Plant Species Classification Using Deep Learning

MASTER OF COMPUTER APPLICATIONS

In

COMPUTER SCIENCE AND ENGINEERING

By

Mr. Kalyan D

(21001F0035)

Under the guidance of

Prof. B. ESWARA REDDY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

COLLEGE OF ENGINEERING

(Autonomous)

ANANTAPURAMU-515002

ANDHRA PRADESH

2021-2023

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

COLLEGE OF ENGINEERING (*Autonomous)*

ANANTHAPURAMU-515002

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project entitled, "**PLANT SPECIES CLASSIFICATION USING DEEP LEARNING**", is Bonafide work of **KALYAN D** bearing Admission No: **21001F0035** submitted to the faculty of Computer Science & Engineering, in partial fulfillment of the requirements for the award of **MASTER OF COMPUTER APPLICATIONS** from Jawaharlal Nehru Technological University Anantapur College of Engineering (Autonomous), Ananthapuramu.

Signature of the Supervisor	Signature of the Head of the Department
Prof. B. Eswara Reddy	**Dr. K. F. Bharati** **M.Tech, Ph.D.**
Professor	Associate Professor & HOD
Dept. of Computer Science & Engineering.	Dept. of Computer Science & Engineering.
J.N.T.U.A. College of Engineering	J.N.T.U.A. College of Engineering
Ananthapuramu-515002	Ananthapuramu-515002

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

COLLEGE OF ENGINEERING (*Autonomous)*

ANANTHAPURAMU-515002

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

I, **KALYAN D**, bearing Admission No. **21001F0035**, hereby declare that the project report entitled "**PLANT SPECIES CLASSIFICATION USING DEEP LEARNING**" under the guidance of **Prof. B. ESWARA REDDY**, JNTUA College of Engineering (Autonomous), Ananthapuramu is submitted inpartial fulfillment of the requirements for the award of the degree of **MASTER OF COMPUTER APPLICATIONS** in Computer Science and Engineering.

This is a record of bonafide work carried out by me and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

KALYAN D

(21001F0035)

ACKNOWLEDGEMENT

The satisfaction and exhilaration that accompany the successful completion of any would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that I now have the opportunity to express my gratitude to all of them.

The first person I would like to thank is my guide, **Dr. B. Eswara Reddy**, Professor of CSE. His wide knowledge and logical way of thinking have made a deep impression on me. His understanding, encouragement, and personalguidance have provided the basis for this thesis. He is a source of inspiration for innovative ideas and his support is known to all his students and colleagues.

My Special thanks to **Dr. K. F. Bharati**, Head of the department, Dept. of Computer Science and Engineering, JNTUA College of Engineering (Autonomous), Ananthapuramu. Her wide support, knowledge, and enthusiastic encouragement have impressed me to be better involved in my project thesis and technical design also her ethical morals helped me to develop my personal and technical skills to deploy my project in success.

I wish to thank **Prof. E. Arunakanthi**, Vice Principal of JNTUA College of Engineering (Autonomous), Ananthapuramu who has extended her support for the success of this project.

I wish to thank **Prof. S.V. Satyanarayana**, Principal, JNTUA College of Engineering (Autonomous), Anantapuramu, for providing an excellent institution environment and all facilities in completing this project.

I wish to thank the teaching staff and non-teaching staff of the Computer Science and Engineering Department, JNTUA College of Engineering (Autonomous), Anantapuramu, for their support to complete my project successfully.

KALYAN D

(21001F0035)

ABSTRACT

The Deep Learning for Plant Species Classification project aims to develop a highly accurate model for identifying different species of plants using deep learning techniques. With the increasing concern for environmental conservation and sustainable agriculture, it is essential to have accurate and efficient methods for plant species identification.

This project will utilize a dataset of plant images and employ convolutional neural networks (CNNs) for feature extraction and classification. Transfer learning techniques will be used to fine-tune pre-trained models to achieve high accuracy with limited training data. Various architectures such as ResNet, Inception, and VGG have been experimented with to find the best model for the task. This approach investigates the impact of different data augmentation techniques such as rotation, scaling, and cropping on the model's performance. The trained model has been tested on a separate validation dataset to evaluate its accuracy and efficiency.

The potential applications of the proposed work include developing plant identification, improving crop management, and aiding in plant conservation efforts. The primary outcome will contribute to the development of accurate and efficient methods for plant species classification and have significant implications for the fields of agriculture and ecology.

