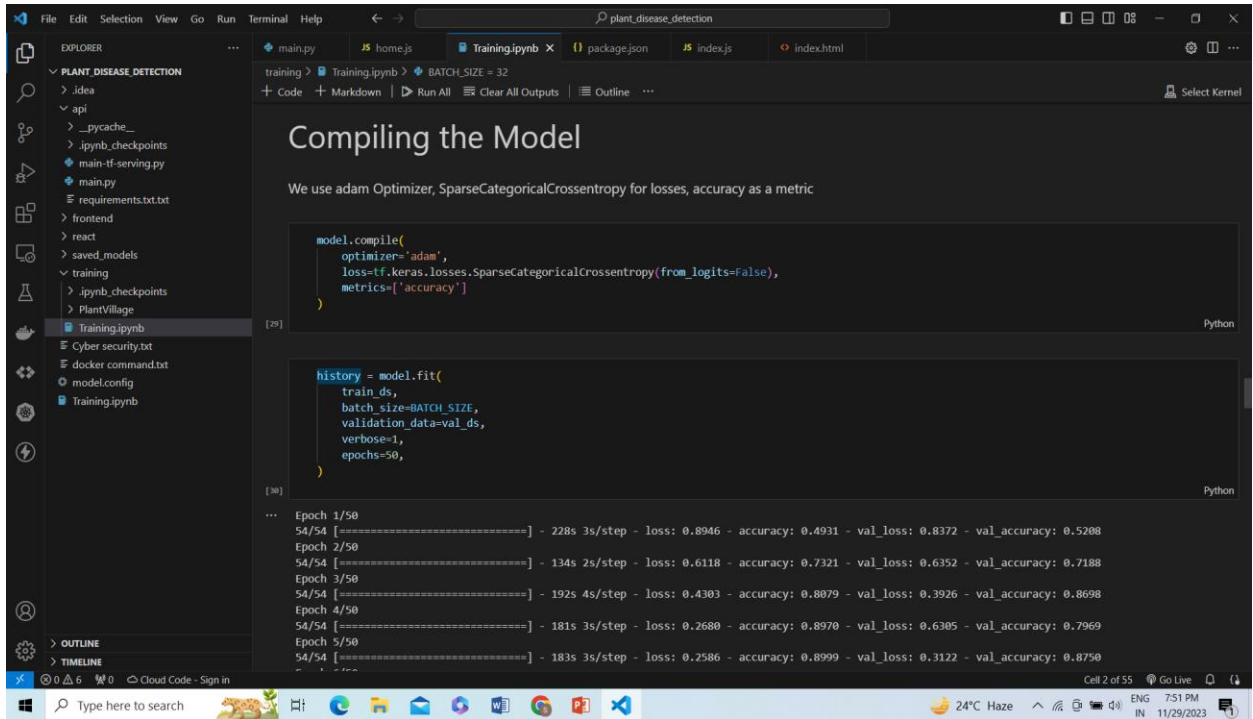
      model.build(input_shape=input_shape)
Python

      model.summary()
```

```
File Edit Selection View Go Run Terminal Help
plant_disease_detection
EXPLORER
main.py home.js Training.ipynb package.json index.js index.html
PLANT_DISEASE_DETECTION
> idea
> api
> __pycache__
> .ipynb_checkpoints
> main-tf-serving.py
> main.py
> requirements.txt.txt
> frontend
> react
> saved_models
> training
> .ipynb_checkpoints
> PlantVillage
> Training.ipynb
> Training.ipynb
OUTLINE
TIMELINE
Type here to search
24°C Haze
ENG 7:50 PM
IN 11/29/2023
```

