THE HAZARDOUS AND NORMAL PLANT CLASSIFICATION USING TENSORFLOW AND KERAS MODEL

A PROJECT REPORT

Submitted by

MAHAA SWETHA J **(211420205085)**

NITHYASHREE K **(211420205101)**

SNEHA B **(211420205146)**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MARCH 2024

BONAFIDE CERTIFICATE

Certified that this project report **"THE HAZARDOUS AND NORMAL PLANT CLASSIFICATION USING TENSORFLOW AND KERAS MODEL"** is the bonafide work of **"MAHAASWETHA J(211420205085),NITHYASHREE K(211420205101), SNEHA B(211420205146) "** who carried out the project under my supervision.

SIGNATURE

Dr. M. HELDA MERCY M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

Department of Information Technology

Panimalar Engineering College

Poonamallee, Chennai - 600 123

SIGNATURE

Dr. K. RAMA DEVI M.E., Ph.D.,

ASSOCIATE PROFESSOR (SUPERVISOR)

Department of Information Technology Panimalar Engineering College

Poonamallee, Chennai - 600 123

SIGNATURE

INTERNAL EXAMINER

SIGNATURE

EXTERNAL EXAMINER

DECLARATION

I hereby declare that the project report entitled "**HAZARDOUS AND NORMAL PLANT CLASSIFICATION USING TENSORFLOW AND KERAS MODEL "** which is being submitted in partial fulfillment of the requirement of the course leading to the award of the 'Bachelor of Technology in Information Technology' in **Panimalar Engineering College, An Autonomous institution Affiliated to Anna University- Chennai** is the result of the project carried out by me under the guidance and supervision of **Dr. K. RAMA DEVI M.E.,Ph.D., Associate Professor in the Department of Information Technology**. I further declared that I or any other person has not previously submitted this project report to any other institution/university for any other degree/ diploma or any other person.

(Mahaaswetha J)

(Nithyashree K)

(Sneha B)

Date**:**

Place**:** Chennai

It is certified that this project has been prepared and submitted under my guidance.

Date: **Dr. K. RAMA DEVI M.E.,Ph.D.,**

Place: Chennai (Associate Professor /IT)

ACKNOWLEDGEMENT

A project of this magnitude and nature requires kind co-operation and support from many, for successful completion . We wish to express our sincere thanks to all those who were involved in the completion of this project.

Our sincere thanks to **Our Beloved Secretary and Correspondent, Dr. P. CHINNADURAI, M.A., Ph.D.,** for his sincere endeavor in educating us in his premier institution.

We would like to express our deep gratitude to **Our Dynamic Directors , Mrs. C. VIJAYA RAJESHWARI and Dr. C. SAKTHI KUMAR, M.E.,M.B.A.,Ph.D.,**and **Dr.Saranya sree sakthikumar.,B.E.,M.B.A.,Ph.D.,** for providing us with the necessary facilities for completion of this project.

We also express our appreciation and gratefulness to **Our Principal Dr. K. MANI, M.E., Ph.D.,** who helped us in the completion of the project. We wish to convey our thanks and gratitude to our head of the department, **Dr. M. HELDA MERCY, M.E., Ph.D.,** Department of Information Technology, for her support and by providing us ample time to complete our project.

We express our indebtedness and gratitude to our Project co-ordinator **Mr. M. DILLI BABU**, **M.E.,Ph.D.,** Associate Professor, Department of Information Technology for his guidance throughout the course of our project. We also express sincere thanks to our supervisor **Dr. K. RAMA DEVI**, **M.E.,Ph.D., Associate Professor** for providing the support to carry out the project successfully. Last, we thank our parents and friends for providing their extensive moral support and encouragement during the course of the project.

ABSTRACT

This study presents a robust approach for classifying hazardous and normal plants using TensorFlow and Keras models. Leveraging deep learning techniques, particularly Convolutional Neural Networks (CNNs), the proposed model aims to accurately distinguish between hazardous and normal plants based on their images. The dataset comprises a diverse range of plant images, allowing the model to learn intricate features crucial for effective classification. Through the training process, the CNN model learns to extract relevant patterns and features from plant images, enabling it to generalize to unseen data. Experimental results demonstrate the model's capability to achieve high accuracy in distinguishing between hazardous and normal plants. The utilization of TensorFlow and Keras facilitates seamless model development, training, and evaluation, contributing to an efficient and accurate plant classification system with potential applications in agriculture, forestry, and environmental monitoring.

LIST OF CONTENT

LIST OF FIGURES

LIST OF ABBREVATION

NLP	-	Natural Language Processing
CNN	-	Convolutional Neural Network
LSTM	-	Long Short-Term Memory
SVM	-	Support Vector Machine
RF	-	Random Forest
LR	-	Logistic Regression
DT	-	Decision Tree
KNN	-	K-Nearest Neighbours
MLP	-	Multilayer Perceptron
API	-	Application Programming Interface
UP	-	User Interface
GPU	-	Graphics Processing Unit
CPU	-	Central Processing Unit
IDE	-	Integrated Development Environment
HTML	-	Hypertext Markup Language
CSS	-	Cascading Style Sheets
JS	-	JavaScript
JSON	-	JavaScript Object Notation

CHAPTER 1

INTRODUCTION

1.INTRODUCTION

In the realm of machine learning, TensorFlow and Keras have emerged as powerful tools for developing sophisticated models, even in niche domains like plant classification. The task of distinguishing between hazardous and normal plants is crucial for various applications, from agriculture to environmental monitoring. Leveraging the capabilities of TensorFlow and the high-level API of Keras, one can construct a robust neural network that learns to differentiate between these two classes based on input data. By feeding the model with relevant features extracted from plant images, the system learns patterns and nuances that define hazardous and normal plants. Through the magic of deep learning, this approach enables the creation of an intelligent classifier, contributing to enhanced safety measures and decision-making processes in fields where plant classification is paramount.

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

PAPER 1

Title: Understanding Hydroponics and Its Scope in India

Author: Madhurima Maiti, Tanushree Saha

Year: 2020

Every plant needs different type of soil for its growth. Some heavy water requiring plants need clay soil, some need sandy soil where standing water can get easily drained out. This demand automatically restricts the diversification of cultivation i.e. only limited amount and limited type of crop can be produced in an area. So, the situation where this varietal need of soil type can be eliminated even without compromising on the crop's need of nutrients, minerals water etc. can be described as Hydroponics. There are 6 basic types of hydroponic systems; Wick, Water Culture, Ebb and Flow (Flood & Drain), Drip (recovery or nonrecovery), N.F.T. (Nutrient Film Technique) and Aeroponic. Hydroponics farming in India is in its rising stage. Now-a-days, in India, the progressively thinking farmers are adopting this innovative and inventive farming technique. Here the plants are totally depended upon the artificially created system. Some success stories of hydroponics farming can be considered as the confidence booster of any grower who is willing to set up this innovative hydroponic system.

PAPER 2

Title: Nutrient Solution for Hydroponics

Author: Moaed Ali Al Meselmani

Year: 2021

Hydroponics is a profitable, sustainable agricultural method and environmental friendly technology for growing plants without soil. It is the fastest-growing agriculture sector, rapidly gaining momentum and popularity, and could dominate food production in the future. Nutrient solution and its management are the cornerstone of a successful hydroponic system and are the most important determinant of crop production and quality, which is largely dependent on the extent to which plant nutrients are acquired from the nutrient solution. All nutrients in the solution in balanced ratio are supplied directly to the plants and the composition of the solution must reflect the uptake ratio of individual elements by the crop. A balanced supply of nutrients is a prerequisite for the efficient use of resources, and stabilization of the solution pH, electrical conductivity, O2 level, and temperature is essential for optimum crop yield in hydroponic systems. In this chapter, the composition of the nutrient solution, nutrient availability which is affected by many factors, and the management of the nutrient solution are discussed.

PAPER 3

Title : Hydroponics

Author: Arjina Shrestha, Bruce Dunn

Year : 2005

“HYDROPONICS” is the growing of plants in a liquid nutrient solution with or without the use of artificial media. Commonly used mediums include expanded clay, coir, perlite, vermiculite, brick shards, polystyrene packing peanuts and wood fiber. Hydroponics has been recognized as a viable method of producing vegetables (tomatoes, lettuce, cucumbers and peppers) as well as ornamental crops such as herbs, roses, freesia and foliage plants. Due to the ban on methyl bromide in soil culture, the demand for hydroponically grown produce has rapidly increased in the last few years.

