GOOGLE AI-ML VIRTUAL INTERNSHIP

An Internship-1 report submitted in partialfulfillment of requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

By

B.Vasupriya Patnaik	**(322103382001)**
B.Sherlie Angel	**(322103382002)**
K.Gowthami	**(322103382030)**
S.Kirthi Naga Sarvani	**(322103382053)**

Under the esteemed guidance of

Internship Mentor	**Name of Course Coordinator**
Mr. M S N MURTY	**Mr.CH SRIKANTH VARMA**
(Assistant Professor)	**(Assistant Professor)**

Department of Computer Science and Engineering

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)

(Affiliated to Andhra University)

VISAKHAPATNAM

2024 – 2025

Gayatri Vidya Parishad College of Engineering (Autonomous)
Visakhapatnam

CERTIFICATE

This report on

"GOOGLE AI-ML VIRTUAL INTERNSHIP"

is a bonafide record of the Internship work submitted

By

B.Vasupriya Patnaik	**(322103382001)**
B.Sherlie Angel	**(322103382002)**
K.Gowthami	**(322103382030)**
S.Kirthi Naga Sarvani	**(322103382053)**

In their V semester in partial fulfilment of the requirements for the Award of Degree of

Bachelor of Technology in
Computer Science and Engineering
(AI&ML)

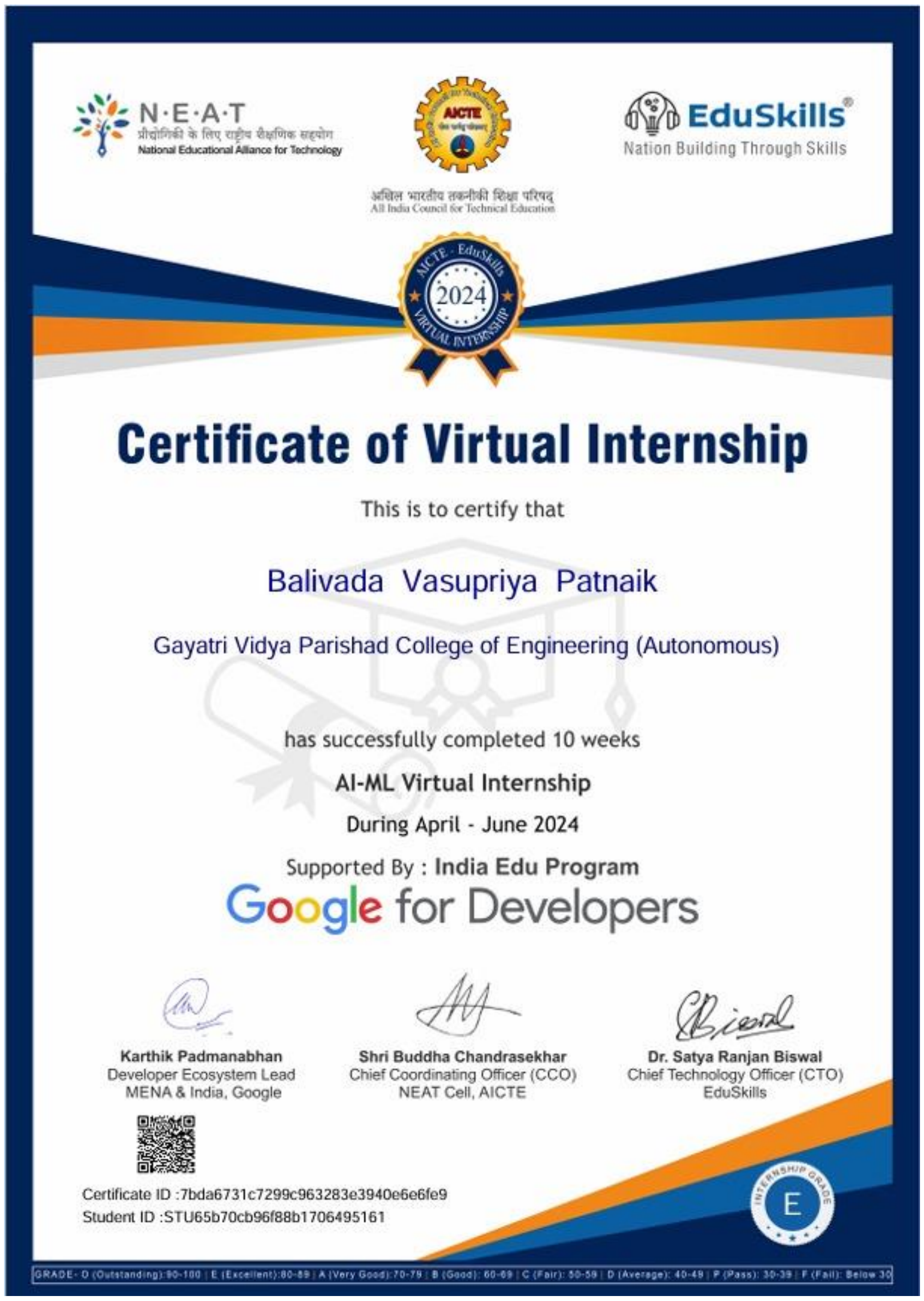
During the academic year 2024-2025

Name of Course Coordinator
Mr.CH.SRIKANTH VARMA
(Assistant Professor)

Head of the Department
Dr. D. UMA DEVI
(Associate Professor)

Internship Mentor
Mr. M S N MURTY
(Assistant Professor)

B.VASUPRIYA PATNAIK (322103382001)



N·E·A·T
प्रौद्योगिकी के लिए राष्ट्रीय शैक्षणिक सहयोग
National Educational Alliance for Technology

अखिल भारतीय तकनीकी शिक्षा परिषद्
All India Council for Technical Education

EduSkills®
Nation Building Through Skills

AICTE - EduSkills 2024 VIRTUAL INTERNSHIP

Certificate of Virtual Internship

This is to certify that

Balivada Vasupriya Patnaik

Gayatri Vidya Parishad College of Engineering (Autonomous)

has successfully completed 10 weeks

AI-ML Virtual Internship

During April - June 2024

Supported By : **India Edu Program**

Google for Developers

[illegible] [illegible] [illegible]

Karthik Padmanabhan
Developer Ecosystem Lead
MENA & India, Google

Shri Buddha Chandrasekhar
Chief Coordinating Officer (CCO)
NEAT Cell, AICTE

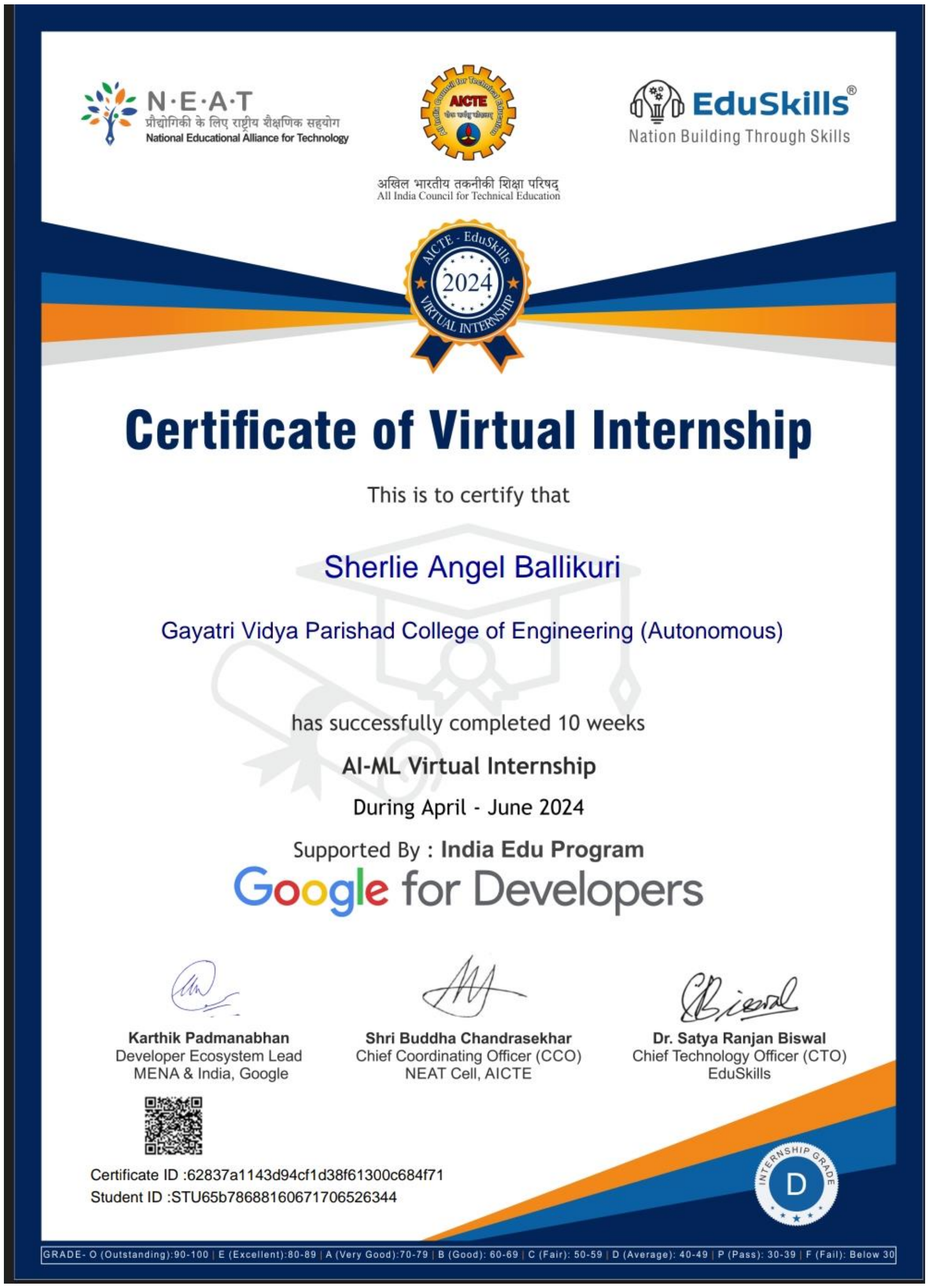
Dr. Satya Ranjan Biswal
Chief Technology Officer (CTO)
EduSkills

Certificate ID :7bda6731c7299c963283e3940e6e6fe9
Student ID :STU65b70cb96f88b1706495161

INTERNSHIP GRADE E

GRADE- O (Outstanding):90-100 | E (Excellent):80-89 | A (Very Good):70-79 | B (Good): 60-69 | C (Fair): 50-59 | D (Average): 40-49 | P (Pass): 30-39 | F (Fail): Below 30

B.SHERLIE ANGEL (322103382002)



N·E·A·T
प्रौद्योगिकी के लिए राष्ट्रीय शैक्षणिक सहयोग
National Educational Alliance for Technology

अखिल भारतीय तकनीकी शिक्षा परिषद्
All India Council for Technical Education

EduSkills®
Nation Building Through Skills

AICTE - EduSkills 2024 VIRTUAL INTERNSHIP

Certificate of Virtual Internship

This is to certify that

Sherlie Angel Ballikuri

Gayatri Vidya Parishad College of Engineering (Autonomous)

has successfully completed 10 weeks

AI-ML Virtual Internship

During April - June 2024

Supported By : India Edu Program

Google for Developers

Karthik Padmanabhan
Developer Ecosystem Lead
MENA & India, Google

Shri Buddha Chandrasekhar
Chief Coordinating Officer (CCO)
NEAT Cell, AICTE

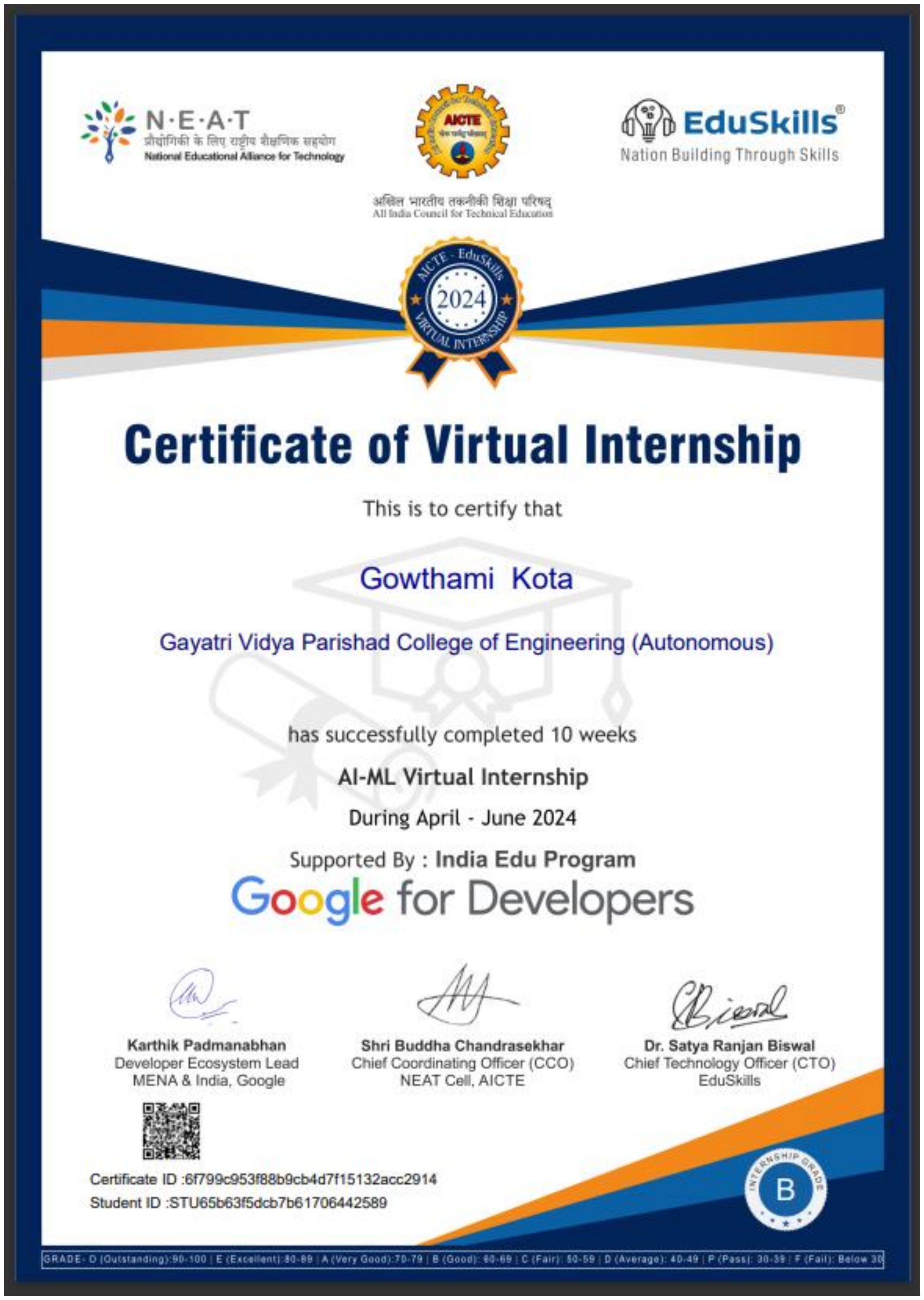
Dr. Satya Ranjan Biswal
Chief Technology Officer (CTO)
EduSkills

Certificate ID :62837a1143d94cf1d38f61300c684f71
Student ID :STU65b78688160671706526344

INTERNSHIP GRADE D

GRADE- O (Outstanding):90-100 | E (Excellent):80-89 | A (Very Good):70-79 | B (Good): 60-69 | C (Fair): 50-59 | D (Average): 40-49 | P (Pass): 30-39 | F (Fail): Below 30

K.GOWTHAMI (322103382030)