```
model.summary()
Model: "sequential_2"
Layer (type) Output Shape Param #
-----
sequential (Sequential) (32, 256, 256, 3) 0
conv2d (Conv2D) (32, 254, 254, 32) 896
max_pooling2d (MaxPooling2D) (32, 127, 127, 32) 0
conv2d_1 (Conv2D) (32, 125, 125, 64) 18496
max_pooling2d_1 (MaxPooling2D) (32, 62, 62, 64) 0
conv2d_2 (Conv2D) (32, 60, 60, 64) 36928
max_pooling2d_2 (MaxPooling2D) (32, 30, 30, 64) 0
conv2d_3 (Conv2D) (32, 28, 28, 64) 36928
max_pooling2d_3 (MaxPooling2D) (32, 14, 14, 64) 0
Total params: 183747 (717.76 KB)
Trainable params: 183747 (717.76 KB)
Non-trainable params: 0 (0.00 Byte)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```





```
Epoch 12/50  
54/54 [=====] - 156s 3s/step - loss: 0.1693 - accuracy: 0.9421 - val_loss: 0.1815 - val_accuracy: 0.9062  
Epoch 13/50  
...  
Epoch 49/50  
54/54 [=====] - 139s 3s/step - loss: 0.0332 - accuracy: 0.9890 - val_loss: 0.1328 - val_accuracy: 0.9635  
Epoch 50/50  
54/54 [=====] - 139s 3s/step - loss: 0.0582 - accuracy: 0.9792 - val_loss: 0.2000 - val_accuracy: 0.9323  
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
scores = model.evaluate(test_ds)
```

```
8/8 [=====] - 6s 723ms/step - loss: 0.2780 - accuracy: 0.9141
```

You can see above that we get 91.00% accuracy for our test dataset. This is considered to be a pretty good accuracy

File Edit Selection View Go Run Terminal Help

plant_disease_detection

EXPLORER

- PLANT_DISEASE_DETECTION
 - idea
 - api
 - _pycache_
 - .ipynb_checkpoints
 - main-tf-serving.py
 - main.py
 - requirements.txt
 - frontend
 - react
 - saved_models
 - training
 - .ipynb_checkpoints
 - PlantVillage
- Training.ipynb
- Cyber security.txt
- docker command.txt
- model.config
- Training.ipynb

main.py JS home.js Training.ipynb X package.json JS index.js index.html

training > Training.ipynb > BATCH_SIZE = 32

+ Code + Markdown | Run All | Clear All Outputs | Outline ...

Select Kernel

```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

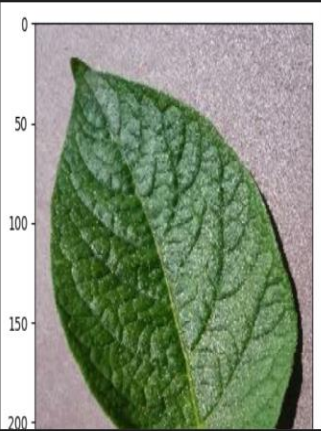
    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:", class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:", class_names[np.argmax(batch_prediction[0])])
```