PAPER 4

Title : A HYDROPONIC SYSTEM FOR INDOOR PLANT GROWTH

Author: SHREYASH MAHADEV GHATAGE, SHUBHAM R. DONE, SOHAIL AKHTAR

Year: 2019

Growing certain plants and vegetables in remote areas such as deserts and the north and South Pole can be a challenge because of the extreme outside weather. Very few species of plants thrive in such situations and are often not used as a food source. In this study, we created a system that can grow common plants and vegetables and can operate without depending on the outside climate. We achieved this by using a technique called Hydroponics. Hydroponics is a method of growing plants without using soil. System

parameters can be maintained and controlled by a sensor such as pH sensor, water temperature sensor and air temperature/humidity sensor. For adequate management of water and nutrients in the hydroponic system the electrical conductivity, pH dissolved oxygen and temperature should be measured because ion concentrations in the nutrient solutions change with time, resulting in a nutrient imbalance. In closed hydroponic systems real time measurement of all nutrients are required but such measurements are not available due to technical problems.

PAPER 5

Title: An Introduction to Small-Scale Soilless and Hydroponic Vegetable Production

Author: Natalie Bumgarner, Natalie Bumgarner

Year: 2019

> Residential and small-scale commercial food production can take many forms. Traditional home gardens that utilize native soil may be the most common, but interest in growing vegetables is not limited to those with suitable outdoor and in-ground sites. In many cases, a gardener may not have access to a plot of soil, or the soil may be of such poor quality that growing in the ground is not an option. Soilless production and hydroponics are options for many and enable small-scale vegetable production where traditional gardens would be impossible.

CHAPTER 3

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Leveraging deep learning techniques, particularly Convolutional Neural Networks (CNNs), the proposed model aims to accurately distinguish between hazardous and normal plants based on their images. The dataset comprises a diverse range of plant images, allowing the model to learn intricate features crucial for effective classification. Through the training process, the CNN model learns to extract relevant patterns and features from plant images, enabling it to generalize to unseen data. Experimental results demonstrate the model's capability to achieve high accuracy in distinguishing between hazardous and normal plants. The utilization of TensorFlow and Keras facilitates seamless model development, training, and evaluation, contributing to an efficient and accurate plant classification system with potential applications in agriculture, forestry, and environmental monitoring.

DISADVANTAGES:

1.They classify only plants based on leafs.
2.Accuracy was low.
3.They did not use multiple architecture.

3.2 PROPOSED SYSTEM

The proposed system aims to enhance industrial safety by developing a robust plant classification model using TensorFlow and Keras. This system focuses on distinguishing between hazardous and normal plants based on images of their physical attributes. By harnessing the power of deep learning, specifically Convolutional Neural Networks (CNNs), the model can effectively learn intricate patterns in plant images, enabling accurate classification. The system comprises several stages. Initially, a comprehensive dataset containing images of both hazardous and normal plants is collected and preprocessed. This dataset is then split into training and validation sets to train the CNN model. The proposed system's combination of TensorFlow and Keras, coupled with the robustness of CNNs, empowers industries to create a reliable and accurate hazardous and normal plant classification solution. This innovative approach stands to revolutionize plant safety practices by leveraging advanced AI technologies to preemptively identify risks and prioritize the well-being of workers and assets.

7.1 ADVANTAGES

- We classify normal plants and hazardous plant.
- We build a framework based application for deployment purpose
- Higher scalability
- We compared more than a two architecture to getting better accuracy level.

.

CHAPTER 4

REQUIREMENT SPECIFICATION

4.1 PROJECT REQUIREMENTS

Requirements are the basic constrains that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements

2. Non-Functional requirements

3. Environment requirements

    A. Hardware requirements

    B. software requirements

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists the requirements of a particular software system. The following details to follow the special libraries like TensorFlow, keras and , matplotlib.

Environment Requirements:

Framework : Keras.

Software Requirements:

- Operating System : Windows / Linux
- Simulation Tool : Anaconda with Jupyter Notebook
- Language : Python

Hardware requirements:

- Processor : Intel i3
- Hard disk : minimum 400 GB
- RAM : minimum 4 GB

4.2 SOFTWARE REQUIREMENTS

4.2.1ANACONDA NAVIGATOR

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows mac OS and Linux.

NAVIGATOR USAGE:

In order to run, many scientific packages depend on specific versions of other packages .Data scientists often use multiple versions of many packages, and use multiple environments to separate these different version. The command line program conda is both a package manager and an environment manager, to help data scientists ensure that each version of each package has all the dependencies It requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages and update them, all inside Navigator.

APPLICATIONS CAN BE ACCESS USING NAVIGATOR

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QTConsole
- Spyder
- VSCode
- Glueviz
- Orange 3 App
- Rodeo
- RStudio

Advanced conda users can also build your own Navigator applications

The simplest way is with Spyder. From the Navigator Home tab, click Spyder, and write and execute your code. You can also use Jupyter Notebooks the same way. Jupyter Notebooks are an increasingly popular system that combine your code, descriptive text, output, images and interactive interfaces into a single notebook file that is edited, viewed and used in a web browser.

NEW IN 1.9:

- Add support for Offline Mode for all environment related actions.
- Add support for custom configuration of main windows links.
- Numerous bug fixes and performance enhancements.

4.2.2 PYTHON

Python is a general-purpose, versatile and popular programming language. It great as a first language because it is concise and easy to read, and it is also a good language to have in any programmer & stack as it can be used for everything from web development to software development and scientific applications.

It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

FEATURES OF PYTHON

A simple language which is easier to learn, Python has a very simple and elegant syntax. It is much easier to read and write Python programs compared to other languages like: C++, Java,

Python makes programming fun and allows you to focus on the solution rather than syntax. If you are a newbie, it is a great choice to start your journey with Python.

● Free and open source

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software's written in it, you can even make changes to the Python source code. Python has a large community constantly improving it in each iteration.

● Portability

You can move Python programs from one platform to another, and run it without any changes. It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

● Extensible and Embeddable

Suppose an application requires high performance. You can easily combine pieces of C/C++or other languages with Python code. This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

● A high-level, interpreted language

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on. Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower level operations.

● Large standard libraries to solve common tasks

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server You can use MySQLdb library using import MySQL db Standard libraries in Python are well tested and used by hundreds of people. So you can be sure that it wont break your application.

● Object-oriented

Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively. With OOP, you are able to divide these complex problems into smaller sets by creating object.

4.2.3 NUMPY

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random

simulation and much more. At the core of the NumPy package, is the array object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of and array will create a new array and delete the original.

- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.

- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.

- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

The points about sequence size and speed are particularly important in scientific computing. As a simple example, consider the case of multiply in each element in a 1D sequence with the corresponding element in another sequence of the same length. If the data are stored in two Python lists, a and b, we could iterate over each element.

The Numeric Python extensions (NumPy henceforth) is a set of extensions to the

Python programming language which allows Python programmers to efficiently manipulate large sets of objects organized in grid-like fashion. These sets of objects are called arrays, and they can have any number of dimensions: one dimensional arrays are similar to standard Python sequences, two-dimensional arrays are similar to matrices from linear algebra. Note that one-dimensional arrays are also different from any other Python sequence, and that two-dimensional matrices are also different from the matrices of linear algebra, in ways which we will mention later in this text. Why are these extensions needed? The core reason is a very prosaic one, and that is that manipulating a set of a million numbers in Python with the standard data structures such as lists, tuples or classes is much too slow and uses too much space. Anything which we can do in NumPy we can do in standard Python – we just may notbe alive to see the program finish.

A more subtle reason for these extensions however is that the kinds of operations that programmers typically want to do on arrays, while sometimes very complex, can often be decomposed into a set of fairly standard operations. This decomposition has been developed similarly in many array languages. In some ways, NumPy is simply the application of this experience to the Python language – thus many of the operations described in NumPy work the way they do because experience has shown that way to be a good one, in a variety of contexts. The languages which were used to guide the development of NumPy include the infamous APL family of languages, Basis, MATLAB, FORTRAN, S and S+, and others.