N·E·A·T
प्रौद्योगिकी के लिए राष्ट्रीय शैक्षणिक सहयोग
National Educational Alliance for Technology

AICTE

अखिल भारतीय तकनीकी शिक्षा परिषद्
All India Council for Technical Education

EduSkills®
Nation Building Through Skills

AICTE - EduSkills 2024 VIRTUAL INTERNSHIP

Certificate of Virtual Internship

This is to certify that

Gowthami Kota

Gayatri Vidya Parishad College of Engineering (Autonomous)

has successfully completed 10 weeks

AI-ML Virtual Internship

During April - June 2024

Supported By : **India Edu Program**

Google for Developers

[illegible] [illegible] [illegible]

Karthik Padmanabhan
Developer Ecosystem Lead
MENA & India, Google

Shri Buddha Chandrasekhar
Chief Coordinating Officer (CCO)
NEAT Cell, AICTE

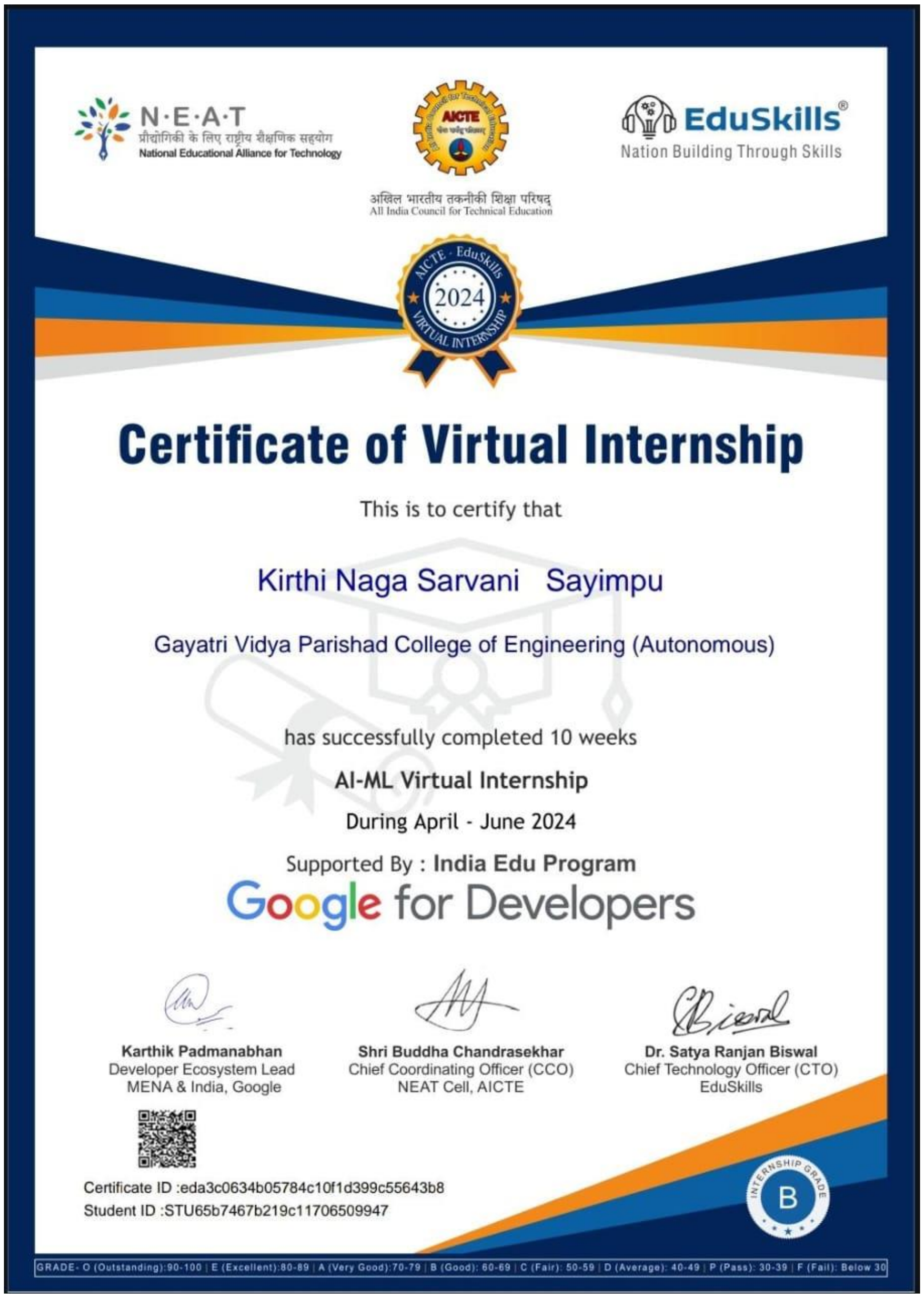
Dr. Satya Ranjan Biswal
Chief Technology Officer (CTO)
EduSkills

Certificate ID :6f799c953f88b9cb4d7f15132acc2914
Student ID :STU65b63f5dcb7b61706442589

INTERNSHIP GRADE B

GRADE- O (Outstanding):90-100 | E (Excellent):80-89 | A (Very Good):70-79 | B (Good): 60-69 | C (Fair): 50-59 | D (Average): 40-49 | P (Pass): 30-39 | F (Fail): Below 30

S.KIRTHI NAGA SARVANI (322103382053)



N·E·A·T
प्रौद्योगिकी के लिए राष्ट्रीय शैक्षणिक सहयोग
National Educational Alliance for Technology

अखिल भारतीय तकनीकी शिक्षा परिषद्
All India Council for Technical Education

EduSkills
Nation Building Through Skills

AICTE - EduSkills 2024 VIRTUAL INTERNSHIP

Certificate of Virtual Internship

This is to certify that

Kirthi Naga Sarvani Sayimpu

Gayatri Vidya Parishad College of Engineering (Autonomous)

has successfully completed 10 weeks

AI-ML Virtual Internship

During April - June 2024

Supported By : **India Edu Program**

Google for Developers

Karthik Padmanabhan
Developer Ecosystem Lead
MENA & India, Google

Shri Buddha Chandrasekhar
Chief Coordinating Officer (CCO)
NEAT Cell, AICTE

Dr. Satya Ranjan Biswal
Chief Technology Officer (CTO)
EduSkills

Certificate ID :eda3c0634b05784c10f1d399c55643b8
Student ID :STU65b7467b219c11706509947

INTERNSHIP GRADE B

GRADE- O (Outstanding):90-100 | E (Excellent):80-89 | A (Very Good):70-79 | B (Good): 60-69 | C (Fair): 50-59 | D (Average): 40-49 | P (Pass): 30-39 | F (Fail): Below 30

ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude to our esteemed institute **Gayatri Vidya Parishad College of Engineering (Autonomous)**,which has provided us an opportunity to fulfill our cherished desire.

We express our sincere thanks to our Principal **Dr. A.B.KOTESWARA RAO, Gayatri Vidya Parishad College of Engineering (Autonomous)** for his encouragement to us during this project, giving us a chance to explore and learn new technologies in the form of mini projects.

We thank our **internship mentor Mr. M S N Murty,** Assistant Professor, Department of Computer Science and Engineering and our **Course Coordinators Mr.B Srinu** and **Mr. CH Srikanth Varma,** Assistant Professors for the kind suggestions and guidance for the successful completion of our internship.

We are highly indebted to **Dr. D. Uma Devi, Associate Professor and Head of the Department of Computer Science and Engineering, Gayatri Vidya Parishad College of Engineering (Autonomous),** for giving us an opportunity to do the internship in college.

We extend our gratitude to our **internship coordinators**, **Dr. Ch. Sita Kumari** and **Mrs. T. Tejeswari**, for presenting us with a variety of internship opportunities.

We are grateful for **EDUSKILLS, GOOGLE &AICTE** for providing us this learning opportunity.

Finally, we are indebted to the teaching and non-teaching staff of the Computer Science and Engineering Department for all their support in completion of our project.

B.Vasupriya Patnaik	**(322103382001)**
B.Sherlie Angel	**(322103382002)**
K.Gowthami	**(322103382030)**
S.Kirthi Naga Sarvani	**(322103382053)**

ABSTRACT

This project documentation offers a comprehensive guide to leveraging TensorFlow, a powerful machine learning framework, for developing and deploying neural networks in various computer vision tasks, including object detection, product image search, and image classification. It begins with an introduction to TensorFlow and its importance in programming neural networks, followed by a step-by-step guide on building object detection models and enhancing their performance through techniques like fine-tuning pre-trained models and optimizing inference speed. The document then explores the implementation of image-based product search functionalities using TensorFlow, covering foundational concepts and advanced techniques, such as integration with Vision API Product Search and personalization features

Additionally, it addresses image classification, detailing the development of custom image classifiers and methods to improve model accuracy and efficiency. Throughout the document, practical examples, code snippets, and implementation guidelines are provided to facilitate understanding and application, making it a valuable resource for both beginners and experienced practitioners in programming neural networks with TensorFlow for a variety of computer vision tasks.

TABLE OF CONTENTS

MNC THAT PROVIDED RESOURCES (GOOGLE)

Google, a global technology leader, is renowned for its innovative products and services that have transformed the way people access and interact with information. Founded in 1998 by Larry Page and Sergey Brin, Google began as a search engine that revolutionized internet search capabilities with its powerful algorithms. Today, Google offers a wide range of services including Gmail, Google Maps, Google Drive, and the Android operating system, serving billions of users worldwide. The company is also a pioneer in artificial intelligence and machine learning, developing advanced technologies such as Google Assistant and TensorFlow. Google's mission is to organize the world's information and make it universally accessible and useful, driving continuous advancements in technology and setting high standards for innovation and user experience.

Google's Virtual AI & ML Internship Program is a premier opportunity for aspiring technologists to dive deep into the world of artificial intelligence and machine learning. Hosted by Google, a global technology leader renowned for its innovative products and services, the program is designed to offer hands-on experience with cutting-edge AI and ML technologies, tools, and methodologies. Interns work alongside Google's expert engineers and researchers, contributing to high-impact projects that push the boundaries of innovation. The program emphasizes learning through real-world applications, from developing advanced algorithms to creating scalable AI solutions. Participants gain invaluable insights into the intricacies of AI development, data analysis, and the implementation of machine learning models, preparing them for future careers in this dynamic and rapidly evolving field. As part of Google, which is committed to organizing the world's information and making it universally accessible and useful, interns have the unique opportunity to be part of an organization that continually drives technological advancement and sets high standards for innovation and user experience.

MODULE 1

1. Program neural networks with TensorFlow

1.1 Introduction

Programming neural networks with TensorFlow allows developers to harness the power of deep learning for a wide range of tasks. TensorFlow, an open-source machine learningframework developed by Google, provides a comprehensive suite of tools for building,training, and deploying neural networks. From creating basic models to tackling complex problems like computer vision, TensorFlow offers an intuitive interface and high-level abstractions, enabling developers to focus on the architecture and experimentation of neural networks rather than low-level implementation details. With TensorFlow, individuals can explore the cutting-edge advancements in artificial intelligence and apply them to real-world scenarios efficiently and effectively

1.2 Advantages

- Ease of Use
- Flexibility
- Scalability
- Community and Ecosystem
- Production Readiness

Fig 1.1 TensorFlow Badge

1.3 Say hello to the "Hello, World" of machine learning

“Hello, World" of ML, where instead of programming explicit rules in a language, such as Java or C++,you’ll build a system trained on data to infer the rules that determine a relationship between numbers.

Consider the following problem: You're building a system that performs activity recognition for fitness tracking. You might have access to the speed at which a person is walking and attempt to infer their activity based on that speed using a conditional.

1.4 What is ML?

Consider the traditional manner of building apps, as represented in the following diagram:

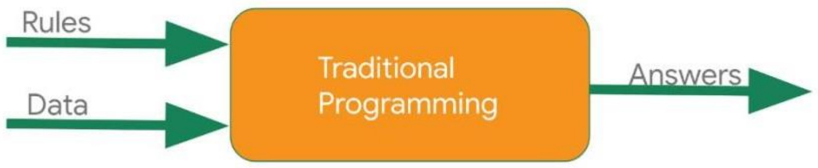


Fig 1.2 Traditional manner of building apps

You express rules in a programming language. They act on data and your program provides answers. In the case of the activity detection, the rules (the code you wrote to define activity types) acted upon the data (the person's movement speed) to produce an answer: the return value from the function for determining the activity status of the user (whether they were walking, running, biking, or doing something else).

The process for detecting that activity status via ML is very similar, only the axes are different.

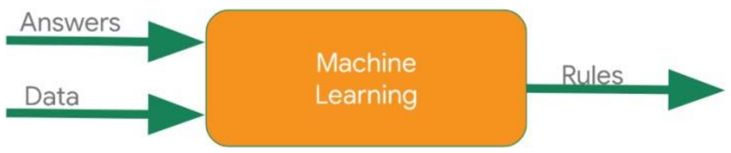


Fig 1.3 ML manner of building apps

Instead of trying to define the rules and express them in a programming language, you provide the answers (typically called labels) along with the data, and the machine infers the rules that determine the relationship between the answers and data. For example, your activity detection scenario might look like this in an ML context:

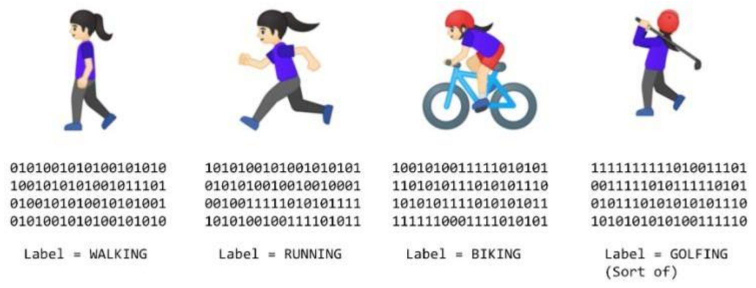


Fig 1.3 Picture viewed as binary

You gather lots of data and label it to effectively say, "This is what walking looks like," or "This is what running looks like." Then, the computer can infer the rules that determine, from the data, what the distinct patterns that denote a particular activity are.

Beyond being an alternative method to programming that scenario, that approach also gives you the ability to open new scenarios, such as the golfing one that may not have been possible under the rules-based traditional programming approach.

In traditional programming, your code compiles into a binary that is typically called a program. In ML, the item that you create from the data and labels is called a model.

So, if you go back to this diagram:

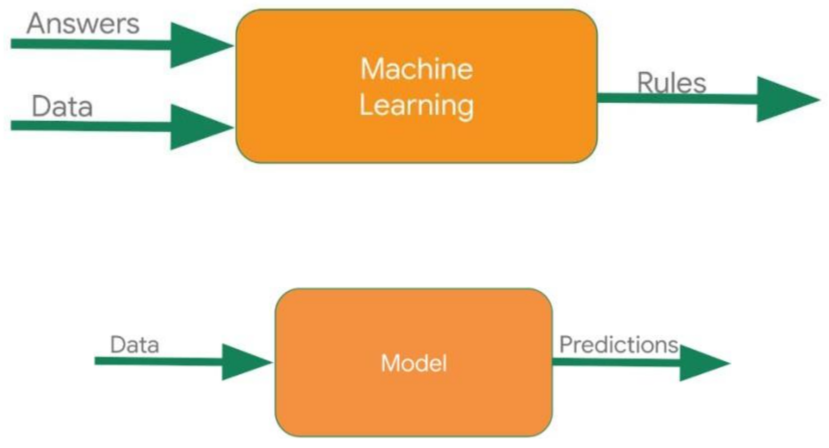


Fig 1.4 Model uses the rules

MODULE 2

2. Get started with object detection

2.1 Introduction

Object detection is a computer vision task that involves identifying and locating objects within images or videos. Unlike image classification, which assigns a label to an entire image, object detection goes a step further by precisely locating and delineating the boundaries of individual objects within the image. This enables machines to not only recognize what objects are present but also understand where they are located in the scene. Object detection finds applications in various fields, including autonomous driving, surveillance, medical imaging, and retail. It forms the basis for many advanced technologies such as facial recognition, pedestrian detection, and industrial quality control. In recent years, deep learning techniques, particularly convolutional neural networks (CNNs), have revolutionized object detection, achieving remarkable accuracy and robustness in detecting objects across diverse environments and conditions. TensorFlow provides powerful tools and libraries for building and deploying object detection models, makingit accessible for researchers and developers to tackle real-world challenges effectively.