[42] Python

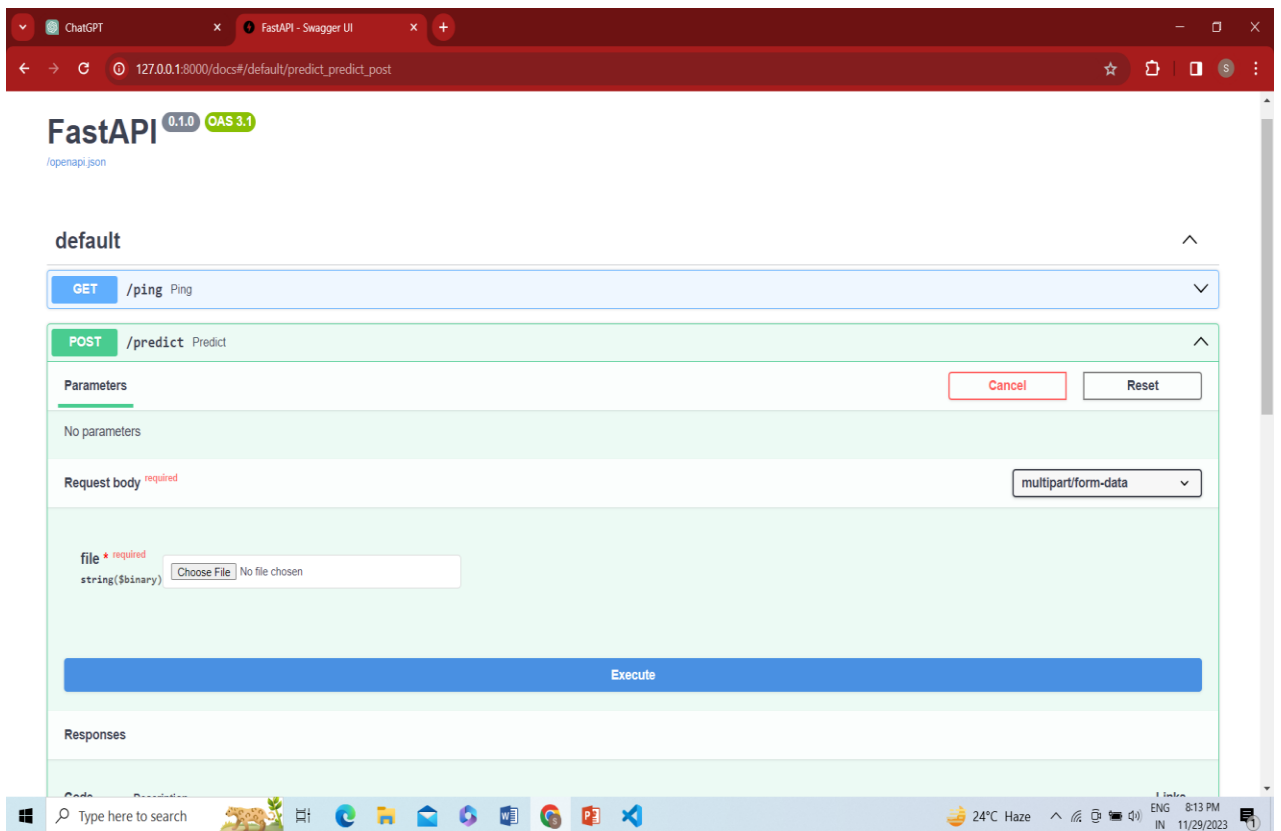
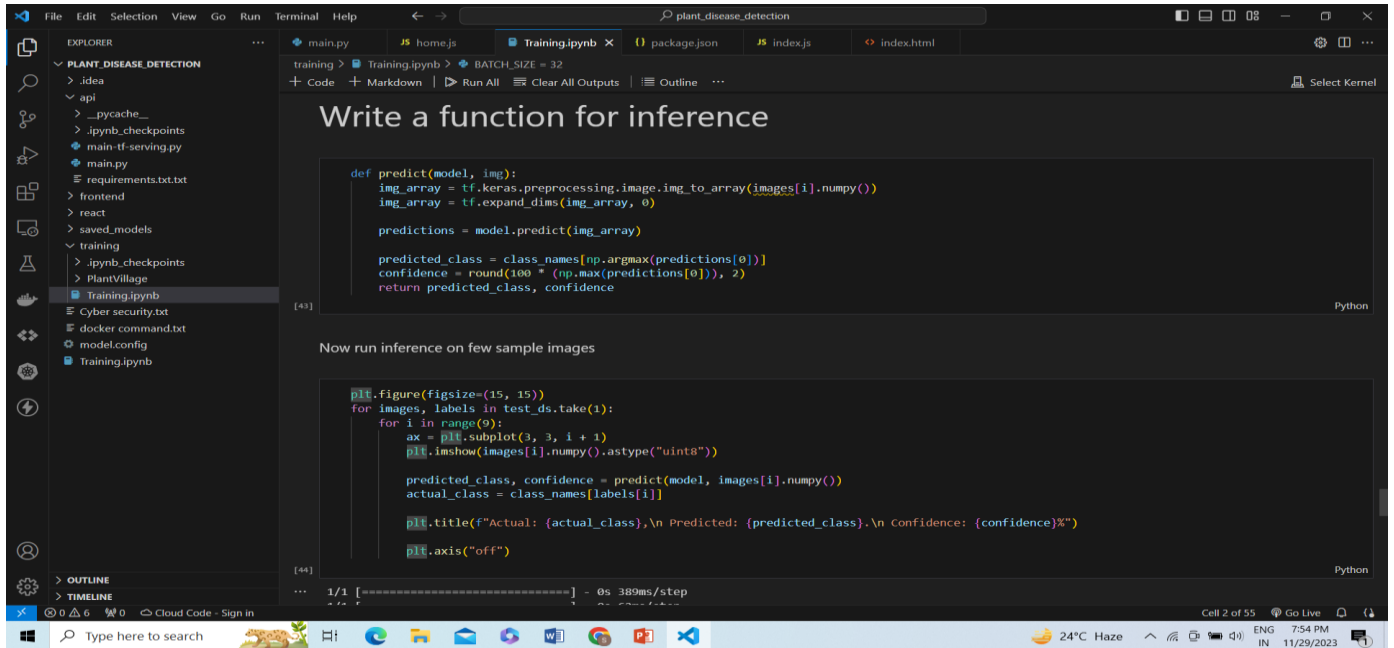
... first image to predict
actual label: Potato__healthy
1/1 [=====] - 4s 4s/step
predicted label: Potato__healthy
...