This heritage will be obvious to users of NumPy who already have experience with these other languages. This tutorial, however, does not assume any such background, and all that is expected of the reader is a reasonable working knowledge of the standard Python language. This document is the “official” documentation for NumPy. It is both a tutorial and the most authoritative source of information about NumPy with the exception of the source code. The tutorial material will walk you through a set of manipulations of simple, small, arrays of numbers, as well as image files.

This choice was made because:

- A concrete data set makes explaining the behaviour of some functions much easier to motivate than simply talking about abstract operations on abstract data sets;
- Every reader will at least an intuition as to the meaning of the data and organization of image files.
- The result of various manipulations can be displayed simply since the data set has a natural graphical representation. All users of NumPy, whether interested in image processing or not, are encouraged to follow the tutorial with a working NumPy installation at their side, testing the examples, and, more importantly, transferring the understanding gained by working on images to their specific domain. The best way to learn is by doing – the aim of this tutorial is to guide you along this "doing."

4.3 HARDWARE ENVIRONMENT

In our project predictive models for early detection machine learning-based dyslexia prediction using Flask is designed to support the development and deployment of our system. Given the nature of our project, the hardware requirements are relatively modest and flexible. Our project can be implemented on standard desktop or laptop computers with moderate computational capabilities. These computers should have sufficient processing power and memory to handle the tasks involved in data preprocessing, model training, and web application deployment. Additionally, access to a stable internet connection is essential for data collection, model updates, and web application hosting. While specialized hardware configurations are not necessary for our project, having access to modern processors and an adequate amount of RAM can expedite the training and inference processes, especially when working with large datasets or complex machine learning models.

CHAPTER 5

SYSTEM DESIGN

5.1 DATA FLOW DIAGRAM

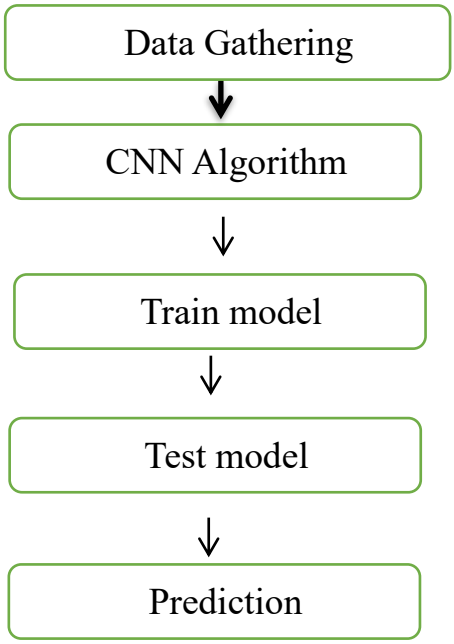


Figure 5.1 Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model. Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top down approach to Systems Design. Symbols and Notations Used in DFDs Using any convention's DFD rules or guidelines, the symbols depict the four components of data flow diagrams.

External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

Data store: files or repositories that hold information for later use, such as a database table or a membership form.

Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details.”

5.2 SYSTEM ARCHITECTURE

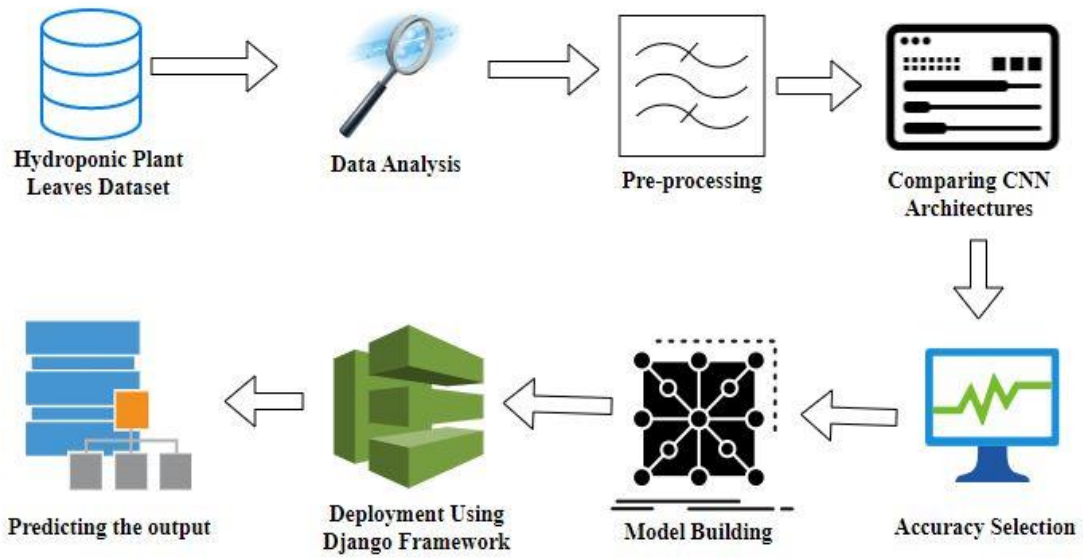


Figure 5.2 System Architecture Diagram

5.3 USE CASE DIAGRAM

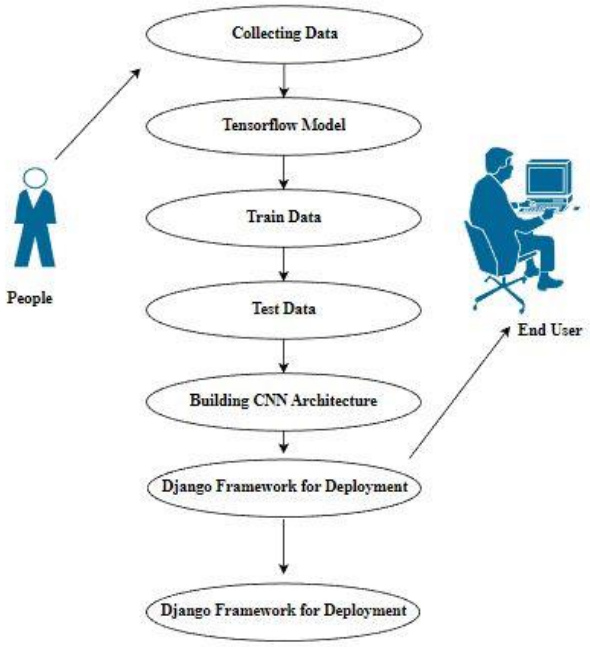


FIGURE 5.3: USECASE DIAGRAM

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

5.4 CLASS DIAGRAM

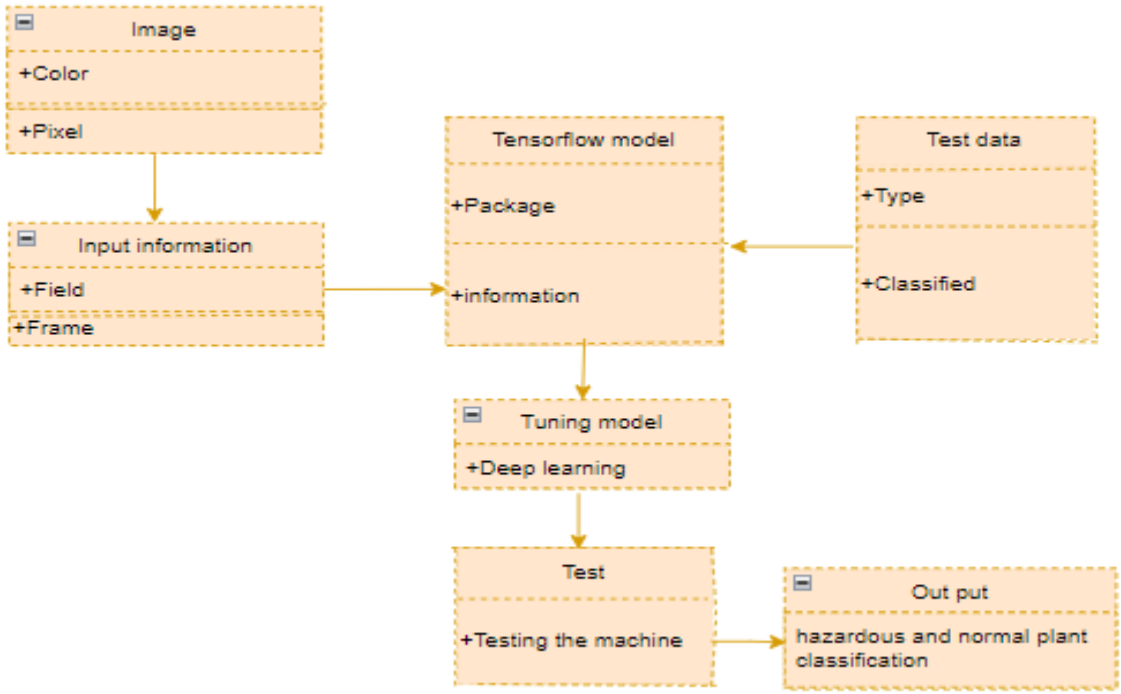


FIGURE 5.4: CLASS DIAGRAM

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the

diagram should be drawn on plain paper and rework as many times as possible to make it correct.