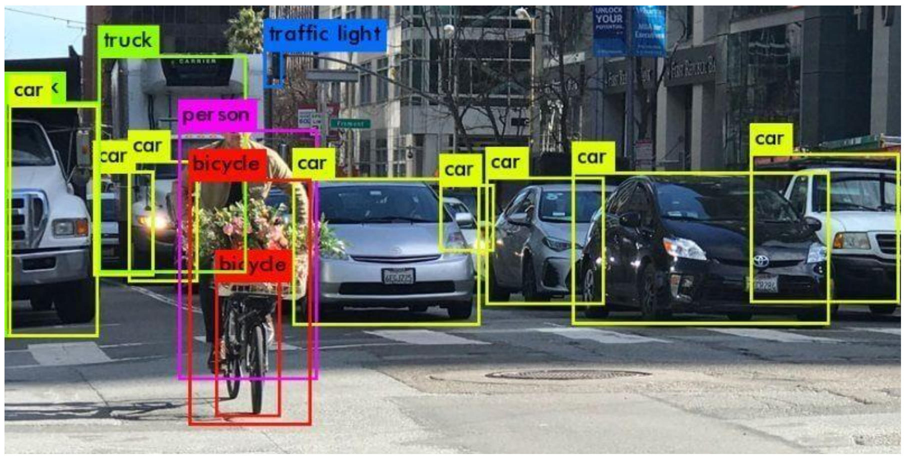


Fig 2.1 Object Detection

2.2 Build an object detector into your mobile app

- Object detection is a computer vision task that detects what objects are in an image and where they are located.
- This API can take an image as an input and detect the prominent objects in the image. It tells

you where the objects are and what type of objects they are.

- We start with an app that already has some UI boilerplate code.
- You can select one of the preset images and it will show up in the larger view. Or you also cantake a photo with your phone camera.
- Assign the ML Kit library to the app dependency.
- Go to the build.gradle file and add it.

2.3 Integrate an object detector using ML Kit Object Detection API

ML Kit is a mobile SDK that brings Google's on-device machine learning expertise to Android and iOS apps. You can use the powerful yet simple to use Vision and Natural Language APIs to solve common challenges in your apps or create brand-new user experiences. All are powered by Google's best-in-class ML models and offered to you at no cost.

ML Kit's APIs all run on-device, allowing for real-time use cases where you want to process a live camera stream, for example. This also means that the functionality is available offline.

This codelab will walk you through simple steps to add Object Detection and Tracking (ODT) fora given image into your existing Android app. Please note that this codelab takes some shortcuts to highlight ML Kit ODT usage.

2.4 What we will build

In this codelab, we are going to build an Android app with ML Kit. Our app will use the ML Kit Object Detection and Tracking API to detect objects in a given image. In the end, we will see something similar to the image on the right.

MODULE 3

3 Go further with object detection

3.1 Train your own object-detection model

To train your own object detection model, start by collecting a dataset of images containing the objects you want to detect and annotating them with bounding boxes. Split the dataset into training, validation, and testing sets. Choose a deep learning framework like TensorFlow, PyTorch, or Keras. Select a pre-trained model such as Faster R-CNN, SSD, YOLO, or RetinaNet, and fine-tune it on your dataset. Monitor the training process, adjusting hyperparameters like learning rate and batch size as needed. Evaluate your model's performance using metrics like precision, recall, and mAP. Once satisfied, deploy your model for inference, integrating it into an application or deploying it to a server or edge device. For TensorFlow, install the Object Detection API, clone the TensorFlow Models repository, install dependencies, compile Protobufs, and configure the model pipeline. Prepare your dataset by organizing images and annotation files, and modify the configuration file of the chosen pre-trained model to suit your task.

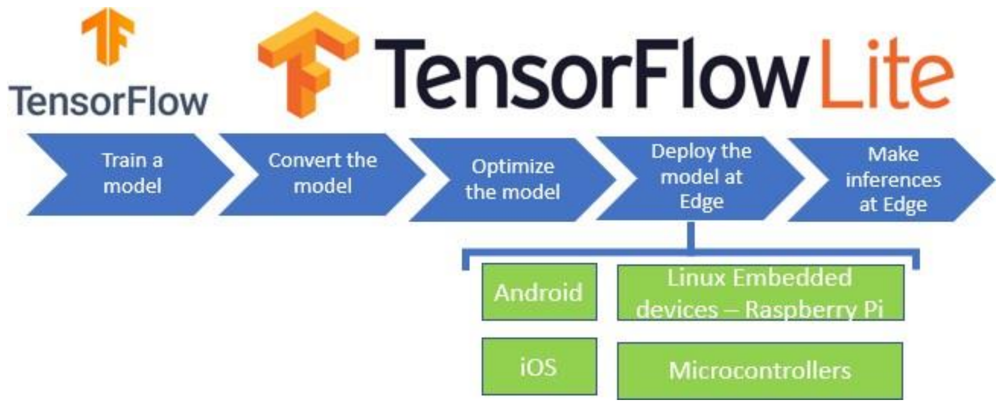


Fig 3.1 TensorFlow Lite process flow

3.2 Build and deploy a custom object-detection model with TensorFlow Lite

To build and deploy a custom object detection model with TensorFlow Lite, begin by preparing your dataset, which involves collecting images of the objects you want to detect and annotating them with bounding boxes. Next, choose a suitable model architecture like MobileNet, EfficientDet, or YOLO, either starting with pre-trained models or training from scratch depending on your requirements. Afterward, preprocess your data by resizing images to the input size expected by the model, normalizing pixel values, and converting annotations to TensorFlow's TFRecord format. Then, train your model using TensorFlow, fine-tuning it on your dataset and monitoring training metrics.

Once trained, convert the TensorFlow model to TensorFlow Lite format using the TensorFlow Lite Converter, optimizing it for deployment on resource-constrained devices and applying optimizations like quantization to reduce model size and inference latency. Finally, integrate the TensorFlow Lite model into your application using TensorFlow Lite Interpreter and deploy it on mobile devices, edge devices, or embedded systems, allowing for efficient object detection inference in real-world scenarios.

3.3 Convert to TensorFlow Lite

```
tflite_convert --output_file=your_model.tflite --saved_model_dir=/path/to/saved_model
```

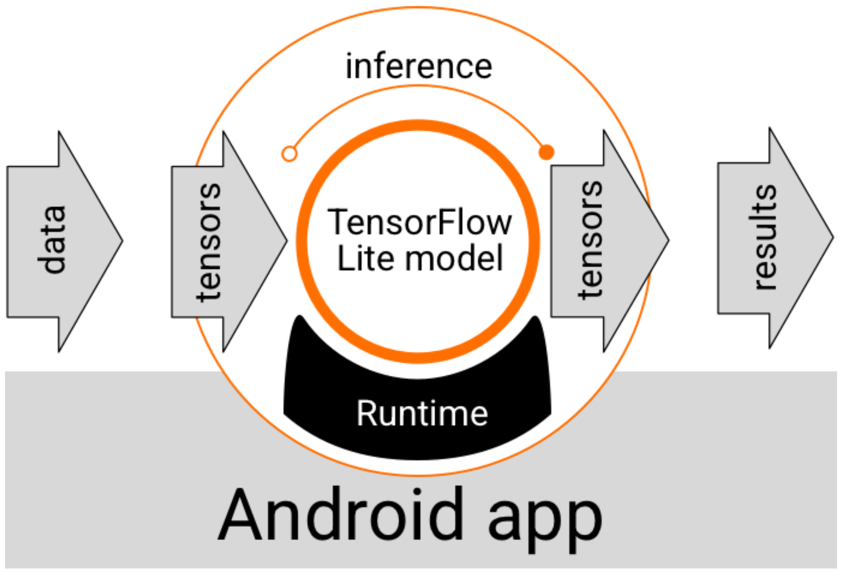


Fig 3.2 Android app

MODULE 4

4. Get started with product image search

4.1 Introduction

- Imagine when you see a cool item on the street and you wonder where you can buy it online. You pull out your smartphone, point the camera at the product and boiler. You find a list of online shops where you can buy similar products. Google Lens already has this feature for a while, but it can also build a similar feature to your mobile app, to let your users search for visually similar products in your own product catalog.
- There are two components to build this feature.
- The first one is an on-device object detector to identify the prominent objects in the image and suggest to the users which object they want to search for. Later in this learning pathway, I'll show you how to build this component using the ML Kit Object Detection and Tracking API. Then we'll learn some material design best practices in building an on-device object detector.
- The second component is a product search backend. Because your product catalog can be big, we should keep them on the image and suggest to the users which object they want to search for.

4.2 Build an object detector into your mobile app

When an user captures an image using their phone camera, we want them to tell us which object in the image that they want to search for in our product catalog. We do so by first detecting objects in the image, then second, show the detection result on the screen, and finally,let the user choose the object that they want to search for. We will implement the Object Detector by using ML Kit's Object Detection and Tracking API.

ML Kit provides many easy-to-use APIs to integrate machine learning into your mobile app without any machine l learning expertise required. To keep everything simple and focused on the machine learning parts of the app, we'll start with a skeleton app that already has a lot of boilerplate UI code. The app can take photos using a phone camera, or it can also load images from the phone's photo output. Next, we'll add code to do object detection.

We start with adding ML Kit to the app dependency. Go to the build.gradle file and add this line. Then next, go to the main activity and find this method. The starter app is implemented so that whenever you pick an image, either by taking photos using the camera app or selecting onefrom the present images, this method will be called. Let's add Object Addiction Code to this method. There are only three steps to use the ML Kit Object Addiction API. First, you need to create an input image from the image that the user has selected. Next, you need to create a Object Addiction Client. Here we specify a few options. We detect objects from a still image,so we use single image mode. We expect that there will be multiple objects in the image, so we enable the Multiple Objects option. And finally, we also enable the Classification optionbecause we want to know what kind of object has the model detected and filter out only the objects that we are interested in. In this case, we want to detect only fashion items.

Then we feed the input image to the detector and print the detection result to lockout. Let's take a closer look at the code that prints the detection result. You can see that the Object Detection API returns a list of detected objects, each representing an object that it found in the input image. In each detected object, there's a bounding box or rectangle showing where the object is located in the input image. And there's also multiple labels indicating which type of object it is. There's a confidence score indicating how confident the model is when it says the object is of that type. Okay, now let's run the app. You can see that in this image, the model candetect one fashion good, and where it is in the image.

It's 94% confident about the detection. Okay, so now you know how to detect objects with MLGate. Very easy, isn't it? We'll draw the detection results on the app UI so that the user can select the object that they want to search for. There's already a method implemented in the starter app that does this for you. You just need to pass the body boxes of the objects to the method. Let's run the app again. You can see that a white dot shows up where the object is detected. Now users can select the object that they want to search for by tapping on the dot. And that's it. You can go to the code lab following this video in the learning pathway to download the source code and go through the steps that I have just shown you.

4.3 Detect objects in images to build a visual product search

To implement a visual product search feature on Android, start by selecting a suitable pre- trained object detection model like SSD MobileNet or Faster R-CNN. Convert this model to TensorFlow Lite format using TensorFlow Lite Converter to ensure compatibility and efficiency on mobile devices. Set up your Android project in Android Studio or any preferred IDE. Integrate the TensorFlow Lite model into your app by placing the `.tflite` file in the assets folder and adding TensorFlow Lite dependency in your app's `build.gradle` file. Next, develop the logic to capture images using the device's camera or select them from the gallery. Utilize TensorFlow Lite Interpreter to perform object detection on these images, extracting informationabout detected objects, their classes, and bounding box coordinates. Implement UI elements to display the detection results, providing users with an intuitive interface for visual product searchdirectly on their Android devices. Finally, refine and optimize the feature for performance and user experience, testing thoroughly on different devices to ensure reliable functionality.

MODULE 5

5 Go further with product image search

5.1 Call the product search backend from the mobile app

We can build a product search backend with your own product catalog by using the Google Cloud Vision API product search. No machine learning expertise is required. First, each backend lives in a Google Cloud project, so we need to know the project ID to call the backend.

The product search backend needs to be deployed on a Google Cloud location somewhere in the world. You'll need to choose a cloud location when you create the backend. And don't forget to take note of the location ID because it will use it later in the app. A product set is a simple container for a group of products. A product catalog can be represented as a product set and its products. A product. After you have created a product set, you can create multiple products and add them to the product set. Reference images. There are images containing various views of your product. For example, you can add two reference images for address, one taken from the front and the other taken from the back. Reference images are used to search for visually similar products.

Once you have created your product set and it has been indexed, you can query the product set using the Cloud Vision API. Also, it's very easy to create a product search backend by using the Vision APIproduct search. You still need to prepare a few things in advance, such as creating a cool cloud account with building enabled. So to help you quickly go through the learning pathway, we have deployed a demo product search backend for you. It was built by using the Vision API product search under the hood, and a product catalog of about 100 dresses and shoes. You need an API key to call the Vision API product search from a mobile app. So to keep it simple for you when using the demo backend, we have created a proxy endpoint.

So what does the proxy endpoint do? When you send a product search request to the proxy, it will addthe necessary authentication factors, such as the API key, and forward the same request to the underlying Vision API product search. Then after receiving the response from the Vision API, it will just pass the response over to the app. In the starter app, it will see the code that configures the demo backend in the product search API plan class. If you build your own product search backend with Vision API product search, you will need to change the demo backend's API endpoint to the Vision API endpoint, putting in your API key, project ID, location ID, and product set ID. but the rest of the code stays the same.

To call the product search backend from the Android app, follow these steps:

- Set Up Backend API: Develop an API on your backend server that can handle product search requests. This API should have endpoints to receive images or image URLs as input for product search. Implement the logic to perform product search using Vision API Product Search or any other suitable technology on the server side. Ensure that the API returns search results to the client in a structured format, such as JSON.

- HTTP Request from Android App: In your Android app, integrate an HTTP client library likeRetrofit, Volley, or OkHttp to make HTTP requests to the backend API. Configure the HTTPclient to send requests to the appropriate endpoint of your backend API.

- Prepare Image Data: Before making the request, prepare the image data to be sent to the backendAPI. You may need to encode the image in a suitable format, such as Base64 encoding, depending on the API's requirements. Ensure that the image data is included in the request payload.

- Send HTTP Request: Use the HTTP client library to send a POST request to the backend APIendpoint dedicated to product search. Include the prepared image data in the request body.

- Handle Response: Implement logic in your app to handle the response from the backend API. Parse the JSON response to extract product search results returned by the server. Display theseresults to the user in the app's UI, allowing them to view similar products or make purchases.

- Error Handling: Handle error scenarios gracefully in your app, such as network errors or server-side errors.