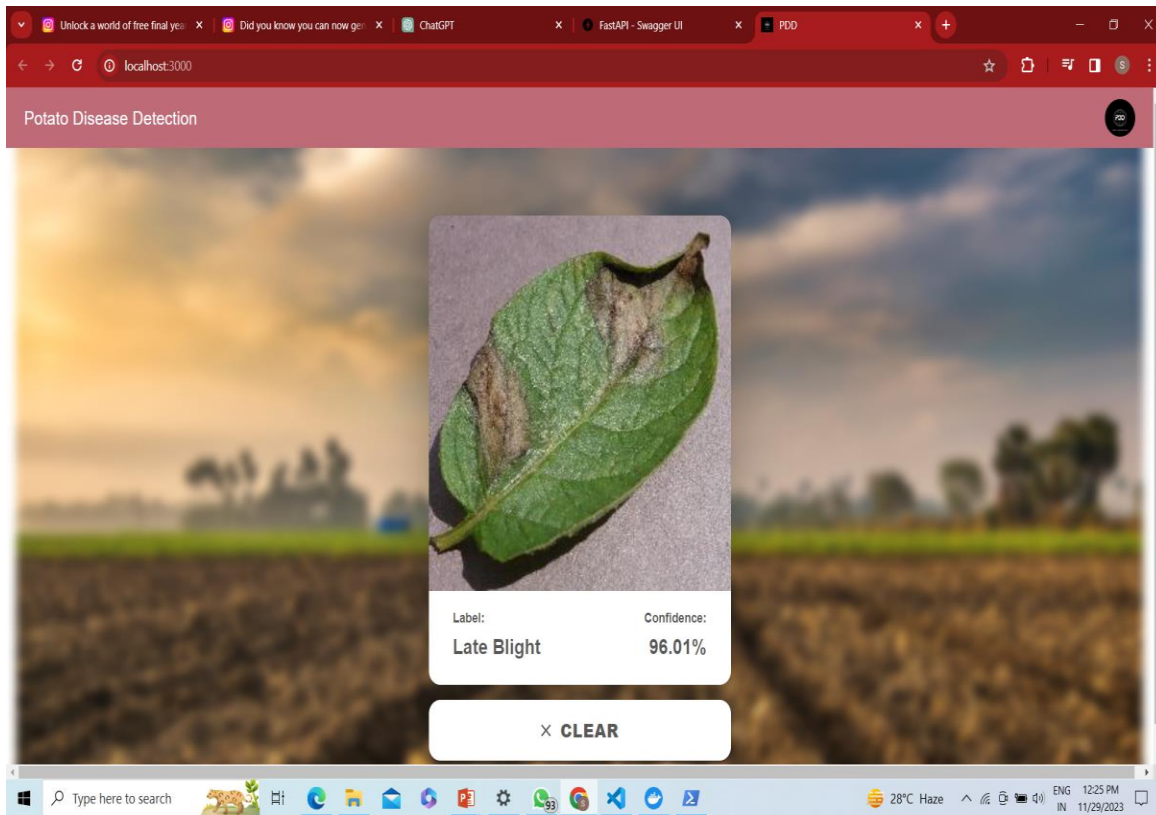
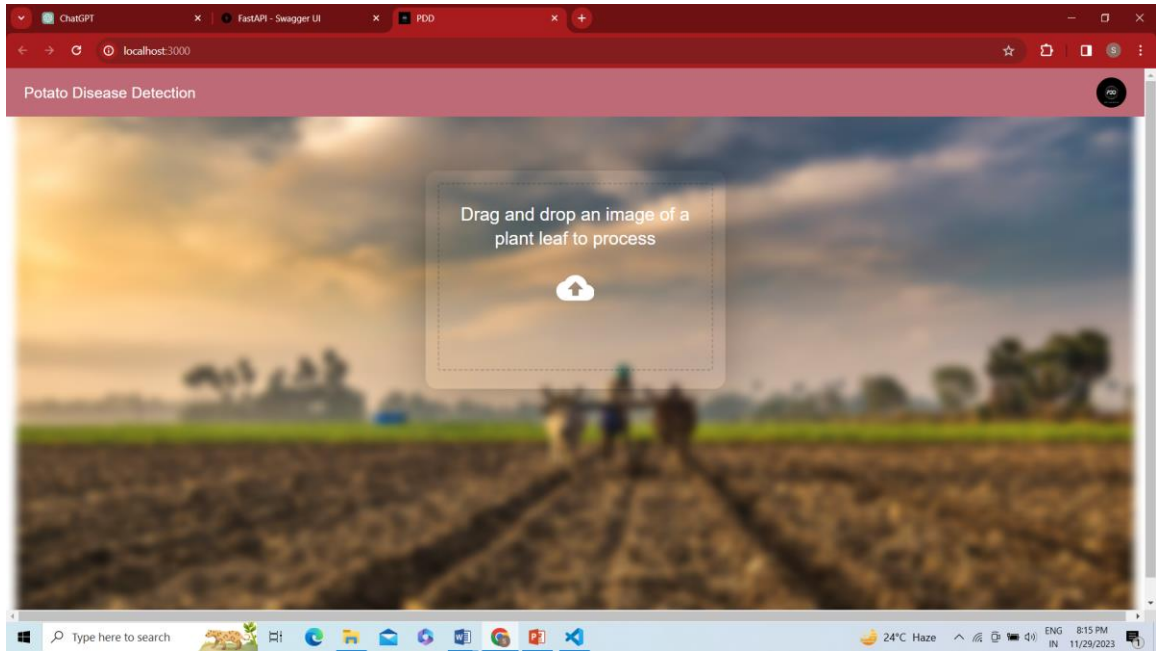