5.5 SEQUENCE DIAGRAM

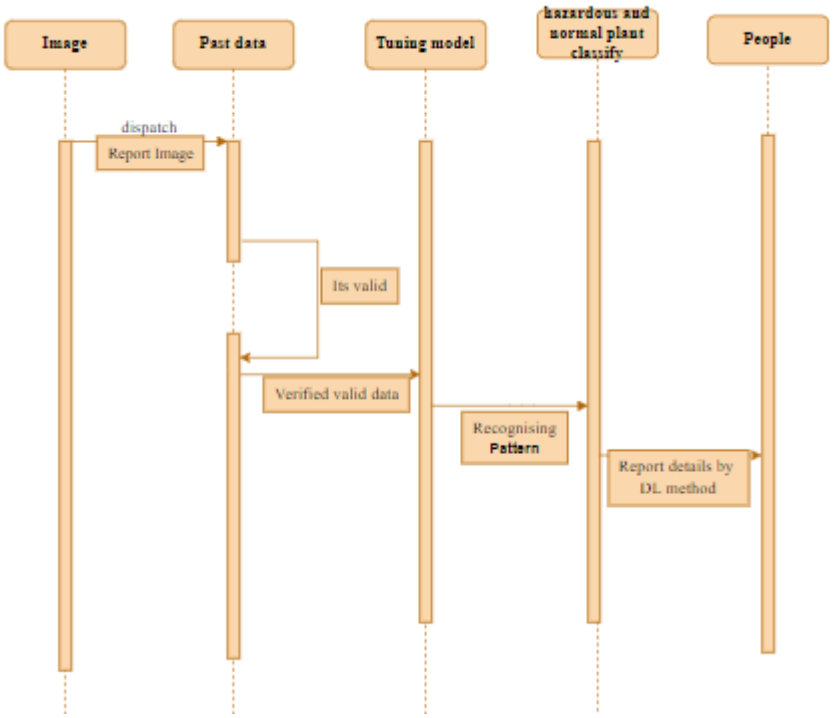


FIGURE 5.5: SEQUENCE DIAGRAM

Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behaviour within your system. Other

dynamic modelling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

5.6 ER DIAGRAM:

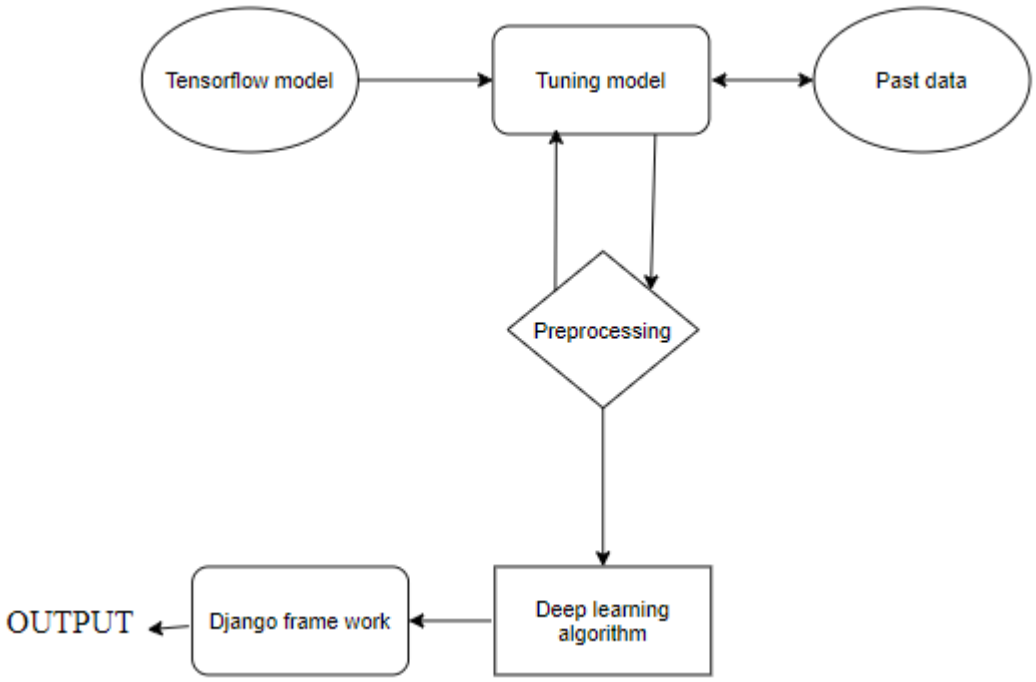


FIGURE 5.6: ER DIAGRAM

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for

database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

5.7 COLLABORATION DIAGRAM:

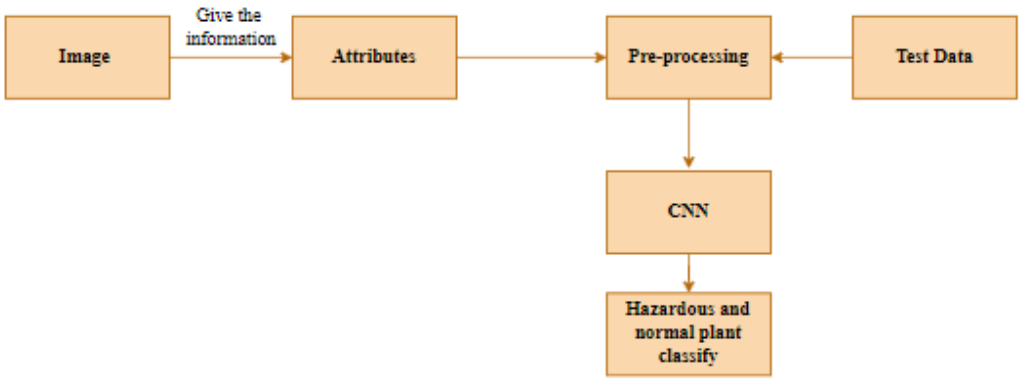


FIGURE 5.7: COLLABORATION DIAGRAM

A collaboration diagram show the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction.

The collaboration diagram can be a decomposition of a class, class diagram, or part of a class diagram.it can be the decomposition of a use case, use case diagram, or part of a use case diagram.

The collaboration diagram shows messages being sent between classes and object (instances). A diagram is created for each system operation that relates to the current development cycle (iteration).

CHAPTER 6

ARCHITECTURE AND SOFTWARE DESCRIPTION

6.1ALEXNET:

AlexNet is the name of a convolutional neural network which has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. AlexNet was the first convolutional network which used GPU to boost performance.

AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU. The pooling layers are used to perform max pooling.

Architecture of AlexNet:

Fig 6.1 Architecture of AlexNet

Convolutional layers:

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

Pooling layers:

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

Dense or Fully connected layers:

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

6.2 RESNET

ResNet is the name of a convolutional neural network which has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. ResNet was the first convolutional network which used GPU to boost performance.

ResNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer. Each

convolutional layer consists of convolutional filters and a nonlinear activation function ReLU. The pooling layers are used to perform max pooling.