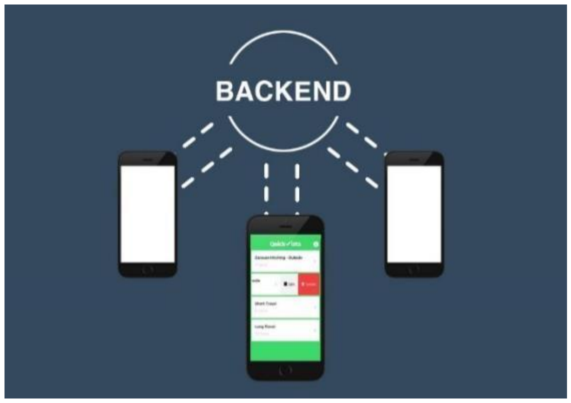


Fig 5.1 Model Backend

5.2 Build a visual product search backend using Vision API Product Search

To build a visual product search backend using Vision API Product Search, start by setting up Google Cloud Platform (GCP). Create a project on GCP if you haven't already and enable both the Vision API and Product Search API in the GCP console. Set up authentication by generating service account credentials, allowing your backend server to access these APIs securely. Once the setup is complete, proceed to upload your product data, including images and metadata, to Cloud Storage or Cloud Console. With your product data in place, use the Product Search API to create product sets and add individual products to them. These sets serve as logical groupings, while products represent the individual items in your catalog, each associated with images and metadata like descriptions and labels. Subsequently, index the product images using the Vision API Product Search, extracting features and making them searchable. Then, develop an API endpoint on your backend server to handle product search requests. Utilize the Vision API Product Search to perform similarity searches based on the provided images, querying theindexed product images to find visually similar matches. Return the search results to the client in a structured format such as JSON, containing pertinent metadata for each product. Ensure your backend infrastructure is scalable and optimized for performance, employing caching strategies and efficient indexing techniques. Lastly, prioritize security by implementing proper authentication and authorization mechanisms, utilizing HTTPS for encrypted communication, and validating user input to mitigate security risks. Through these steps, you can establish arobust visual product search backend, empowering your application to deliver accurate and relevant search results based on visual queries.

5.3 Module Description

This module provides a comprehensive guide to developing a visual product search backend using Google Cloud Platform (GCP) and Vision API Product Search. It covers all necessary steps, starting with setting up GCP by creating a project, enabling the Vision API and Product Search API, and configuring authentication through service account credentials for secure API access. The module guides you through preparing your product data by uploading images and metadata to Cloud Storage or Cloud Console and organizing your catalog into product sets and individual products, each associated with descriptions and labels.

It then explains how to index product images using the Vision API Product Search, extracting features for enhanced searchability. The development of a backend API endpoint is detailed, enabling you to handle product search requests and perform similarity searches to find visually similar products from the indexed images. Results are returned to clients in a structured JSON format, including pertinent metadata for each product. To ensure efficient performance and scalability, the module covers the implementation of caching strategies and efficient indexing techniques. Security is prioritized by outlining robust authentication and authorization mechanisms, using HTTPS for encrypted communication, and validating user inputs to mitigate security risks. By following these steps, this module enables the creation of a robust visual product search backend, empowering applications to deliver accurate and relevant search results based on visual queries.

MODULE 6

6. Go further with image classification

6.1 Build a flower recognizer

To build a flower recognizer, well start by collecting a dataset of flower images representing different species. This dataset will need to be organized into training and validation sets. Once you have your data, preprocessing steps such as resizing images to a uniform size and normalizing pixel values will be necessary to prepare them for training. Withthe data ready, you'll choose a suitable model architecture, typically a convolutional neural network (CNN) designed for image classification tasks. Options include popular architectures like AlexNet, VGG, ResNet, or MobileNet. Depending on your preference and available resources, you can either train a model from scratch or utilize a pre-trained model and fine- tune it for your specific flower recognition task. During model training, you'll split your dataset, dedicating a portion to training the model while monitoring its performance on the validation set to ensure it's not overfitting. Adjusting hyperparameters such as learning rate and batch size may be necessary to optimize training. Finally, you'll evaluate the trained model's performance on the validation set to assess its accuracy and effectiveness in recognizing different flower species.

6.2 Create a custom model for your image classifier

- Model Maker abstracts a lot of the specifics of designing the neural network so you don't have to deal with network design, and things like convolutions, dense, relu, flatten, loss functions sand optimizers.
- Creating a custom image classifier involves several key steps. First, you need to gather a dataset containing images of the classes you want to classify, such as different types of flowers. This dataset should be split into training, validation, and testing sets to ensure proper evaluation of the model's performance. Once you have your data, you can proceed to design the architecture of your custom model. This involves selecting the number and type of layers, as well as deciding on the overall structure of the network.
- After designing the architecture, you'll need to implement it using a deep learning framework like TensorFlow or PyTorch. This involves coding the layers and connections of your model according to your design specifications. Once the model is implemented, you can train it using

the training set of your dataset. During training, the model learns to map input images to their corresponding class labels through a process of optimization, adjusting the parameters of the network to minimize the loss function.

- As the model trains, you'll monitor its performance on the validation set, adjusting hyper parameters as needed to improve performance and prevent overfitting. Once training is complete and the model has achieved satisfactory performance on the validation set, you can evaluate its performance on the test set to assess its generalization ability.
- Finally, you can deploy the trained model to classify new images in real-world applications. This may involve integrating the model into a software application or deploying it to a cloud service for inference. Through these steps, you can create a custom image classifier tailored to your specific classification task.

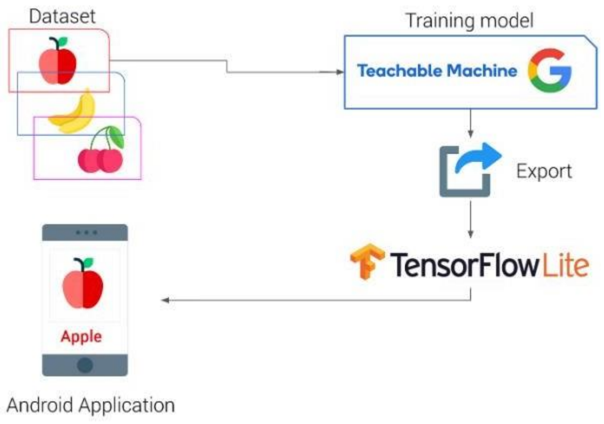


Fig 6.1 Image Classification using model

6.3 Integrate a Custom Model into your App

- Integrating a custom model into your app involves several steps to ensure seamless operation and optimal performance. First, you need to prepare your app's infrastructure to handle modelinference efficiently. This may involve setting up appropriate dependencies and libraries for running machine learning models. Once your app is ready, you can import your custom model into the project, ensuring compatibility with the chosen development framework, whether it's TensorFlow, PyTorch, or another.
- Next, you'll need to implement the necessary logic to load the model and perform inference on new input data. This typically involves initializing the model, loading its parameters from

storage, and executing forward passes to make predictions. You may need to preprocess inputdata to ensure it matches the format expected by the model.

- Once the model is integrated into your app's codebase, you can incorporate it into the user interface to provide value to your users. This could involve designing UI elements that allow users to interact with the model, such as capturing images for classification or providing inputdata through other means. Finally, thorough testing and validation are crucial to ensure that the integrated model functions as expected across various device types and usage scenarios. Through careful integration and testing, you can provide users with a seamless and reliable experience when using your app's custom ML model.

7. CASE STUDY

Plant Disease Detection

1. Problem Statement

The objective of this project is to develop a robust and efficient system for the classification of plant diseases using image processing and machine learning techniques. The system aims to identify and categorize different types of plant diseases from images of affected plant parts, primarily leaves. By leveraging advanced algorithms and a comprehensive dataset, the system should provide accurate and timely diagnoses to assist farmers and agricultural professionals in making informed decisions for disease management and crop protection.

2. Solution Overview

Agriculture is a cornerstone of the global economy, playing a critical role in supplying essential resources including food, raw materials, and employment opportunities. This sector supports millions of livelihoods and contributes significantly to national and international markets. However, the impact of plant diseases represents a formidable challenge to agricultural productivity. Plant diseases can severely diminish crop yields, degrade the quality of produce, and ultimately lead to significant economic losses for farmers and agricultural enterprises. The consequences of these diseases extend beyond immediate financial impacts, potentially affecting food availability and leading to broader food security concerns.

Early and accurate detection of plant diseases is therefore paramount for mitigating these adverse effects and promoting sustainable agricultural practices. Timely identification of disease outbreaks enables farmers to implement targeted interventions, such as appropriate treatments or preventative measures, which can prevent the spread of diseases and preserve crop health. This proactive approach not only helps to safeguard the quality and quantity of agricultural outputs but also supports long-term food security by reducing the risk of crop failures and ensuring the stability of food supplies. Advancements in diagnostic technologies and disease monitoring are critical in enhancing our ability to address these challenges effectively and maintain a resilient agricultural sector.

FLOWCHART

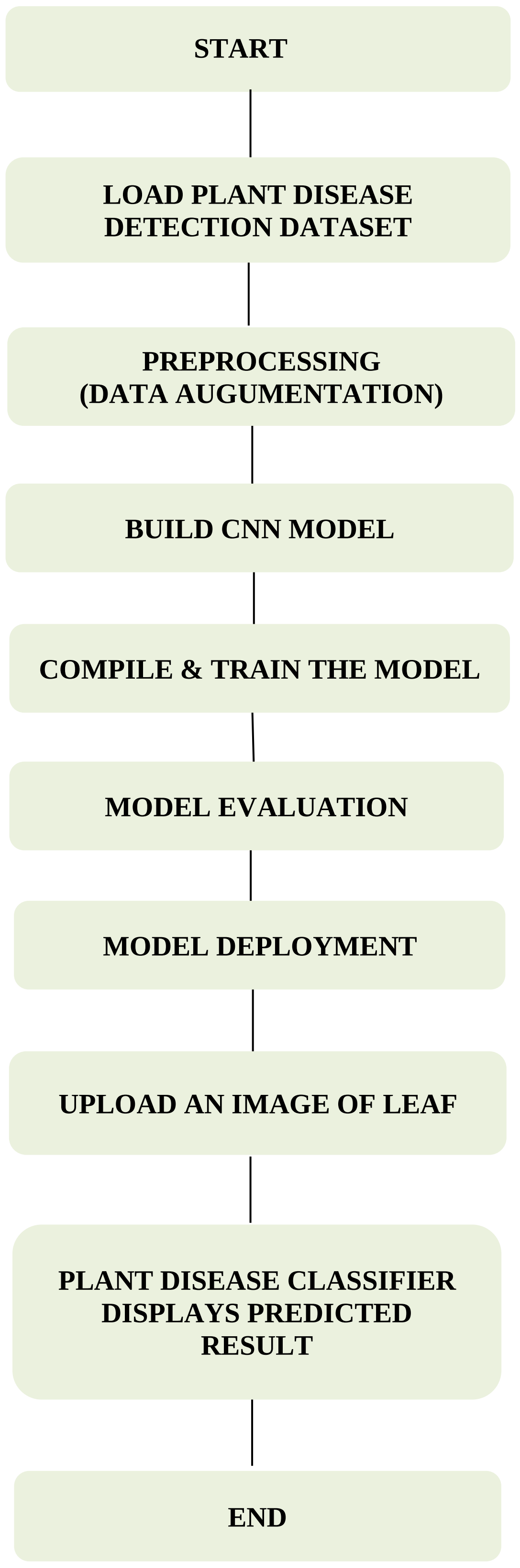

START
LOAD PLANT DISEASE DETECTION DATASET
PREPROCESSING (DATA AUGUMENTATION)
BUILD CNN MODEL
COMPILE & TRAIN THE MODEL
MODEL EVALUATION
MODEL DEPLOYMENT
UPLOAD AN IMAGE OF LEAF
PLANT DISEASE CLASSIFIER DISPLAYS PREDICTED RESULT
END

Fig 7.1 Flow chart of our model

1. Data Collection

Gathering Data

Download the dataset from Kaggle with three classes: Healthy, Rust, and Powdery.

Extract and organize the images into separate folders for each class (Healthy, Rust, Powdery).

Labeling Data

Verify that each image is correctly labeled according to its class.

If the dataset is not pre-labeled, manually label the images using a tool like Label Img or through scripts.

2. Data Preprocessing

Image Augmentation

Implement data augmentation techniques such as rotation, flipping, zooming, and shifting to increase dataset variability. Use libraries like *Keras' ImageDataGenerator* for augmentation.

Resizing and Normalization

Resize all images to a uniform size (e.g., 128x128 or 256x256 pixels) for consistency. Normalize pixel values to a range of [0, 1] to improve model training.

3. Building the Model

CNN Architecture

Design a Convolutional Neural Network (CNN) using Keras and TensorFlow.

Typical architecture

- Input Layer
- Several Convolutional Layers (e.g., Conv2D)
- Activation Functions (e.g., ReLU)
- Pooling Layers (e.g., MaxPooling2D)
- Fully Connected Layers (Dense)
- Output Layer with softmax activation for multi-class classification.

Compilation

Compile the model using an optimizer like Adam.

Use categorical crossentropy as the loss function for image multi-class classification.

Define metrics such as accuracy to evaluate model performance.

4. Training the Model

Loading Data:

Use ImageDataGenerator to load and preprocess images on-the-fly.

Implement real-time data augmentation during training.

Model Training:

Split the dataset into training and validation sets.

Train the model using the training set while validating its performance on the validation set.

Monitor metrics and adjust hyperparameters to prevent overfitting.

5. Evaluating the Model

Validation

Assess the model’s accuracy, loss, and other performance metrics on the validation set.

Ensure that the model generalizes well to new data and is not overfitting.

6. Deployment

Saving the Model:

Save the trained model using model.save('model.h5') to a file for future use.

Flask API:

- ✓ Set up a Flask web application.
- ✓ Create an endpoint for users to upload plant images.
- ✓ Load the saved model within the Flask app.
- ✓ Implement preprocessing steps (resizing, normalization) for the uploaded image.
- ✓ Use the model to predict the plant’s health or disease status and return the results to the user.