Cell 2 of 55 Go Live

Type here to search

24°C Haze 7:54 PM 11/29/2023





CONCLUSION

In developing a **Plant Disease Identification System**, this project has successfully integrated all 12 Program Objectives (POs) essential for a comprehensive engineering education. The project has demonstrated a strong foundation in engineering knowledge by applying technical expertise to address the complex issue of diseases caused in plants. Through problem analysis, the team has identified key challenges and designed innovative solutions, showcasing proficiency in investigating intricate problems. Utilizing modern tools effectively, the project underscores the importance of staying current in technology for disease prevention and cure. The project also highlights the engineer's responsibility to society by addressing online safety and promoting ethical behavior. Consideration for environmental sustainability is evident in the system's design and implementation. Emphasizing teamwork and individual contributions, the project fosters collaboration and effective communication skills. Project management and financial acumen have been crucial in ensuring the success of the Pant Disease Identification System. Lastly, the project encourages a commitment to lifelong learning, reflecting a dedication to continuous improvement and innovation in combating such diseases.

By doing this Project we achieve

[illegible]

REFERENCES

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Comput. Vis. Image Underst.* 110, 346–359. doi: 10.1016/j.cviu.2007.09.014

Chéné, Y., Rousseau, D., Lucidarme, P., Bertheloot, J., Caffier, V., Morel, P., et al. (2012). On the use of depth camera for 3d technology has developed in the past few years, and will continue to do so. With ever improving number and quality of sensors on mobiles devices, we consider it likely that highly accurate diagnoses via the smartphone are only a question of time.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at:

<http://journal.frontiersin.org/article/10.3389/fpls.2016.01419>

https://github.com/salathgroup/plantvillage_deep_learning_paper_dataset

https://github.com/salathgroup/plantvillage_deep_learning_paper_analysis

More image data can be found at

https://www.plantvillage.org/en/plant_images