Architecture of ResNet:

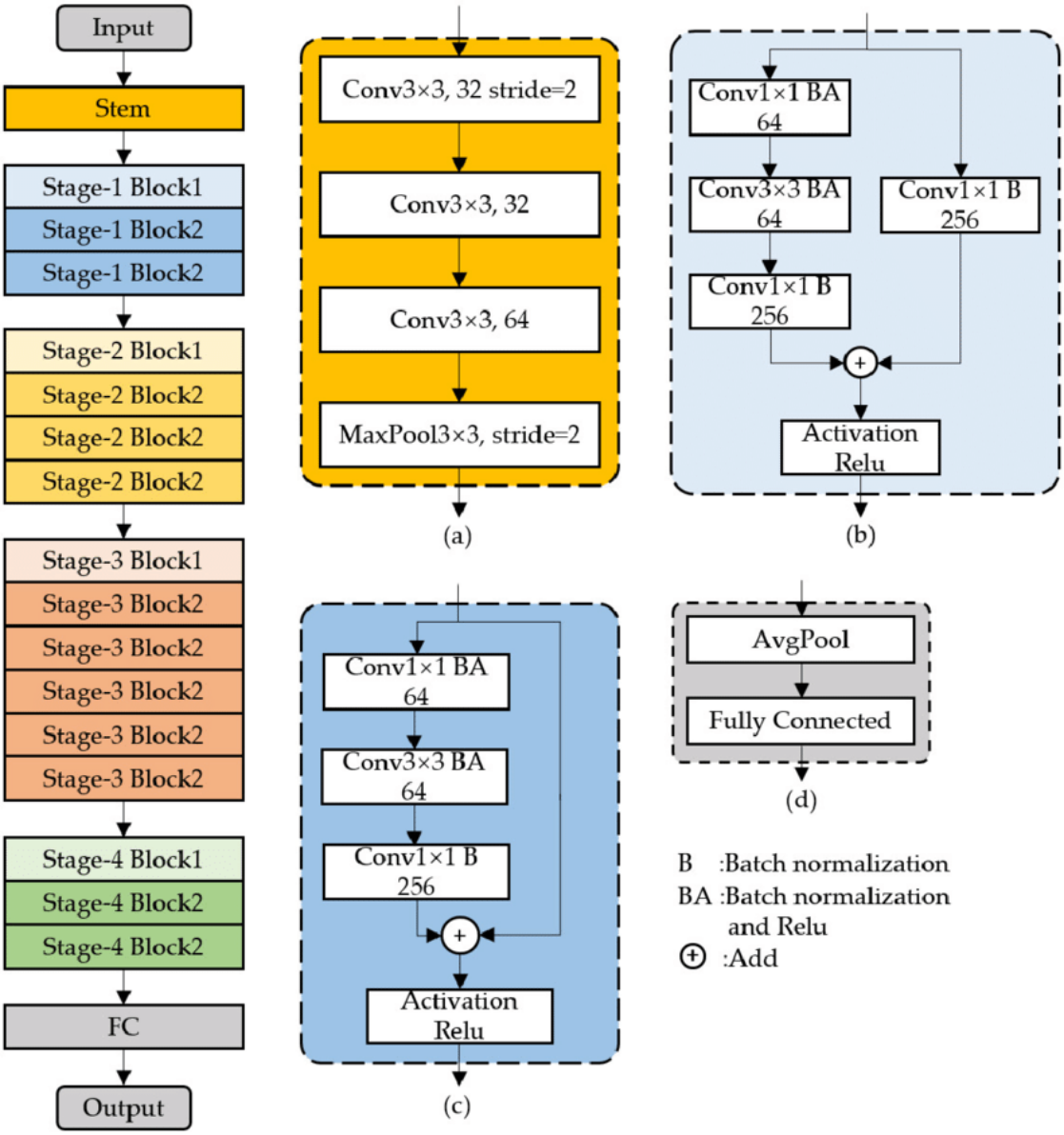


Fig 6.2.Architecture of ResNet

Convolutional layers:

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

Pooling layers:

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

Dense or Fully connected layers:

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

6.3 SOFTWARE DESCRIPTION:

The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400 packages as well as the Conda package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the conda

install command or using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the `conda build` command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

6.3.1ANACONDA NAVIGATOR:

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

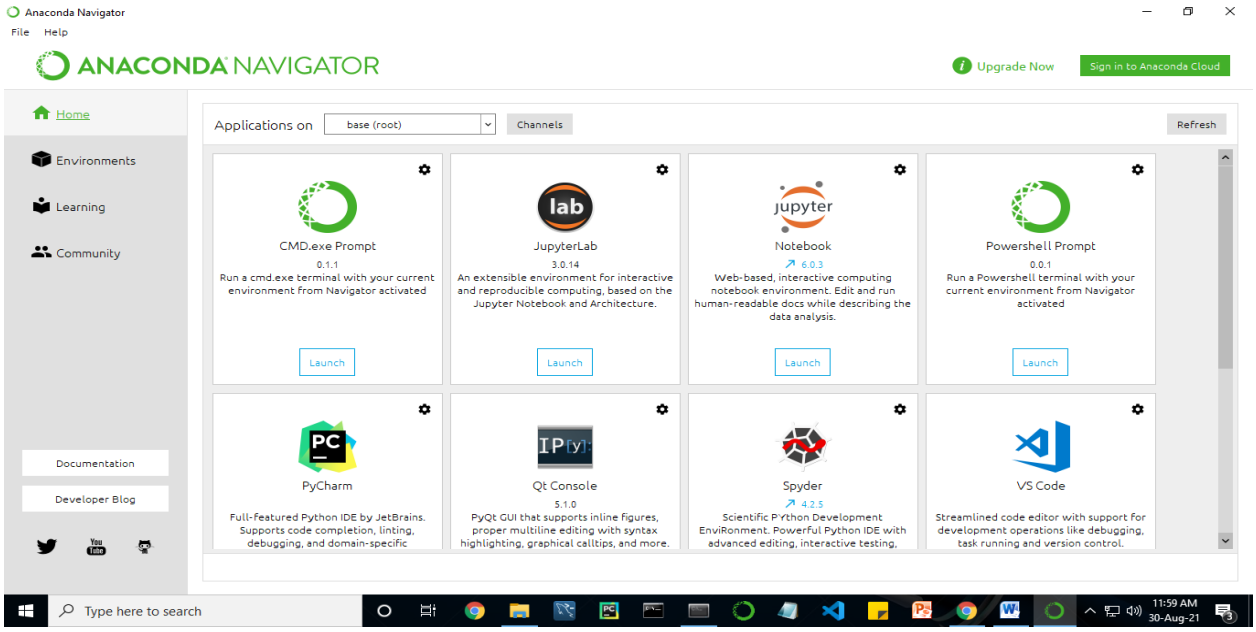
Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

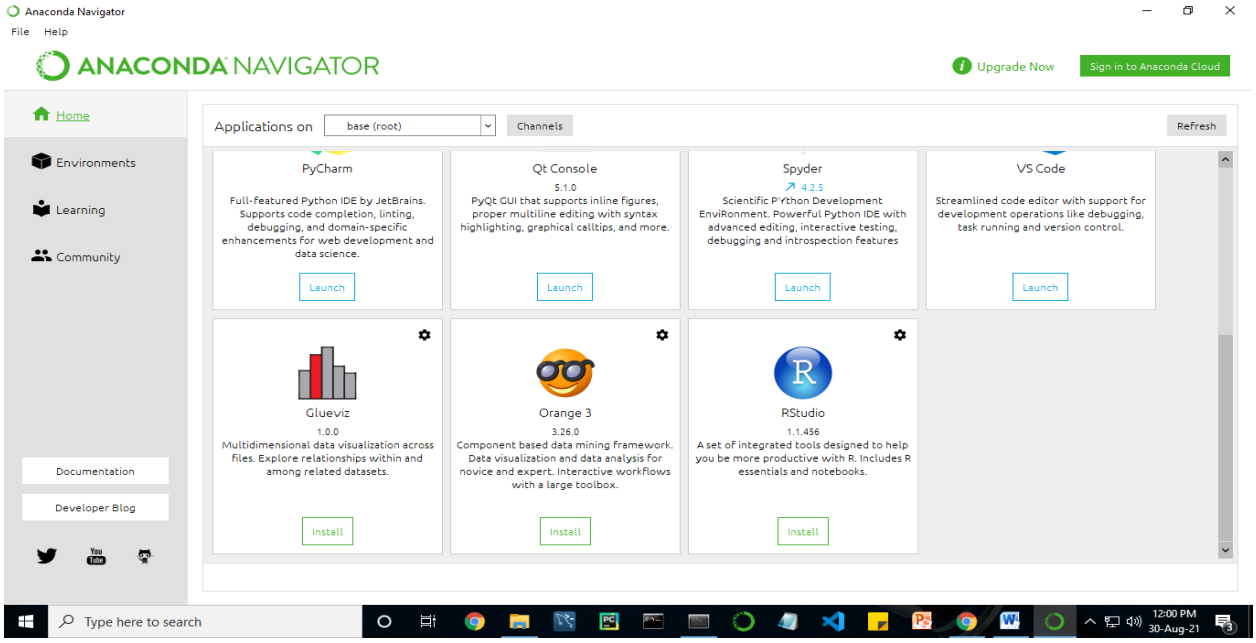
Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.





Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

Anaconda comes with many built-in packages that you can easily find with conda list on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

6.3.2 JUPYTER NOTEBOOK:

This website acts as "meta" documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

- Launch the jupyter notebook app
- In the Notebook Dashboard navigate to find the notebook: clicking on its name will open it in a new browser tab.
- Click on the menu *Help -> User Interface Tour* for an overview of the Jupyter Notebook App user interface.
- You can run the notebook document step-by-step (one cell a time) by pressing *shift + enter*.
- You can run the whole notebook in a single step by clicking on the menu *Cell -> Run All*.
- To restart the kernel (i.e. the computational engine), click on the menu *Kernel -> Restart*. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc…).

Purpose: To support interactive data science and scientific computing across all programming languages.

File Extension: An IPYNB file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

Working Process:

- Download and install anaconda and get the most useful package for machine learning in Python.
- Load a dataset and understand its structure using statistical summaries and data visualization.
- Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

- It will force you to install and start the Python interpreter (at the very least).
- It will give you a bird's eye view of how to step through a small project.
- It will give you confidence, maybe to go on to your own small projects.

When you are applying machine learning to your own datasets, you are working on a project. A machine learning project may not be linear, but it has a number of well-known steps:

- Define Problem.
- Prepare Data.
- Evaluate Algorithms.
- Improve Results.
- Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

Here is an overview of what we are going to cover:

1. Installing the Python anaconda platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

CHAPTER 7

SOFTWARE TESTING

7.1 LEVELS OF TESTING:

The software, which has been developed, has to be tested to prove its validity. Testing is considered to be the least creative phase of the whole cycle of system design. In the real sense it is the phase, which helps to bring out the creativity of the other phases makes it shine.

1. White Box Testing
2. Black Box Testing
3. Unit Testing
4. Functional Testing
5. Performance Testing
6. Integration Testing
7. Validation Testing
8. System Testing
9. Output Testing
10. User Acceptance Testing

7.1.1.1. White Box Testing:

White box testing, also known as clear box testing, glass box testing, or structural testing, is a software testing technique that examines the internal structure and implementation of the software being tested. Unlike black box testing, which focuses on testing the functionality of the software without considering its internal workings, white box testing involves analysing and testing the internal code paths, control flows, and data structures of the software.

7.1.1.2. Black Box Testing:

Black box testing is a software testing technique in which the internal workings or implementation details of the software being tested are not known to the tester. Instead, the tester focuses solely on the external behaviour and functionality of the software, treating it as a "black box" where inputs are provided, and outputs are observed without knowledge of the internal code structure.

7.1.1.3. Unit Testing:

Unit Testing involves testing individual units or components of the software to ensure they function correctly in isolation. A unit is the smallest testable part of any software, typically a function, method, or procedure.

7.1.1.4. Functional Testing:

Functional testing is a type of software testing that evaluates the functionality of a software application by testing its features and functionalities against specified requirements. The main objective of functional testing is to verify that the software behaves as expected and meets the functional requirements defined for it.

7.1.1.5. Performance Testing:

Performance testing evaluates how the system performs under various conditions, such as load testing to measure its response time and scalability, stress testing to determine its limits, and endurance testing to assess its stability over time.

7.1.1.6. Integration Testing:

Integration testing focuses on verifying the interaction between different modules or components of the software and ensuring they work together as expected. The purpose of integration testing is to verify that the interactions between these components work correctly and that they integrate seamlessly to perform the intended functionality.

7.1.1.7. Validation Testing:

Validation testing is a type of software testing that verifies whether a software application meets the requirements and expectations of its intended users and stakeholders. It is typically performed after the completion of development and before the software is released to the end-users or deployed into production.

7.1.1.8. System Testing:

System testing evaluates the complete, integrated system to verify that it meets specified requirements and functions correctly in its intended environment. It involves retesting the existing functionality of the software to verify that it still works as expected after modifications.

7.1.1.9. Output Testing:

"Output testing" typically refers to a type of testing where the output generated by a software system is examined to ensure it meets specified requirements or expectations. This type of testing is particularly relevant for systems that produce various forms of output, such as reports, documents, files, or data.

7.1.1.10. User Acceptance Testing: user acceptance testing (UAT), this type of testing involves validating the software against business requirements and ensuring it meets the needs of end-users.

CHAPTER 8

IMPLEMENTATION AND RESULT

8.1 IMPLEMENTATION

ANACONDA

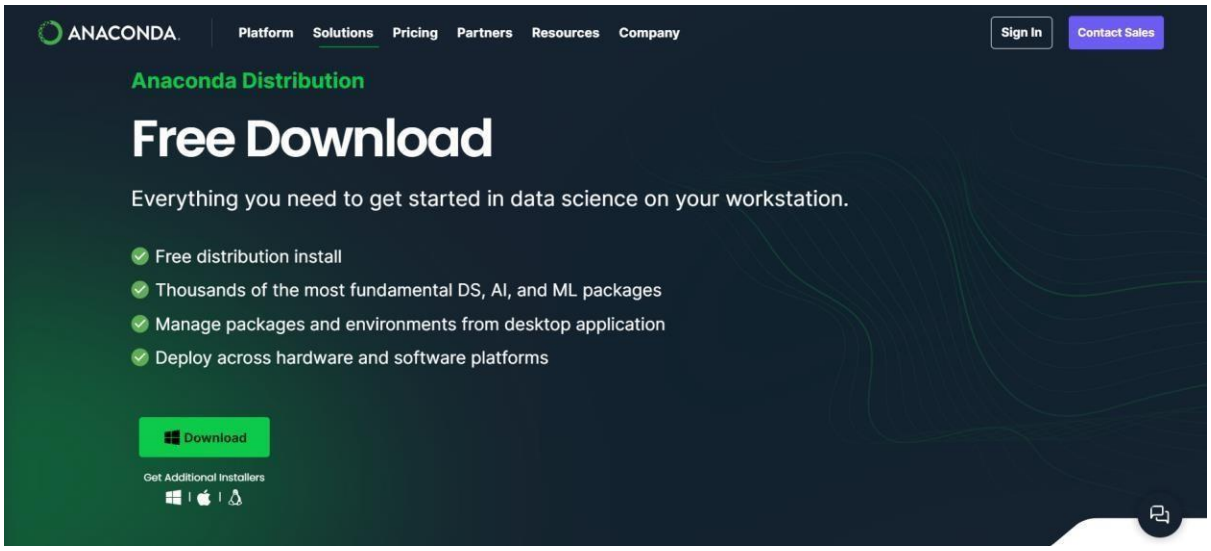


FIGURE 8.1:Install Anaconda

Open Source

Access the open-source software you need for projects in any field, from data visualization to robotics.

User-friendly

With our intuitive platform, you can easily search and install packages and create, load, and switch between environments.

Trusted

Our securely hosted packages and artifacts are methodically tested and regularly updated.

Anaconda Repository

Our repository features over 8,000 open-source data science and machine learning packages, Anaconda-built and compiled for all major operating systems and architectures.

Conda

Conda is an open-source package and environment management system that runs on Windows, macOS, and Linux. Conda quickly installs, runs, and updates packages and their dependencies. It also easily creates, saves, loads, and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language.

Anaconda Navigator

Our desktop application lets you easily manage integrated applications, packages, and environments without using the command line.

Cloud Environment Backup

Connect Anaconda Navigator to our community portal, Anaconda Cloud, to securely store your local environments in the cloud.

FIGURE 8.2: Anaconda prompt

Anaconda Prompt is a command-line interface (CLI) that comes with the Anaconda distribution of Python. It provides a way to interact with Anaconda and its associated tools and libraries. Anaconda software helps you create an environment for many different versions of Python and package versions. Anaconda is also used to install, remove, and upgrade packages in your project environments. Furthermore, you may use Anaconda to deploy any required project with a few mouse clicks.