Keywords: plant species classification, feature selection, feature extraction, CNN, deep learning.

CONTENTS

Acknowledgment	iv
Abstract	v
Content	vi
Chapters	**Page No.**
Chapter-1. Introduction	**1**
1.1 Introduction	1
1.2 Existing System	2
1.3 Proposed System	3
Chapter-2. System Analysis	**5**
2.1 Architecture Diagram	5
2.2 Methodology and Algorithms	6
2.3 Feasibility Study	9
2.4 Software Development Life Cycle	10
Chapter-3. Software Requirement Specifications	**12**
3.1 Functional Requirements & Non-Functional Requirements	12
3.2 System Specifications	13
Chapter-4. System Design	**14**
4.1 System Design	14
4.2 Modules	15
Chapter-5. UML Diagrams	**16**
5.1 Goals	16
5.2 Use Case Diagram	16
5.3 Class Diagram	17
5.4 Sequence Diagram	18
5.5 Collaboration Diagram	19

5.6 Deployment Diagram 19

5.7 Activity Diagram 20

5.8 Component Diagram 21

5.9 ER Diagram 21

5.10 DFD Diagram 22

Chapter-6. System Testing 23

6.1 Testing Activities 23

6.2 Unit Testing 25

6.3 Integration Testing 25

6.4 Acceptance Testing 25

Chapter-7. Implementation 27

Chapter-8. Output Result 33

Chapter-9. Conclusion & Future Scope 39

Chapter-10. Reference 40

CHAPTER 1

INTRODUCTION

Preserving biodiversity is imperative, and a deeper understanding of the nuances in identifying plant species through traditional artisanal characteristics is crucial. Memorizing specific botanical terminology can be challenging for the average person. The concept of automating plant species identification is becoming increasingly feasible. Machine learning and deep neural networks assume a pivotal role in this endeavor. Consequently, we are employing a convolutional neural network (CNN) rooted in deep learning to glean intricate features from leaf images, subsequently leveraging these features for precise plant species classification. Deep learning techniques consistently surpass any manual, artisanal approach in this context.

1.1 INTRODUCTION

Plants represent the foremost group of multicellular organisms, predominantly constituting photosynthetic eukaryotes within the plant kingdom. In the annals of biological classification, plants were historically categorized as one of two kingdoms encompassing all life forms apart from animals, with algae and fungi also falling under this umbrella. Nevertheless, the modern definition of the plant kingdom now specifically excludes not only prokaryotes like archaea and bacteria but also fungi and certain types of algae. According to a refined classification, plants collectively form the Viridiplantae clade, denoting "green plants" in Latin. Within this clade, you'll find an array of organisms, including flowering plants, conifers, gymnosperms, pteridophytes and their kin, tomentosum, liverworts, mosses, and various green algae. However, this classification omits red algae as well as brown algae from the plant kingdom.

Green plants predominantly harness solar energy through photosynthesis, a process facilitated by primary chloroplasts originating from endosymbiotic relationships with cyanobacteria. These chloroplasts house chlorophyll a and b, bestowing upon them their distinctive green hue. While many plants thrive independently, some adopt parasitic or mycotrophic lifestyles, forfeiting the capacity for regular chlorophyll production and photosynthesis while still yielding flowers, fruits, and seeds.

The planet hosts an estimated 320,000 plant species, with the majority, roughly between 260,000 to 290,000, exhibiting the capacity for seed production. Green plants play a pivotal role by contributing significantly to the generation of molecular oxygen and forming the foundational basis for the majority of Earth's ecosystems. Plants that yield grains, fruits, and

vegetables hold a paramount position as essential sustenance for humanity, having been systematically cultivated for millennia. Beyond their nutritional significance, plants hold cultural and utilitarian value, finding applications in areas such as adornment through jewelry, construction materials, and the crafting of writing instruments. Moreover, the botanical kingdom has served as a wellspring for a diverse array of pharmaceuticals and psychotropic substances, underlining its profound impact on human health and well-being.

1.2 EXISTING METHOD

This model places significant emphasis on an established methodology originally crafted using various machine learning algorithms. The image acquisition process involved sowing seeds, and images were captured using a smartphone boasting a resolution of 1344 x 2240 pixels. To ensure precise image acquisition, a metal frame was meticulously erected at a height of 40 cm above the crop row to accommodate the image-capturing device.

The subsequent stages of image processing and feature extraction were meticulously executed through MATLAB programming. This system was put into practice to examine the characteristics of several plant species, namely peanuts, thorn apples, morning glories, purslanes, and velvetleaf plants.