Use-case diagram

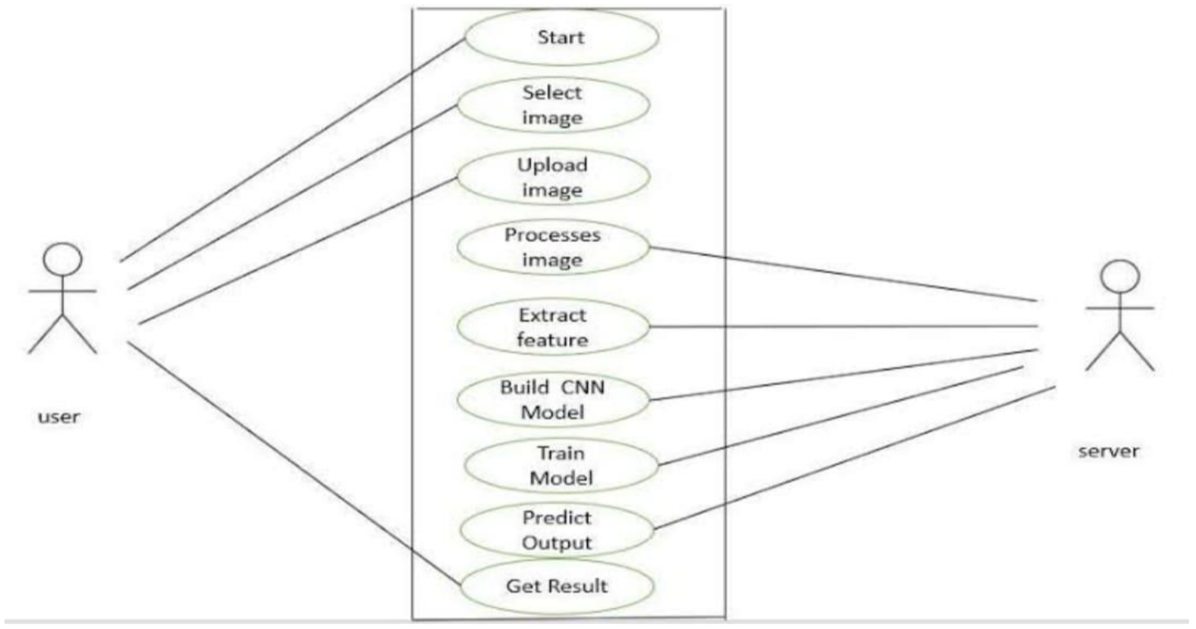
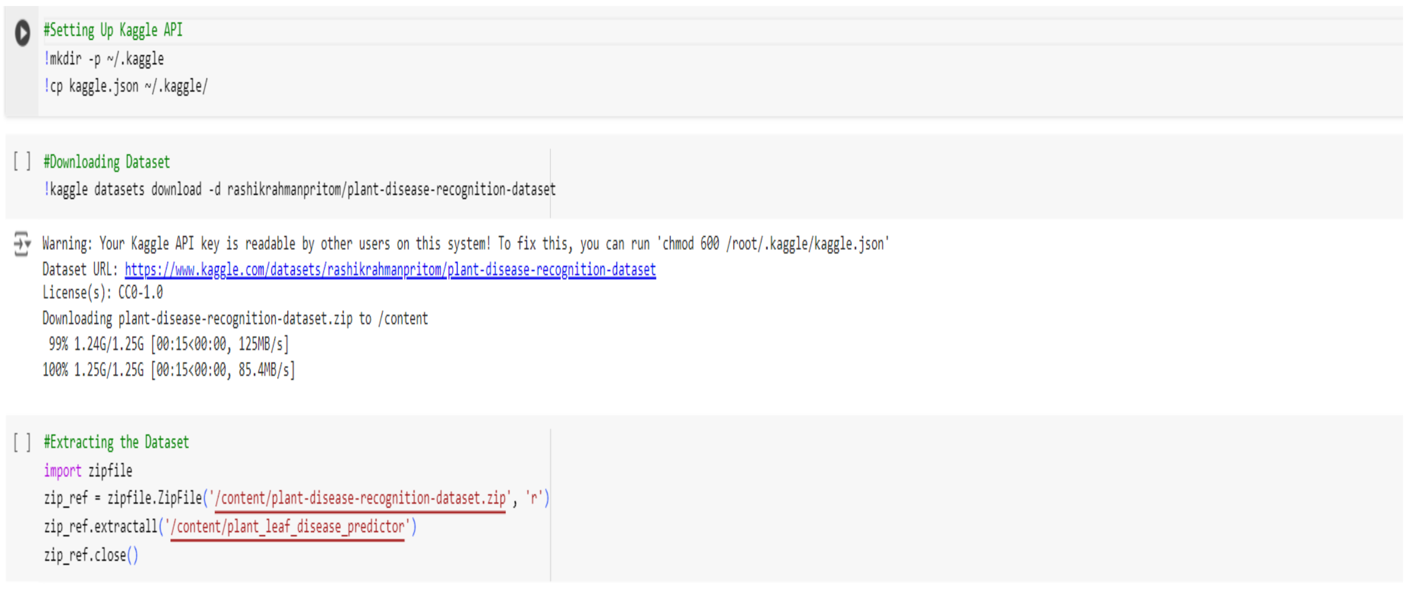


Fig 7.2 Use-case diagram of our model

7. Source Code

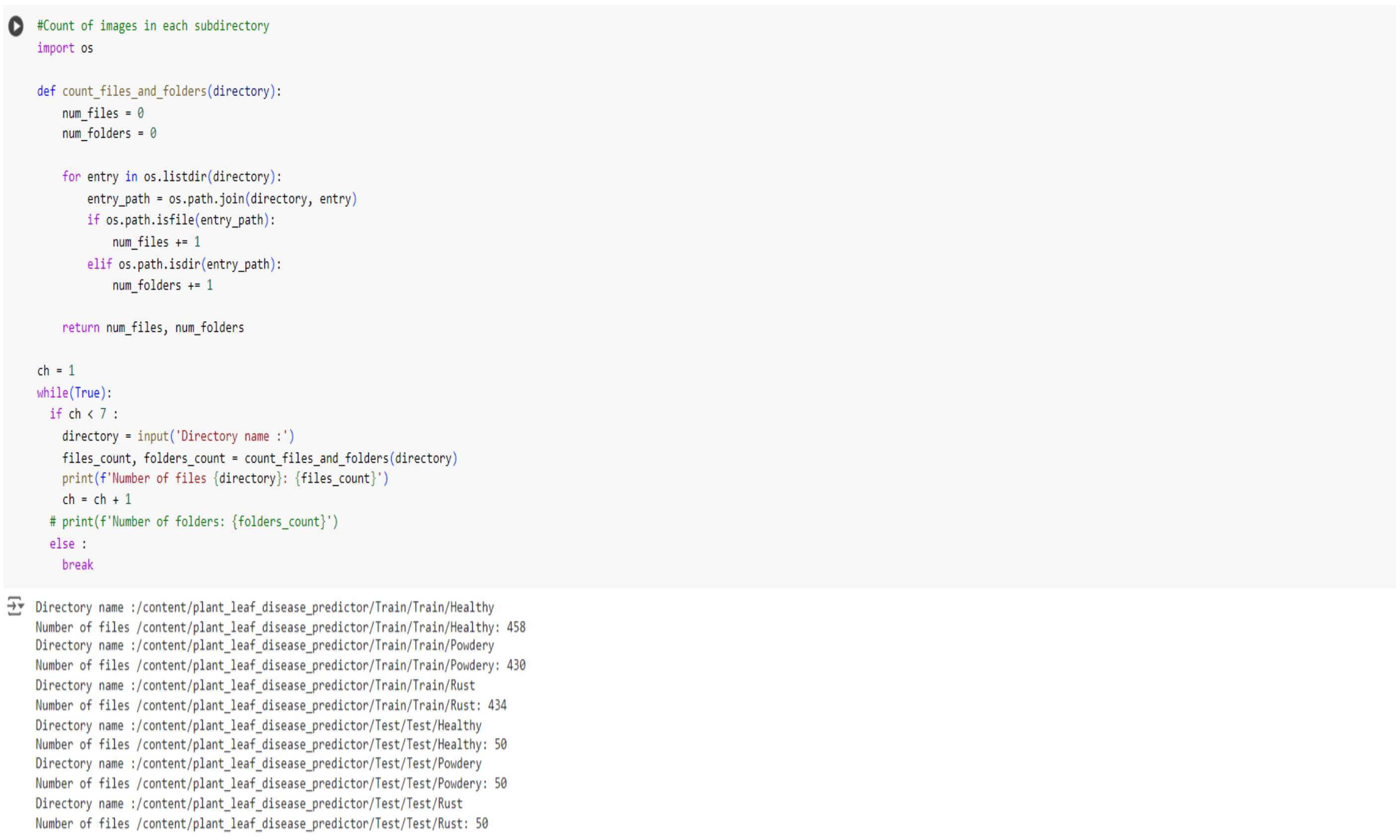


```
#Setting Up Kaggle API
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
[ ] #Downloading Dataset
    !kaggle datasets download -d rashikrahmanpritom/plant-disease-recognition-dataset
```

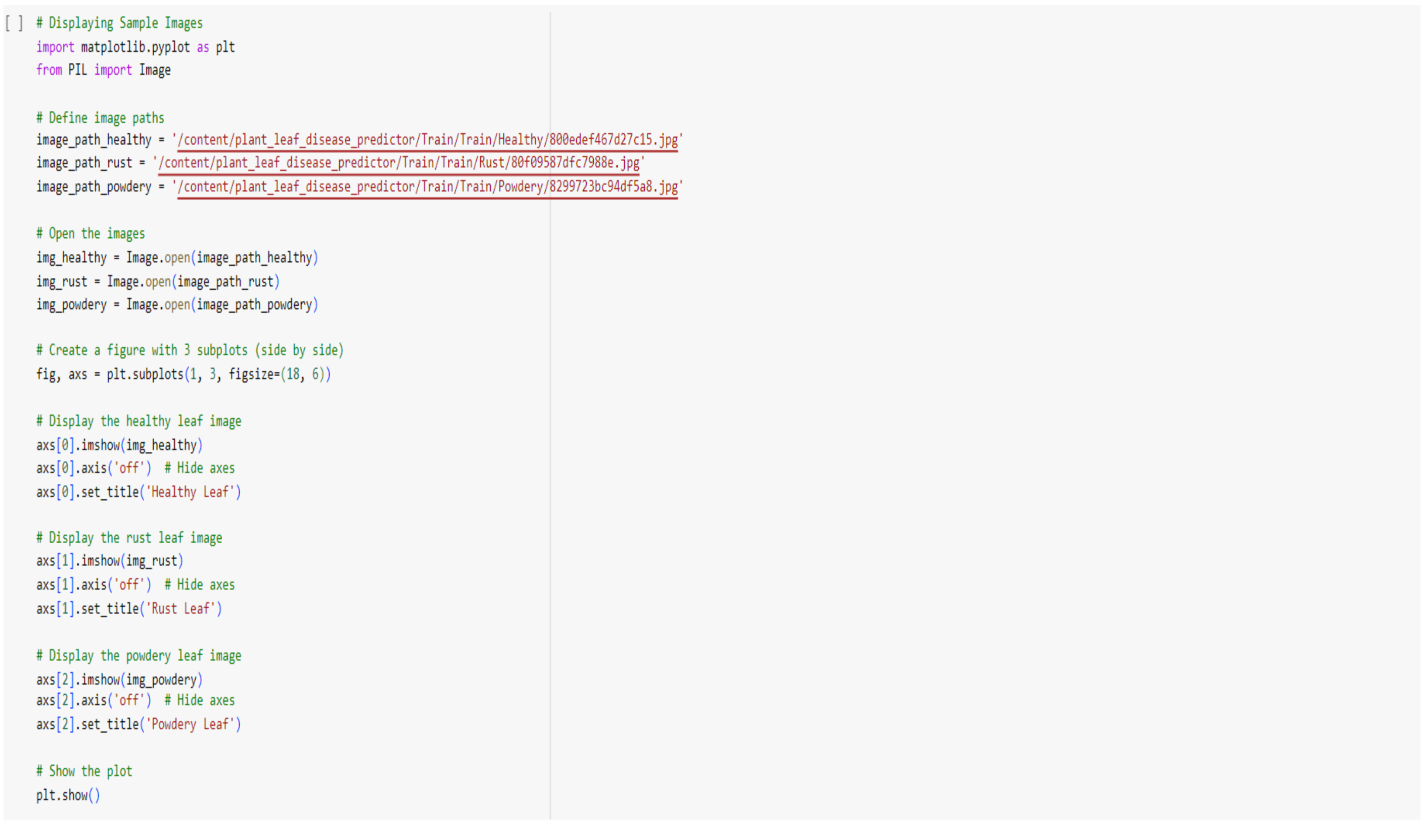
```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Dataset URL: https://www.kaggle.com/datasets/rashikrahmanpritom/plant-disease-recognition-dataset
License(s): CC0-1.0
Downloading plant-disease-recognition-dataset.zip to /content
 99% 1.24G/1.25G [00:15<00:00, 125MB/s]
100% 1.25G/1.25G [00:15<00:00, 85.4MB/s]
```

```
[ ] #Extracting the Dataset
    import zipfile
    zip_ref = zipfile.ZipFile('/content/plant-disease-recognition-dataset.zip', 'r')
    zip_ref.extractall('/content/plant_leaf_disease_predictor')
    zip_ref.close()
```



```python
#Count of images in each subdirectory
import os

def count_files_and_folders(directory):
    num_files = 0
    num_folders = 0

    for entry in os.listdir(directory):
        entry_path = os.path.join(directory, entry)
        if os.path.isfile(entry_path):
            num_files += 1
        elif os.path.isdir(entry_path):
            num_folders += 1

    return num_files, num_folders

ch = 1
while(True):
  if ch < 7 :
    directory = input('Directory name :')
    files_count, folders_count = count_files_and_folders(directory)
    print(f'Number of files {directory}: {files_count}')
    ch = ch + 1
  # print(f'Number of folders: {folders_count}')
  else :
    break
```

```
Directory name :/content/plant_leaf_disease_predictor/Train/Train/Healthy
Number of files /content/plant_leaf_disease_predictor/Train/Train/Healthy: 458
Directory name :/content/plant_leaf_disease_predictor/Train/Train/Powdery
Number of files /content/plant_leaf_disease_predictor/Train/Train/Powdery: 430
Directory name :/content/plant_leaf_disease_predictor/Train/Train/Rust
Number of files /content/plant_leaf_disease_predictor/Train/Train/Rust: 434
Directory name :/content/plant_leaf_disease_predictor/Test/Test/Healthy
Number of files /content/plant_leaf_disease_predictor/Test/Test/Healthy: 50
Directory name :/content/plant_leaf_disease_predictor/Test/Test/Powdery
Number of files /content/plant_leaf_disease_predictor/Test/Test/Powdery: 50
Directory name :/content/plant_leaf_disease_predictor/Test/Test/Rust
Number of files /content/plant_leaf_disease_predictor/Test/Test/Rust: 50
```



```python
[ ] # Displaying Sample Images
import matplotlib.pyplot as plt
from PIL import Image

# Define image paths
image_path_healthy = '/content/plant_leaf_disease_predictor/Train/Train/Healthy/800edef467d27c15.jpg'
image_path_rust = '/content/plant_leaf_disease_predictor/Train/Train/Rust/80f09587dfc7988e.jpg'
image_path_powdery = '/content/plant_leaf_disease_predictor/Train/Train/Powdery/8299723bc94df5a8.jpg'

# Open the images
img_healthy = Image.open(image_path_healthy)
img_rust = Image.open(image_path_rust)
img_powdery = Image.open(image_path_powdery)

# Create a figure with 3 subplots (side by side)
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Display the healthy leaf image
axs[0].imshow(img_healthy)
axs[0].axis('off')  # Hide axes
axs[0].set_title('Healthy Leaf')

# Display the rust leaf image
axs[1].imshow(img_rust)
axs[1].axis('off')  # Hide axes
axs[1].set_title('Rust Leaf')

# Display the powdery leaf image
axs[2].imshow(img_powdery)
axs[2].axis('off')  # Hide axes
axs[2].set_title('Powdery Leaf')

# Show the plot
plt.show()
```