8.2 SAMPLECODE

Module 1:

```
import warnings

warnings.filterwarnings('ignore')


import os

import glob

import numpy as np


from tensorflow.keras.preprocessing.image import ImageDataGenerator


from tensorflow.keras.models import Sequential

from PIL import Image


from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import MaxPooling2D
```

```
from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Activation



from keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt
ALMOND = 'DATASET/TRAIN/almond'

CHERRY = 'DATASET/TRAIN/Cherry'

JUTE = 'DATASET/TRAIN/jute'

SOYABEAN = 'DATASET/TRAIN/soyabean'


def plot_images(item_dir, n=6):

    all_item_dir = os.listdir(item_dir)

    item_files = [os.path.join(item_dir, file) for file in all_item_dir][:n]



    plt.figure(figsize=(80, 40))

    for idx, img_path in enumerate(item_files):

        plt.subplot(3, n, idx+1)

        img = plt.imread(img_path)

        plt.imshow(img, cmap='gray')

        plt.axis('off')



    plt.tight_layout()



def image_details_print(data,path):
```

```
    print('======== Images in: ', path)
    for key,values in data.items():
        print(key,':\t', values)



def images_details(path):
    files=[f for f in glob.glob(path + "**/*.*", recursive=True)]
    data={}
    data['Images_count']=len(files)
    data['Min_width']=10**100
    data['Max_width']=0
    data['Min_height']=10**100
    data['Max_height']=0


    for f in files:
        img=Image.open(f)
        width,height=img.size
        data['Min_width']=min(width,data['Min_width'])
        data['Max_width']=max(width, data['Max_width'])
        data['Min_height']=min(height, data['Min_height'])
        data['Max_height']=max(height, data['Max_height'])


    image_details_print(data,path)
print("")
```

```
print("TRAINING DATA FOR ALMOND:")

print("")

images_details(ALMOND)

print("")

plot_images(ALMOND, 10)
print("")

print("TRAINING DATA FOR CHERRY:")

print("")

images_details(CHERRY)

print("")

plot_images(CHERRY, 10)
print("")

print("TRAINING DATA FOR JUTE:")

print("")

images_details(JUTE)

print("")

plot_images(JUTE, 10)
```

:

```
print("")

print("TRAINING DATA FOR SOYABEAN:")

print("")

images_details(SOYABEAN)

print("")

plot_images(SOYABEAN, 10)
```

```
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,hor
izontal_flip=True)

training_set=train_datagen.flow_from_directory('dataset/train',target_size=(224,22
4),batch_size=32,class_mode='categorical')
test_datagen=ImageDataGenerator(rescale=1./255)

test_set=test_datagen.flow_from_directory('dataset/test',target_size=(224,224),bat
ch_size=32,class_mode='categorical')
Classifier=Sequential()

Classifier.add(Convolution2D(32,(3,3),input_shape=(224,224,3),activation='relu'))

Classifier.add(MaxPooling2D(pool_size=(2,2)))

Classifier.add(Flatten())

Classifier.add(Dense(38, activation='relu'))

Classifier.add(Dense(20, activation='softmax'))

Classifier.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['a
ccuracy'])
model_path = "MANUAL.h5"



callbacks = [

    ModelCheckpoint(model_path, monitor='accuracy', verbose=1, save_best_only=True)

]
epochs = 10

batch_size = 512



#### Fitting the model

history = Classifier.fit(

           training_set, steps_per_epoch=training_set.samples // batch_size,

           epochs=epochs,

           validation_data=test_set,validation_steps=test_set.samples  batch_size,
```

```
            callbacks=callbacks)
import matplotlib.pyplot as plt


def graph():
    #Plot training & validation accuracy values
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()


graph()


import matplotlib.pyplot as plt
def graph():
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
graph()
```

Module 2:

AlexNet Architecture

```
import warnings

warnings.filterwarnings('ignore')
import tensorflow

import tensorflow as tf

print(tf.__version__)



import keras

import keras.backend as K

from keras.models import Model

from keras.layers import Input, Dense, Conv2D, Conv3D, DepthwiseConv2D, SeparableConv2D, Conv3DTranspose

from keras.layers import Flatten, MaxPool2D, AvgPool2D, GlobalAvgPool2D, UpSampling2D, BatchNormalization

from keras.layers import Concatenate, Add, Dropout, ReLU, Lambda, Activation, LeakyReLU, PReLU



from IPython.display import SVG

from keras.utils.vis_utils import model_to_dot



from time import time

import numpy as np



from keras.callbacks import ModelCheckpoint
```

```
from tensorflow.keras.callbacks import EarlyStopping



import warnings

warnings.filterwarnings('ignore')
from tensorflow.keras.preprocessing.image import ImageDataGenerator



train=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_
flip=True,validation_split = 0.2)

train_data=train.flow_from_directory(directory
='DATASET/TRAIN',target_size=(224,224),

                                     batch_size=32,class_mode='categorical')
test=ImageDataGenerator(rescale=1./255)

test_data=test.flow_from_directory(directory
='DATASET/TEST',target_size=(224,224),

                                   batch_size=32,class_mode='categorical')
def alexnet(input_shape, n_classes):

  input = Input(input_shape)



  # actually batch normalization didn't exist back then

  # they used LRN (Local Response Normalization) for regularization

  x = Conv2D(96, 11, strides=4, padding='same', activation='relu')(input)

  x = BatchNormalization()(x)

  x = MaxPool2D(3, strides=2)(x)



  x = Conv2D(256, 5, padding='same', activation='relu')(x)

  x = BatchNormalization()(x)

  x = MaxPool2D(3, strides=2)(x)
```

```
  x = Conv2D(384, 3, strides=1, padding='same', activation='relu')(x)


  x = Conv2D(384, 3, strides=1, padding='same', activation='relu')(x)


  x = Conv2D(256, 3, strides=1, padding='same', activation='relu')(x)
  x = BatchNormalization()(x)
  x = MaxPool2D(3, strides=2)(x)


  x = Flatten()(x)
  x = Dense(4096, activation='relu')(x)
  x = Dense(4096, activation='relu')(x)


  output = Dense(n_classes, activation='softmax')(x)


  model = Model(input, output)

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'
,tensorflow.keras.metrics.Precision()])
  return model


input_shape = 224, 224, 3
n_classes = 20


K.clear_session()
```

```
model = alexnet(input_shape, n_classes)

model.summary()
model_path = "ALEXNET.h5"



from keras.callbacks import ModelCheckpoint



M = ModelCheckpoint(model_path, monitor='accuracy', verbose=1, save_best_only=True)
epochs = 100

batch_size = 64
#### Fitting the model

history = model.fit(

           train_data, steps_per_epoch=train_data.samples batch_size,

           epochs=epochs,

           validation_data=test_data,validation_steps=test_data.samples
batch_size,

           callbacks=[M])
history.history.keys()
import matplotlib.pyplot as plt

import numpy as np



plt.figure(figsize=(20, 8))

plt.plot(history.history['accuracy'])



for i in range(epochs):

    if i%5 == 0:


plt.annotate(np.round(history.history['accuracy'][i]*100,2),xy=(i,history.history[
'accuracy'][i]))
```

```
plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.show()
plt.figure(figsize=(20, 8))

plt.plot(history.history['loss'])



for i in range(epochs):

    if i%5 == 0:


plt.annotate(np.round(history.history['loss'][i]*100,2),xy=(i,history.history['los
s'][i]))

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.show()
```

Module 3:

RESNET Architecture

```
import warnings

warnings.filterwarnings('ignore')
import tensorflow

import tensorflow as tf

print(tf.__version__)
```

```
import keras

import keras.backend as K

from keras.models import Model

from keras.layers import Input, Dense, Conv2D, Conv3D, DepthwiseConv2D, SeparableConv2D, Conv3DTranspose

from keras.layers import Flatten, MaxPool2D, AvgPool2D, GlobalAvgPool2D, UpSampling2D, BatchNormalization

from keras.layers import Concatenate, Add, Dropout, ReLU, Lambda, Activation, LeakyReLU, PReLU



from IPython.display import SVG

from keras.utils.vis_utils import model_to_dot



from time import time

import numpy as np



from keras.callbacks import ModelCheckpoint

from tensorflow.keras.callbacks import EarlyStopping



import warnings

warnings.filterwarnings('ignore')
from tensorflow.keras.preprocessing.image import ImageDataGenerator



train=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,validation_split = 0.2)
```

```
train_data=train.flow_from_directory(directory
'DATASET/TRAIN',target_size=(224,224),

                                        batch_size=32,class_mode='categorical')
test=ImageDataGenerator(rescale=1./255)

test_data=test.flow_from_directory(directory 'DATASET/TEST',target_size=(224,224),

                                      batch_size=32,class_mode='categorical')
def resnet(input_shape, n_classes):



  def conv_bn_rl(x, f, k=1, s=1, p='same'):

    x = Conv2D(f, k, strides=s, padding=p)(x)

    x = BatchNormalization()(x)

    x = ReLU()(x)

    return x

  def identity_block(tensor, f):

    x = conv_bn_rl(tensor, f)

    x = conv_bn_rl(x, f, 3)

    x = Conv2D(4*f, 1)(x)

    x = BatchNormalization()(x)

    x = Add()([x, tensor])

    output = ReLU()(x)

    return output

  def conv_block(tensor, f, s):

    x = conv_bn_rl(tensor, f)

    x = conv_bn_rl(x, f, 3, s)

    x = Conv2D(4*f, 1)(x)
```

```
  x = BatchNormalization()(x)

  shortcut = Conv2D(4*f, 1, strides=s)(tensor)
  shortcut = BatchNormalization()(shortcut)

  x = Add()([x, shortcut])
  output = ReLU()(x)
  return output
def resnet_block(x, f, r, s=2):
  x = conv_block(x, f, s)
  for _ in range(r-1):
    x = identity_block(x, f)
  return x
input = Input(input_shape)

x = conv_bn_rl(input, 64, 7, 2)
x = MaxPool2D(3, strides=2, padding='same')(x)

x = resnet_block(x, 64, 3, 1)
x = resnet_block(x, 128, 4)
x = resnet_block(x, 256, 6)
x = resnet_block(x, 512, 3)

x = GlobalAvgPool2D()(x)
```

```
  output = Dense(n_classes, activation='softmax')(x)


  model = Model(input, output)


model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'
,tensorflow.keras.metrics.Precision()])

  return model

input_shape = 224, 224, 3

n_classes = 20



K.clear_session()

model = resnet(input_shape, n_classes)

model.summary()
model_path = "MOBILENET.h5"



from keras.callbacks import ModelCheckpoint



M = ModelCheckpoint(model_path, monitor='accuracy', verbose=1, save_best_only=True)
epochs = 100

batch_size = 512
#### Fitting the model

history = model.fit(

           train_data, steps_per_epoch=train_data.samples // batch_size,

           epochs=epochs,

           validation_data=test_data,validation_steps=test_data.samples
batch_size,
```

```
            callbacks=[M])
history.history.keys()
import matplotlib.pyplot as plt

import numpy as np



plt.figure(figsize=(20, 8))

plt.plot(history.history['accuracy'])

for i in range(epochs):

    if i%5 == 0:


plt.annotate(np.round(history.history['accuracy'][i]*100,2),xy=(i,history.history[
'accuracy'][i]))

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.show()
plt.figure(figsize=(20, 8))

plt.plot(history.history['loss'])

for i in range(epochs):

   if i%5 == 0:


plt.annotate(np.round(history.history['loss'][i]*100,2),xy=(i,history.history['los
s'][i]))

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.show()
```

8.3 SAMPLE SCREENSHOT:

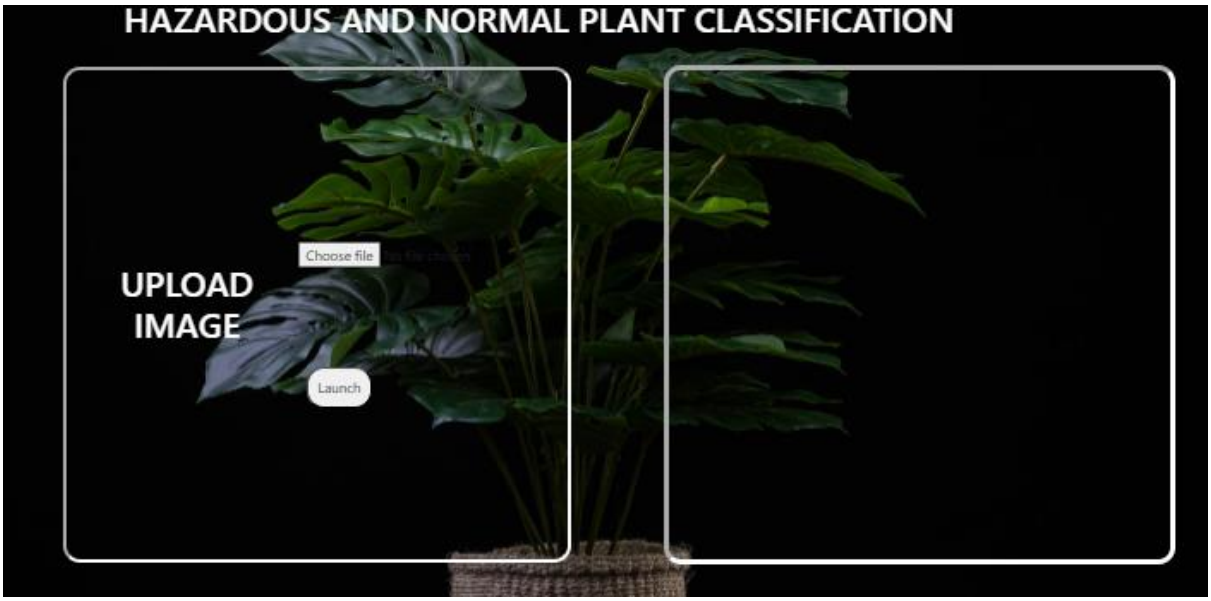


Fig 8.3 Samples screenshot

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENT

9.1 CONCLUSION

It focused on images from given dataset to predict the pattern of hazardous and normal plant using CNN model. This brings some of the following insights about plant prediction. The major benefits of the CNN classification, deployment can classify the images automatically.

9.2 FUTURE WORK

- We can deploy the model in any cloud-based system.
- We can implement more than three architectures
- We can connect this model to the hardware

CHAPTER 10

REFERENCES

10.REFERENCES

1.Zhang, L., Yu, S., & Huang, W. (2019). Machine learning-based classification of hazardous and normal plants using multispectral imagery. IEEE Access, 7, 156052-156061.

2.Mehmood, F., & Lee, Y. S. (2020). A comparative study of machine learning algorithms for classification of hazardous and normal plants. Sensors, 20(22), 6546.

3.Chen, Y., Tian, H., & Cheng, H. (2018). Classification of hazardous and normal plants using deep learning techniques. Journal of Hazardous Materials, 354, 30-38.

4.Li, W., Liu, C., & Li, Y. (2017). A novel approach to classify hazardous and normal plants based on sensor data fusion. Sensors and Actuators B: Chemical, 251, 826-834.

5.Yan, J., Wu, S., & Zhou, J. (2016). Decision fusion-based classification of hazardous and normal plants using wireless sensor networks. Journal of Network and Computer Applications, 71, 73-82.

6.Gomes, J., & Ramos, H. (2019). Hazardous and normal plant classification using convolutional neural networks with transfer learning. Expert Systems with Applications, 128, 254-265.

7. Wang, Z., Jiang, Y., & Zhang, Y. (2018). Classification of hazardous and normal plants based on ensemble learning Chemometrics and Intelligent Laboratory Systems, 182, 68-76.

8. Khan, M. A., Kim, Y., & Lee, J. (2017). Hazardous and normal plant classification using support vector machines with feature selection. Applied Sciences, 7(1), 52.

9. Kim, J., Kim, K., & Kim, H. (2016). Comparative analysis of hazardous and normal plant classification using various feature extraction techniques. International Journal of Control, Automation and Systems, 14(5), 1296-1305.

10. Wu, Q., & Cheng, H. (2015). Classification of hazardous and normal plants based on principal component analysis and k-nearest neighbor algorithm. International Journal of Distributed Sensor Networks, 11(4), 283657.