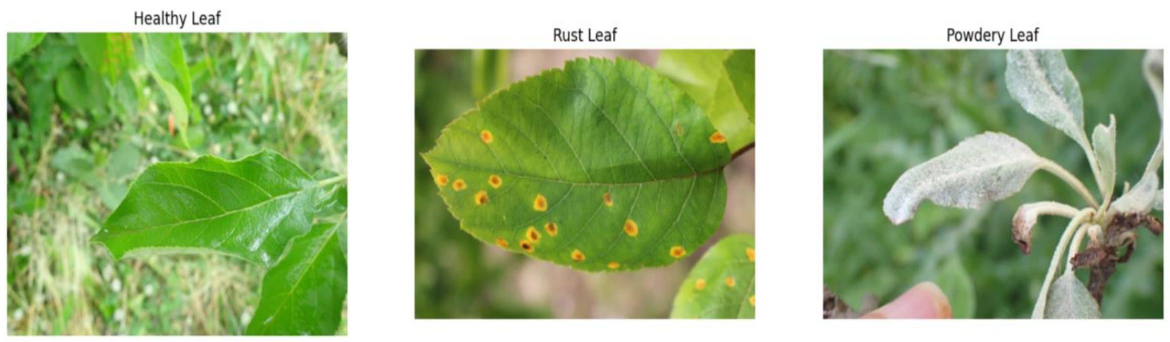
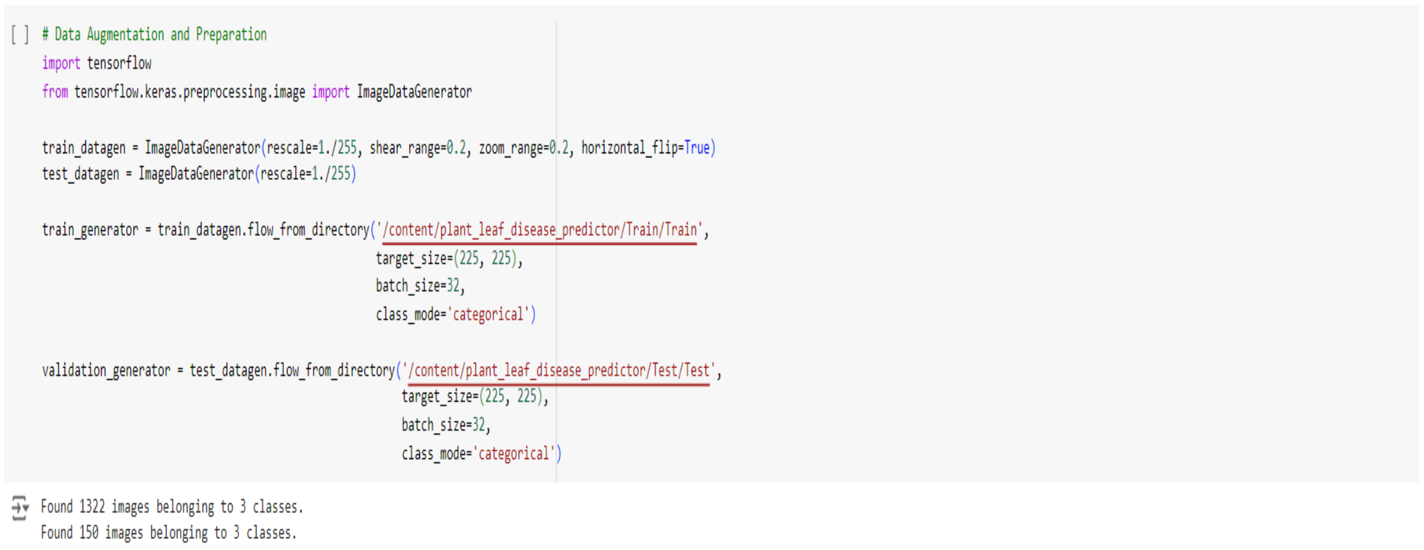
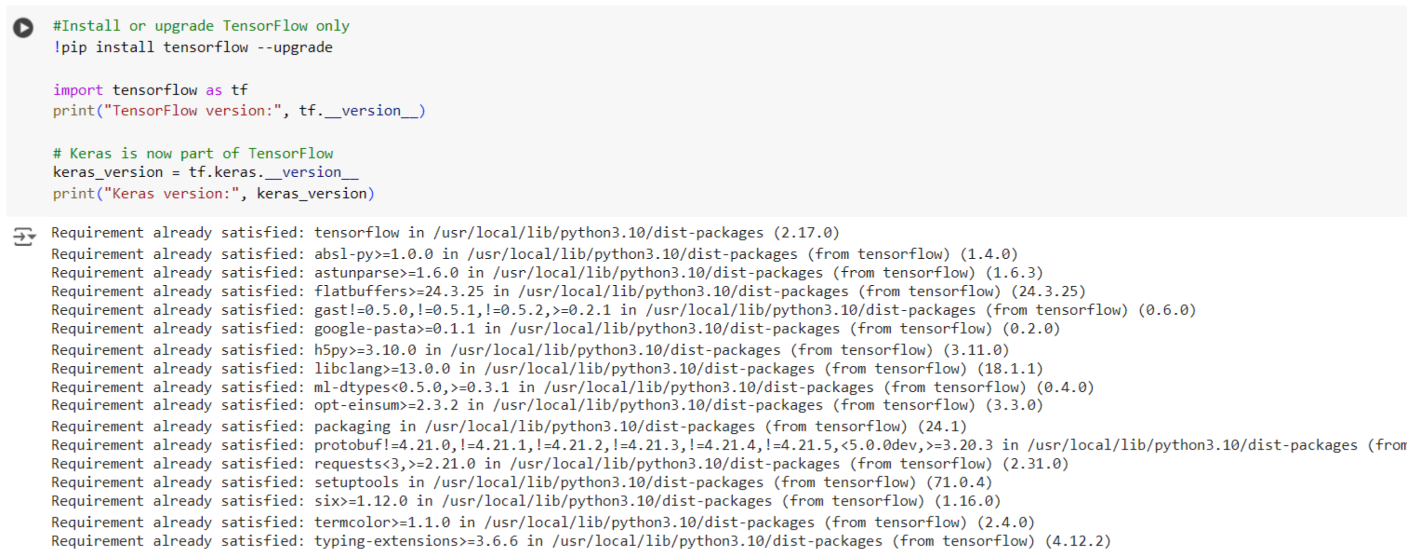


Fig 7.3 Sample images from dataset



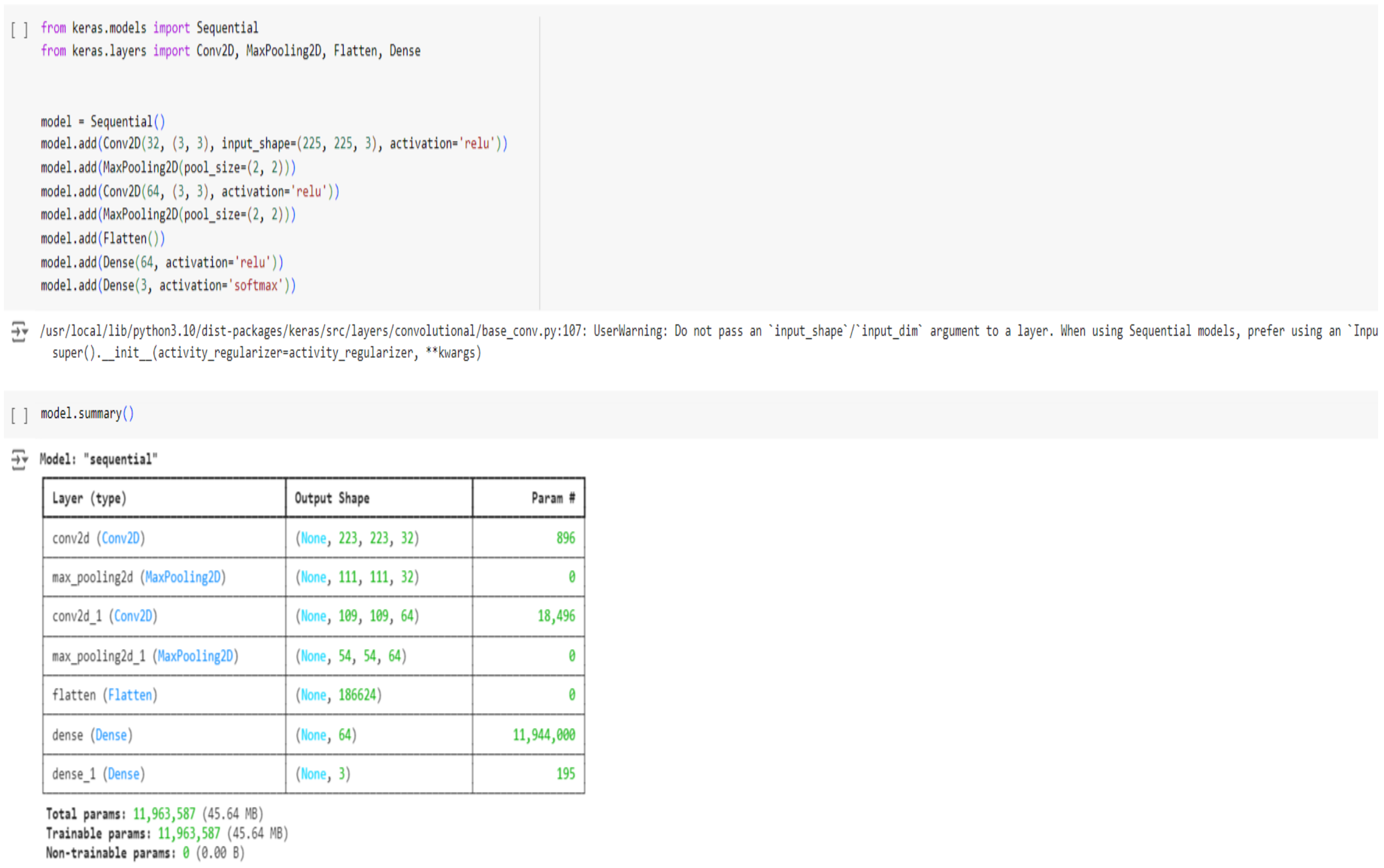
```
# Data Augmentation and Preparation
import tensorflow
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('/content/plant_leaf_disease_predictor/Train/Train',
                                                    target_size=(225, 225),
                                                    batch_size=32,
                                                    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory('/content/plant_leaf_disease_predictor/Test/Test',
                                                        target_size=(225, 225),
                                                        batch_size=32,
                                                        class_mode='categorical')
```

```
Found 1322 images belonging to 3 classes.
Found 150 images belonging to 3 classes.
```



```
#Install or upgrade TensorFlow only
!pip install tensorflow --upgrade

import tensorflow as tf
print("TensorFlow version:", tf.__version__)

# Keras is now part of TensorFlow
keras_version = tf.keras.__version__
print("Keras version:", keras_version)
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.31.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
```

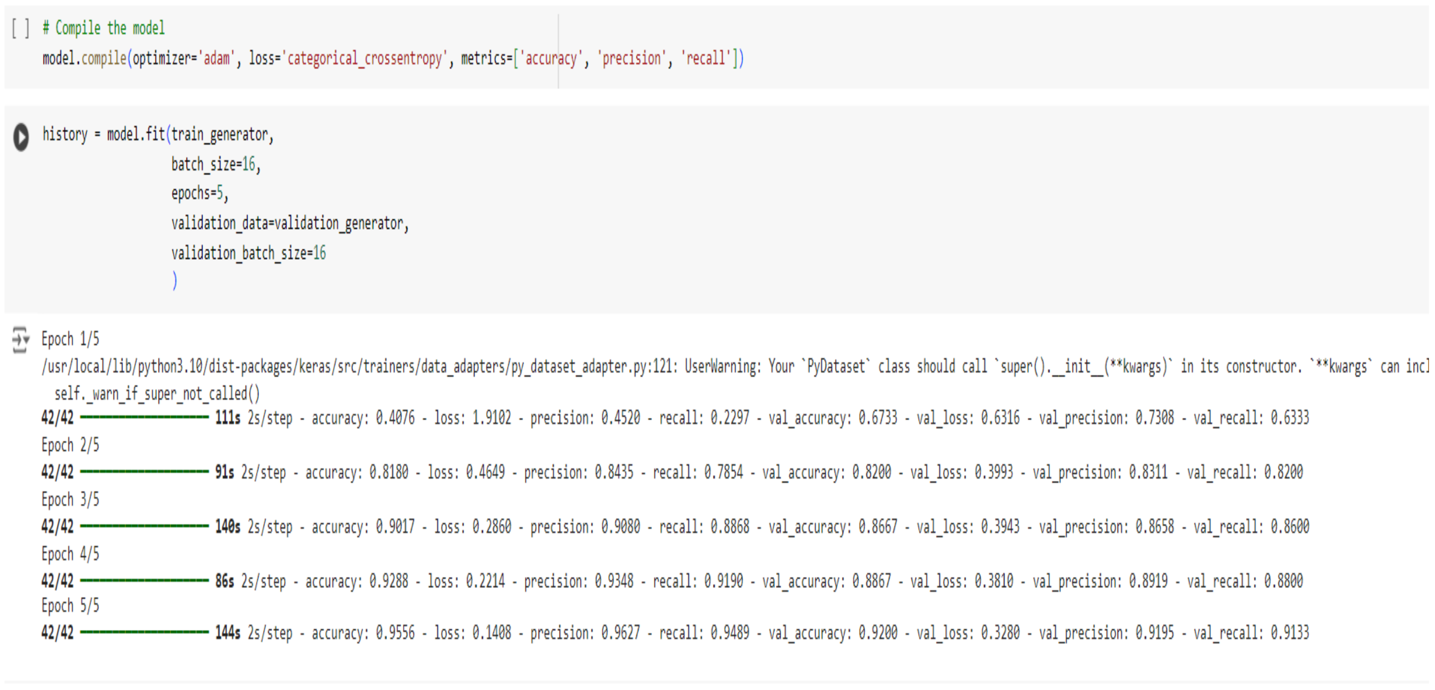


```
[ ] from keras.models import Sequential
    from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense


    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(225, 225, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(3, activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
[ ] model.summary()
```

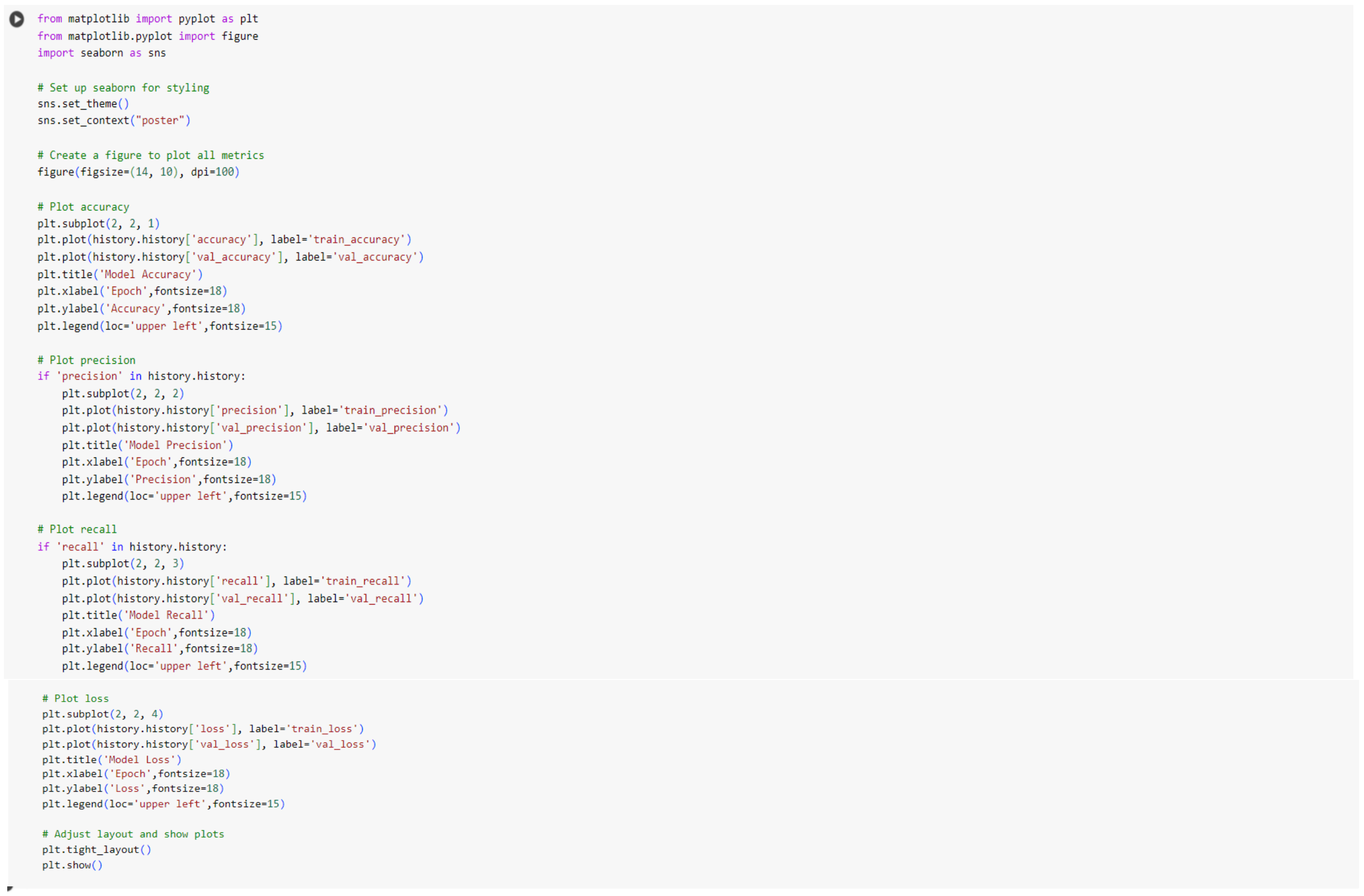
```
Model: "sequential"
┌────────────────────────────────┬───────────────────────┬─────────────┐
│ Layer (type)                   │ Output Shape          │     Param # │
├────────────────────────────────┼───────────────────────┼─────────────┤
│ conv2d (Conv2D)                │ (None, 223, 223, 32)  │         896 │
├────────────────────────────────┼───────────────────────┼─────────────┤
│ max_pooling2d (MaxPooling2D)   │ (None, 111, 111, 32)  │           0 │
├────────────────────────────────┼───────────────────────┼─────────────┤
│ conv2d_1 (Conv2D)              │ (None, 109, 109, 64)  │      18,496 │
├────────────────────────────────┼───────────────────────┼─────────────┤
│ max_pooling2d_1 (MaxPooling2D) │ (None, 54, 54, 64)    │           0 │
├────────────────────────────────┼───────────────────────┼─────────────┤
│ flatten (Flatten)              │ (None, 186624)        │           0 │
├────────────────────────────────┼───────────────────────┼─────────────┤
│ dense (Dense)                  │ (None, 64)            │  11,944,000 │
├────────────────────────────────┼───────────────────────┼─────────────┤
│ dense_1 (Dense)                │ (None, 3)             │         195 │
└────────────────────────────────┴───────────────────────┴─────────────┘
 Total params: 11,963,587 (45.64 MB)
 Trainable params: 11,963,587 (45.64 MB)
 Non-trainable params: 0 (0.00 B)
```



```
[ ] # Compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy', 'precision', 'recall'])
```

```
history = model.fit(train_generator,
                    batch_size=16,
                    epochs=5,
                    validation_data=validation_generator,
                    validation_batch_size=16
                    )
```

```
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can incl
  self._warn_if_super_not_called()
42/42 ━━━━━━━━━━━━━━━━━━━━ 111s 2s/step - accuracy: 0.4076 - loss: 1.9102 - precision: 0.4520 - recall: 0.2297 - val_accuracy: 0.6733 - val_loss: 0.6316 - val_precision: 0.7308 - val_recall: 0.6333
Epoch 2/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 91s 2s/step - accuracy: 0.8180 - loss: 0.4649 - precision: 0.8435 - recall: 0.7854 - val_accuracy: 0.8200 - val_loss: 0.3993 - val_precision: 0.8311 - val_recall: 0.8200
Epoch 3/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 140s 2s/step - accuracy: 0.9017 - loss: 0.2860 - precision: 0.9080 - recall: 0.8868 - val_accuracy: 0.8667 - val_loss: 0.3943 - val_precision: 0.8658 - val_recall: 0.8600
Epoch 4/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.9288 - loss: 0.2214 - precision: 0.9348 - recall: 0.9190 - val_accuracy: 0.8867 - val_loss: 0.3810 - val_precision: 0.8919 - val_recall: 0.8800
Epoch 5/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 144s 2s/step - accuracy: 0.9556 - loss: 0.1408 - precision: 0.9627 - recall: 0.9489 - val_accuracy: 0.9200 - val_loss: 0.3280 - val_precision: 0.9195 - val_recall: 0.9133
```



```python
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns

# Set up seaborn for styling
sns.set_theme()
sns.set_context("poster")

# Create a figure to plot all metrics
figure(figsize=(14, 10), dpi=100)

# Plot accuracy
plt.subplot(2, 2, 1)
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch',fontsize=18)
plt.ylabel('Accuracy',fontsize=18)
plt.legend(loc='upper left',fontsize=15)

# Plot precision
if 'precision' in history.history:
    plt.subplot(2, 2, 2)
    plt.plot(history.history['precision'], label='train_precision')
    plt.plot(history.history['val_precision'], label='val_precision')
    plt.title('Model Precision')
    plt.xlabel('Epoch',fontsize=18)
    plt.ylabel('Precision',fontsize=18)
    plt.legend(loc='upper left',fontsize=15)

# Plot recall
if 'recall' in history.history:
    plt.subplot(2, 2, 3)
    plt.plot(history.history['recall'], label='train_recall')
    plt.plot(history.history['val_recall'], label='val_recall')
    plt.title('Model Recall')
    plt.xlabel('Epoch',fontsize=18)
    plt.ylabel('Recall',fontsize=18)
    plt.legend(loc='upper left',fontsize=15)

# Plot loss
plt.subplot(2, 2, 4)
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model Loss')
plt.xlabel('Epoch',fontsize=18)
plt.ylabel('Loss',fontsize=18)
plt.legend(loc='upper left',fontsize=15)

# Adjust layout and show plots
plt.tight_layout()
plt.show()
```

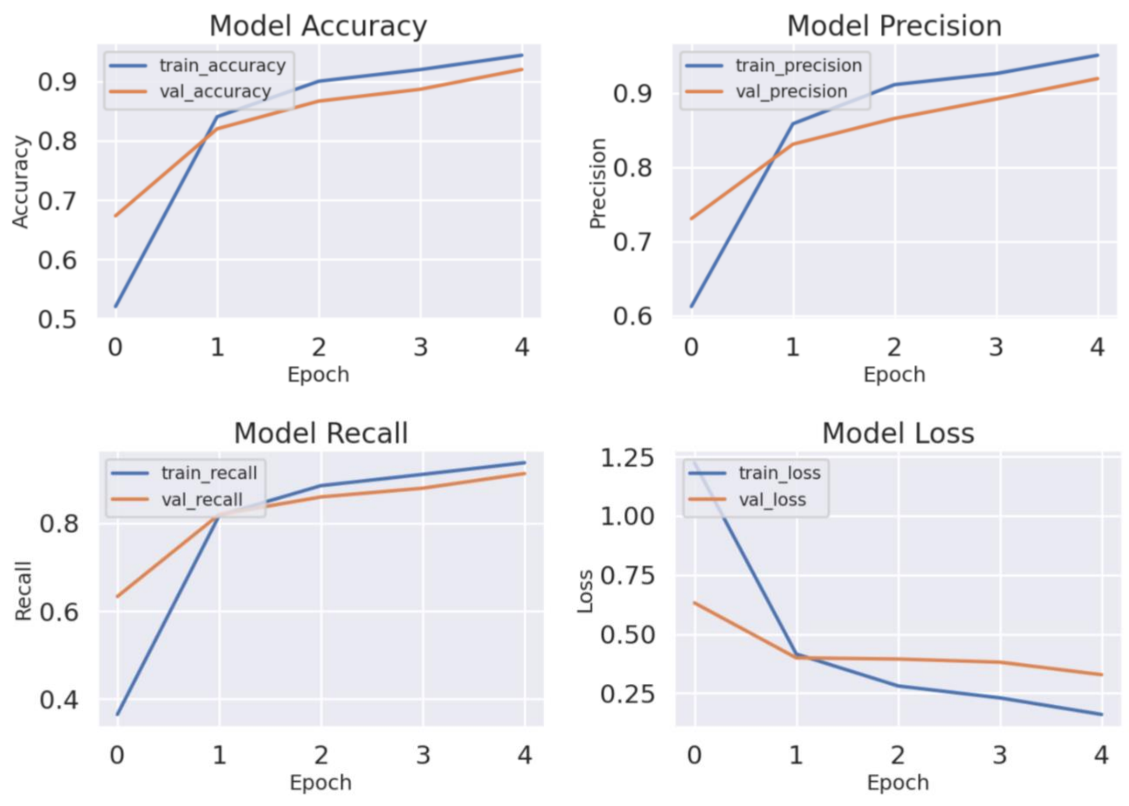
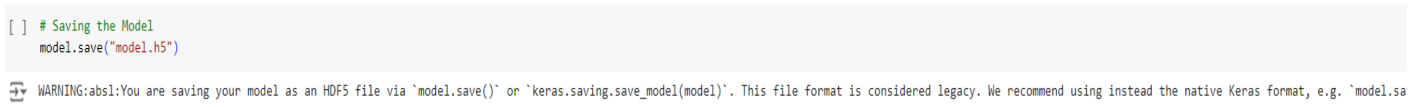
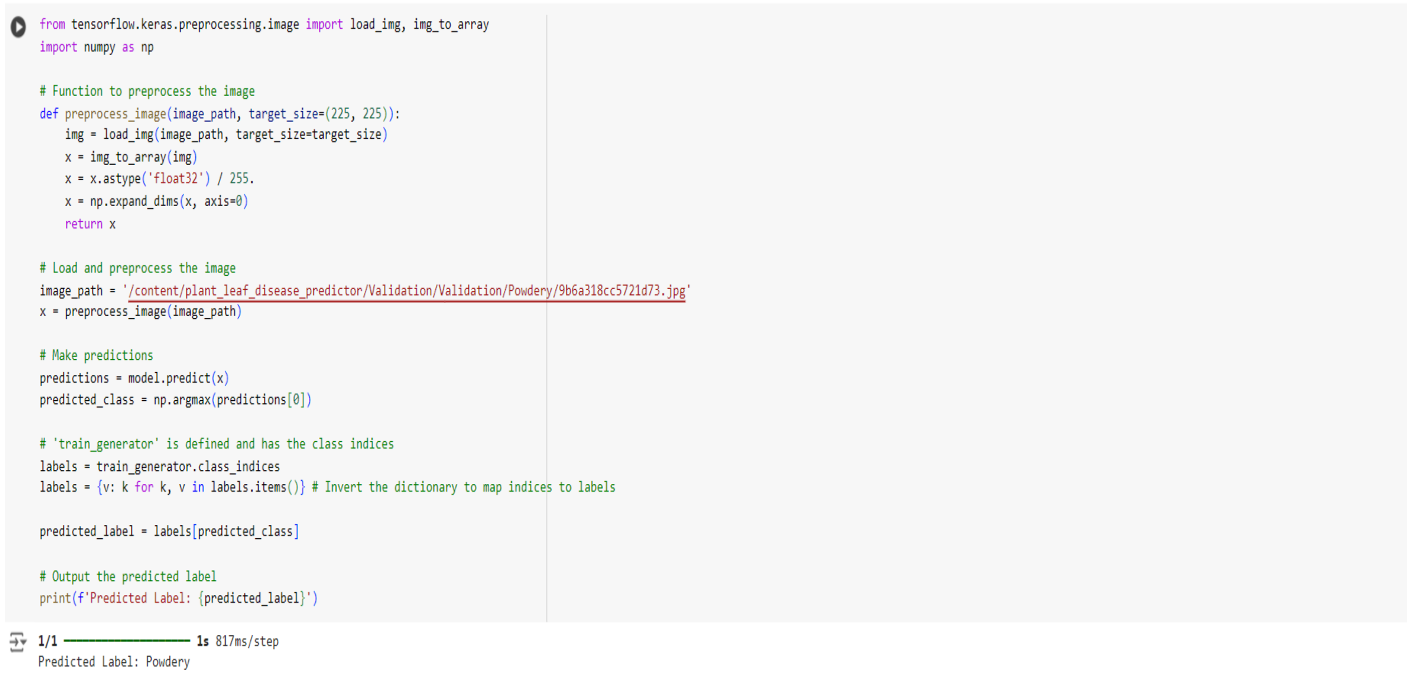


Fig 7.4 Plots of Performance Metrics

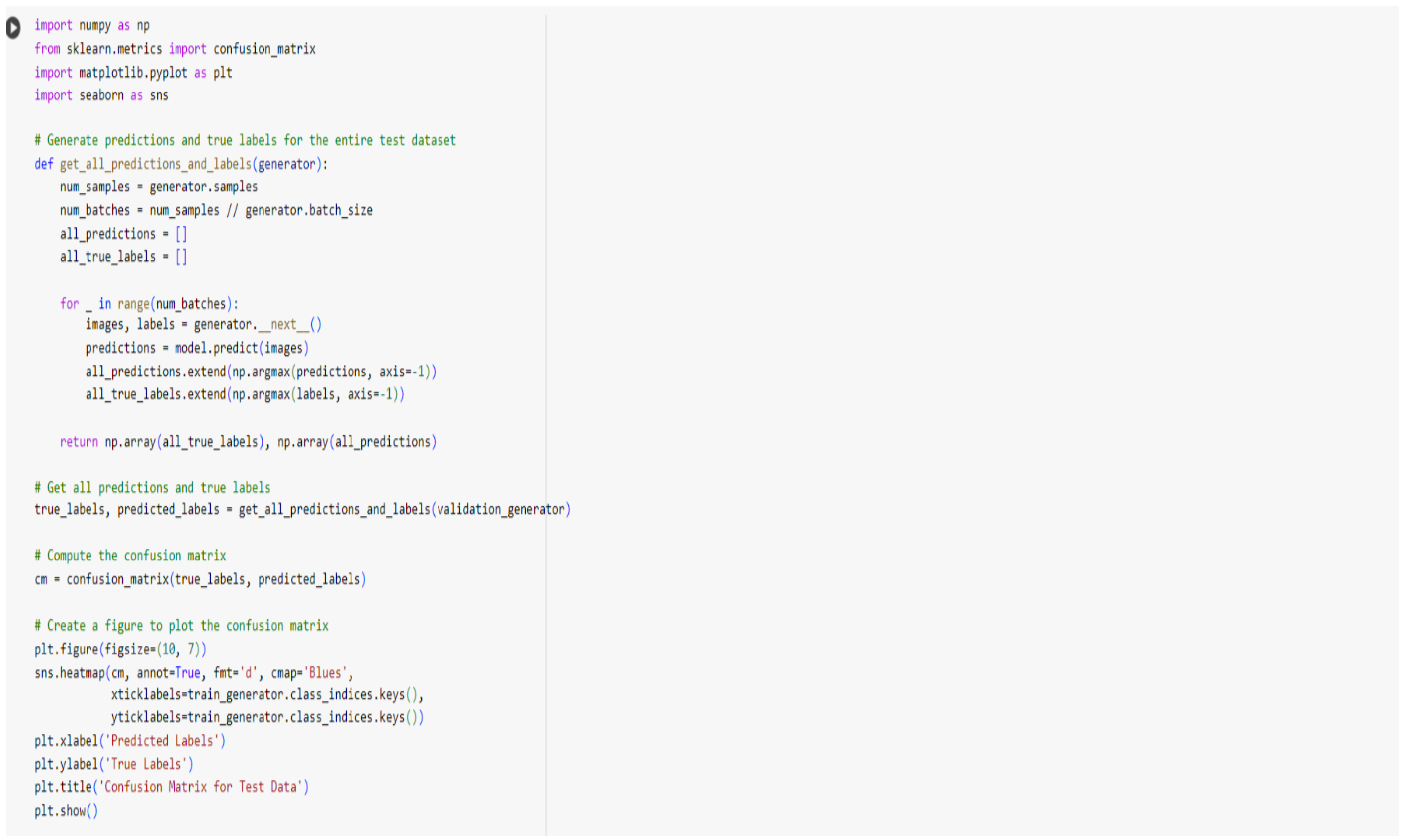


```
[ ] # Saving the Model
    model.save("model.h5")

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.sa
```

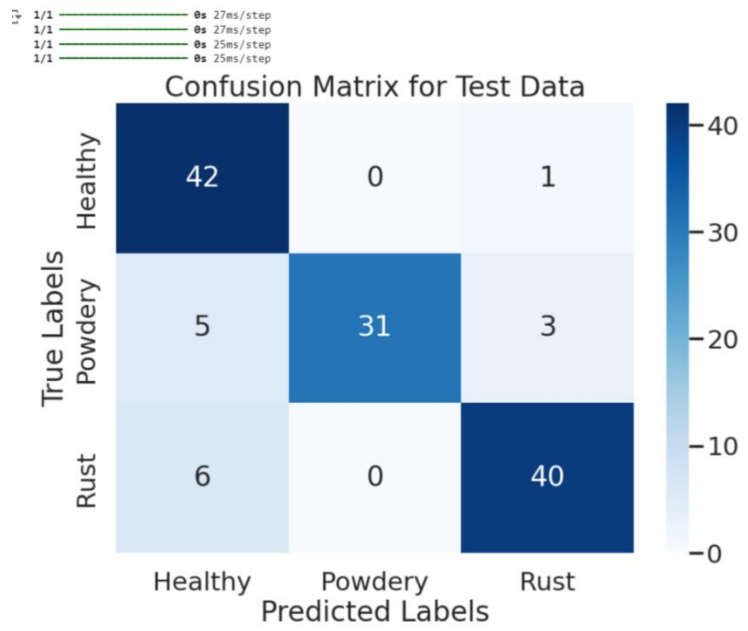


```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Function to preprocess the image
def preprocess_image(image_path, target_size=(225, 225)):
    img = load_img(image_path, target_size=target_size)
    x = img_to_array(img)
    x = x.astype('float32') / 255.
    x = np.expand_dims(x, axis=0)
    return x

# Load and preprocess the image
image_path = '/content/plant_leaf_disease_predictor/Validation/Validation/Powdery/9b6a318cc5721d73.jpg'
x = preprocess_image(image_path)

# Make predictions
predictions = model.predict(x)
predicted_class = np.argmax(predictions[0])

# 'train_generator' is defined and has the class indices
labels = train_generator.class_indices
labels = {v: k for k, v in labels.items()} # Invert the dictionary to map indices to labels

predicted_label = labels[predicted_class]

# Output the predicted label
print(f'Predicted Label: {predicted_label}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 817ms/step
Predicted Label: Powdery
```



```python
import numpy as np
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Generate predictions and true labels for the entire test dataset
def get_all_predictions_and_labels(generator):
    num_samples = generator.samples
    num_batches = num_samples // generator.batch_size
    all_predictions = []
    all_true_labels = []

    for _ in range(num_batches):
        images, labels = generator.__next__()
        predictions = model.predict(images)
        all_predictions.extend(np.argmax(predictions, axis=-1))
        all_true_labels.extend(np.argmax(labels, axis=-1))

    return np.array(all_true_labels), np.array(all_predictions)

# Get all predictions and true labels
true_labels, predicted_labels = get_all_predictions_and_labels(validation_generator)

# Compute the confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Create a figure to plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=train_generator.class_indices.keys(),
            yticklabels=train_generator.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Test Data')
plt.show()
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

Fig 7.5 Confusion Matrix

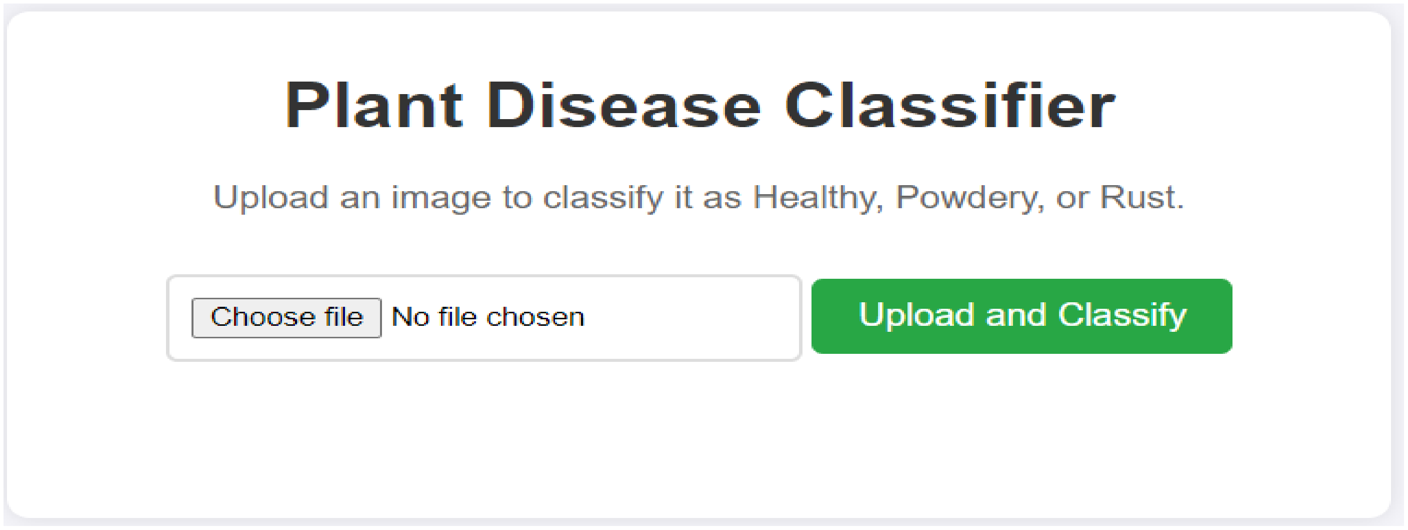
8. Conclusion

The implementation of early and precise disease detection technologies in agriculture is poised to transform the way farmers manage crop health. To support this transformation, we designed a web application for plant leaf disease detection. This proactive tool helps significantly reduce crop losses by identifying diseases early, allowing for timely and efficient interventions. Furthermore, by minimizing the need for excessive pesticide use, the application promotes a healthier environment and safer food production. Early detection and precise treatment ensure that crops remain healthy, which is vital for maximizing yields and maintaining the quality of agricultural produce.

Moreover, integrating these innovative solutions into agricultural practices supports sustainability and contributes to broader food security and economic stability. By reducing reliance on harmful chemicals, farmers can adopt more eco-friendly practices, preserving soil health and biodiversity. This project aims to address key challenges in plant disease management by providing farmers with the tools needed to maintain productive crops, ultimately supporting the agricultural community. By achieving these goals, the initiative not only revolutionizes how plant diseases are managed but also fortifies the agricultural sector's resilience against future challenges. This holistic approach ensures that farming remains a viable and sustainable livelihood, securing food supplies for communities worldwide**.**

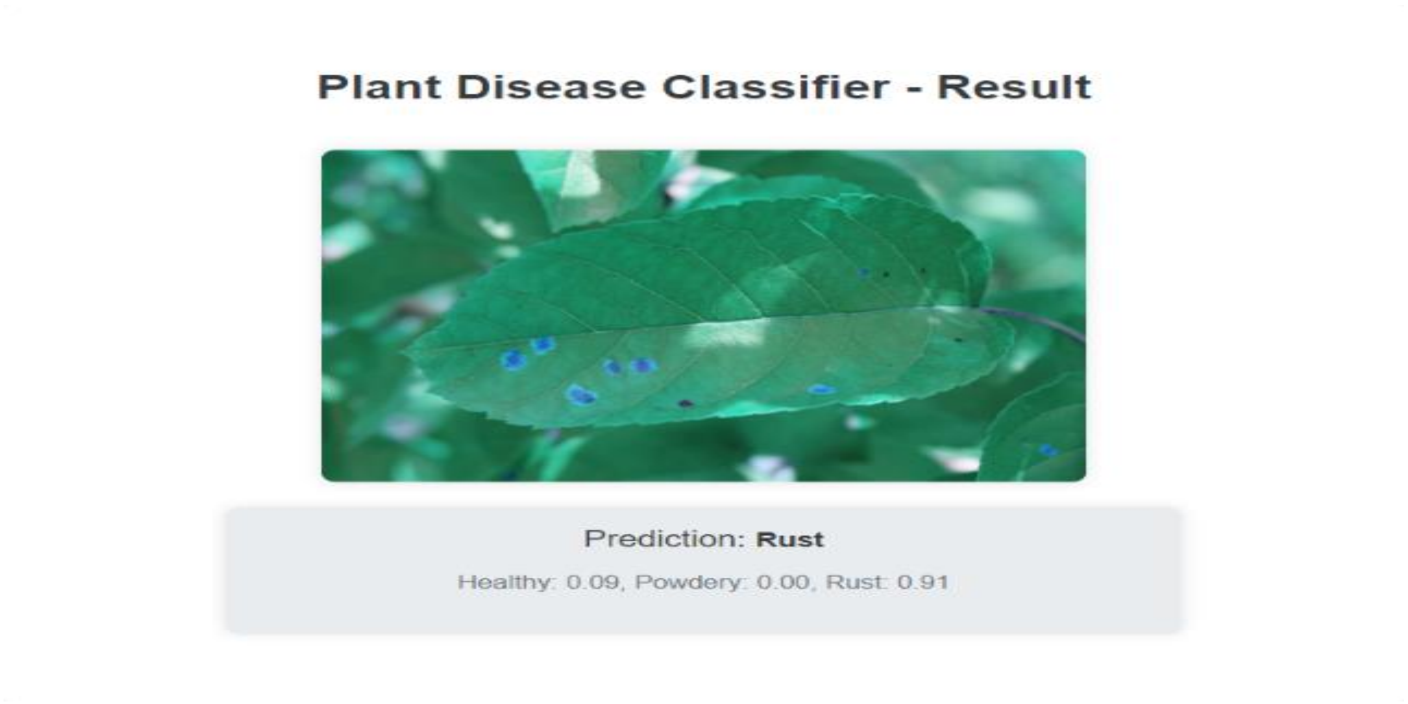
9. Outputs

We see the interface of our website as:


Plant Disease Classifier

Upload an image to classify it as Healthy, Powdery, or Rust.

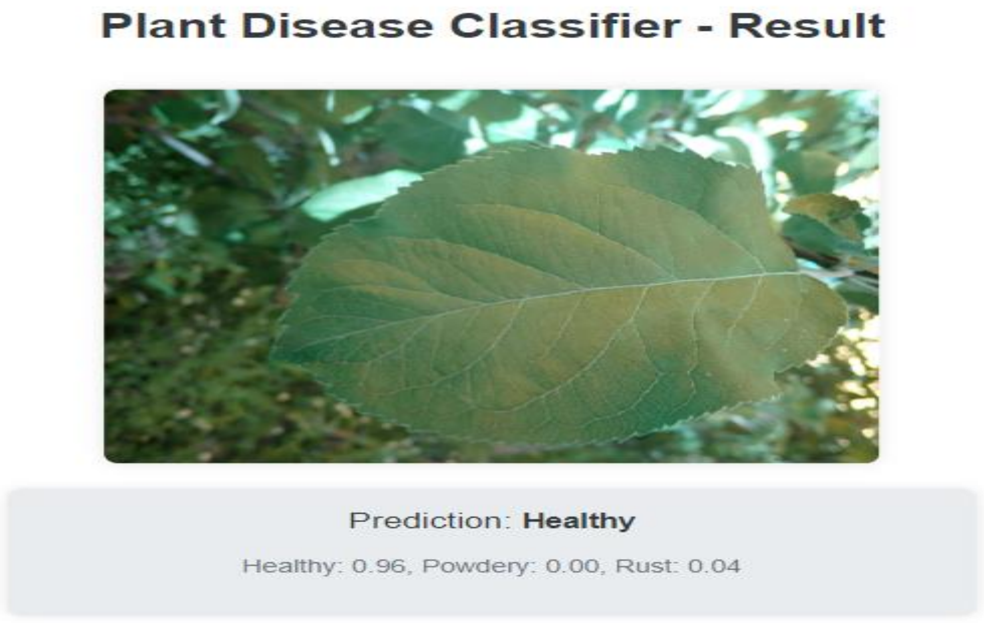
Fig 9.1 Home page


Plant Disease Classifier - Result

Prediction: **Rust**

Healthy: 0.09, Powdery: 0.00, Rust: 0.91

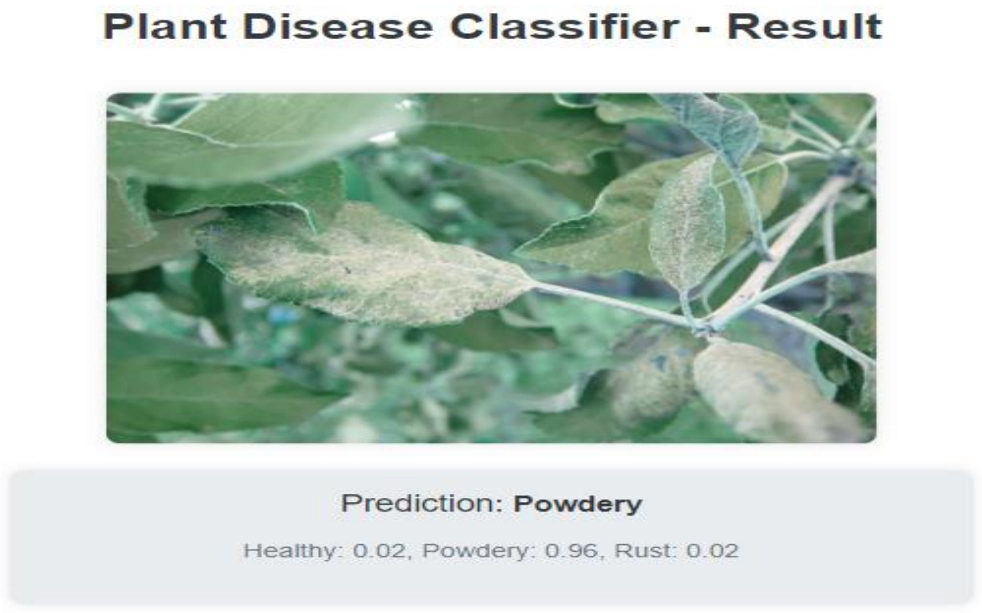
Fig 9.2 Result of Prediction as Rust


Plant Disease Classifier - Result

Prediction: **Healthy**

Healthy: 0.96, Powdery: 0.00, Rust: 0.04

Fig 9.3 Result of Prediction as Healthy


Plant Disease Classifier - Result

Prediction: **Powdery**

Healthy: 0.02, Powdery: 0.96, Rust: 0.02

Fig 9.4 Result of Prediction as Powdery

CONCLUSION

During the internship with Google AI-ML through Eduskills, We had the opportunity to immerse ourselves deeply in the field of artificial intelligence and machine learning. The internship provided a comprehensive curriculum that covered both foundational and advanced concepts, offering a robust understanding of key topics such as TensorFlow, Keras, object detection, and image classification. The structured courses were designed to bridge the gap between theoretical knowledge and practical application. Through rigorous training in TensorFlow and Keras, We gained hands-on experience in developing and fine-tuning complex AI models. This practical exposure was instrumental in translating theoretical concepts into actionable skills, allowing us to effectively handle real-world AI challenges.

Beyond the structured learning, the internship involved engaging with a series of hands-on projects that pushed us to apply the concepts we had learned in various scenarios. Working on projects related to object detection and image classification, we tackled real-world problems, which significantly honed our technical proficiency and problem-solving abilities. These projects required us to not only implement algorithms but also adapt them to specific requirements, enhancing our ability to innovate and think critically. The experience was transformative in our journey as an AI professional. It allowed us to build a strong foundation in key AI technologies while also developing practical skills that are crucial for navigating the dynamic field of artificial intelligence and machine learning. The challenges we faced and overcame during this internship have prepared us to tackle future obstacles and drive innovation within this exciting domain.

Overall, this internship has been a significant milestone in our professional growth, equipping us with the expertise and confidence to advance in the rapidly evolving landscape of AI and ML.

REFERENCE LINKS

1. https://www.researchgate.net/publication/353438370_Keras_and_TensorFlow_A_Hands-On_Experience
2. https://developers.google.com/learn/pathways/tensorflow#codelab https://developers.google.com/codelabs/tensorflow-2-computervision
3. https://developers.google.com/learn/pathways/get-started-object-detection?hl=en
4. https://developers.google.com/learn/pathways/going-further-object-detection?hl=en
5. https://developers.google.com/learn/pathways/get-started-image-product search?hl=en
6. https://developers.google.com/learn/pathways/going-further-image-product search?hl=en
7. https://developers.google.com/learn/pathways/going-further-image-classification?hl=en