To define the corner coordinates of the rectangular boundary around the plant, a straightforward marking process was employed within the **"imtool"** function. Subsequently, the **"imopen"** function was applied to execute morphological opening on either grayscale or binary images, utilizing a designated structuring element. For the extraction of geometric shape characteristics such as area, perimeter, major axis length, minor axis length, and equivalent diameter, the **"regionprops"** function was systematically employed on the binary images.

Incorporating color attributes, we delved into the RGB, HSI, and L*a*b color spaces. Notably, HSI and L*a*b were found to align more closely with human visual color perception, yielding superior outcomes in the final system. Consequently, RGB color images underwent a transformation into both HSI and L*a*b color spaces. Texture features were also considered, and for this purpose, the Gray Level Co-occurrence Matrix and Gray Level Run Length Matrix came into play. These matrices were calculated for four distinct directions at intervals of 0°, 45°, 90°, and 135°, with each comparison involving pixels situated at a one-pixel distance. This resulted in the computation of four GLCMs for every image, capturing intricate texture information.

Various feature selection methods, including correlation-based selection, Information Gain, Gain Ratio, and OneR, were employed to pinpoint the most pivotal feature set for effectively distinguishing between distinct plant types. The Correlation-based Feature Selection (CFS) technique excels in isolating a subset of features strongly aligned with class labels while minimizing their mutual relevance. Information Gain gauges the reduction in dataset feature entropy, serving as a valuable criterion for feature selection. Meanwhile, Gain Ratio, a refinement of Information Gain, mitigates bias by considering intrinsic information from each attribute and eliminating attribute-specific bias values. Collectively, these methods play a pivotal role in identifying the most influential features, essential for the discrimination of diverse plant types.

To classify the plant-type image they used two strategies were applied one using five classifiers among them is Multilayer perception by using this exit system used to train and split the dataset, k-Nearest Neighbors (KNN) algorithm is a non-parametric machine learning method that can be useful for plant species classification when the decision boundaries between different classes are complex or not well-defined. It can capture the local patterns in the feature space and make predictions based on the neighbors' class labels.

The existing system's essence lies in its assessment of two renowned boosting techniques, Adaboost. M1 and LogitBoost algorithms, aimed at bolstering plant classification performance across four distinct classifiers: Multi-layer Perceptron (MLP), k-Nearest Neighbors (KNN), Random Forest, and Support Vector Machine (SVM).This evaluation process encompassed the application of four diverse feature filtering methods, which included Correlation-based Feature Selection (CFS), Information Gain (IG), Gain Ratio (GR), and OneR. These techniques were meticulously applied to the extracted image features, spanning across five distinct datasets, to refine the plant classification process.

1.3 PROPOSED SYSTEM

In Our proposed approach delves into the classification of diverse plant species through the application of Convolutional Neural Networks (CNN) within the realm of deep learning. The imperative nature of biodiversity preservation underscores the necessity for a deeper understanding of plant species identification. Traditional methods reliant on hand-crafted features prove intricate in this context. What sets our CNN apart is its inherent capacity for both feature selection and feature extraction, simplifying and enhancing the plant species classification process.

In CNN we can take multiple datasets and large datasets at a time, and no need to upload data and train the data every time, the dataset was trained in the background and the plant species classification will calculate the accuracy output without using any data points. In deep learning neurons will train one image multiple times and then will classify and save it aside separately, so we can save time, it will increase the speed of the work and provides good accuracy.

In this proposed system we are using Python language to build the project its storage value is less, and Python has a prebuilt library so by using it we can save time and it is less lengthy code, time complexity will be reduced while image preprocessing and web browsing. Hence, we proposed our work, CNN uses four layers that assist in extracting information from an image, the layers are the convolution layer, ReLU layer, pooling layer, and fully connected layer, where we are mainly classifying the five types of plant species using CNN.

CHAPTER 2

SYSTEM ANALYSIS

System analysis is the process of looking at a system for development or troubleshooting needs. Information technology is an example of this, where computer-based systems need to have a specific analysis based on its structure and design.

2.1 ARCHITECTURE

Figure 2.1 depicts an architecture that details the responsibilities that the various objects play in the classification of plant species. The dataset, which gathers data for the categorization of plant species, is the primary element in this architecture. The following three components are the system, training, and testing, where the data is thoroughly analyzed and prepared for classification.

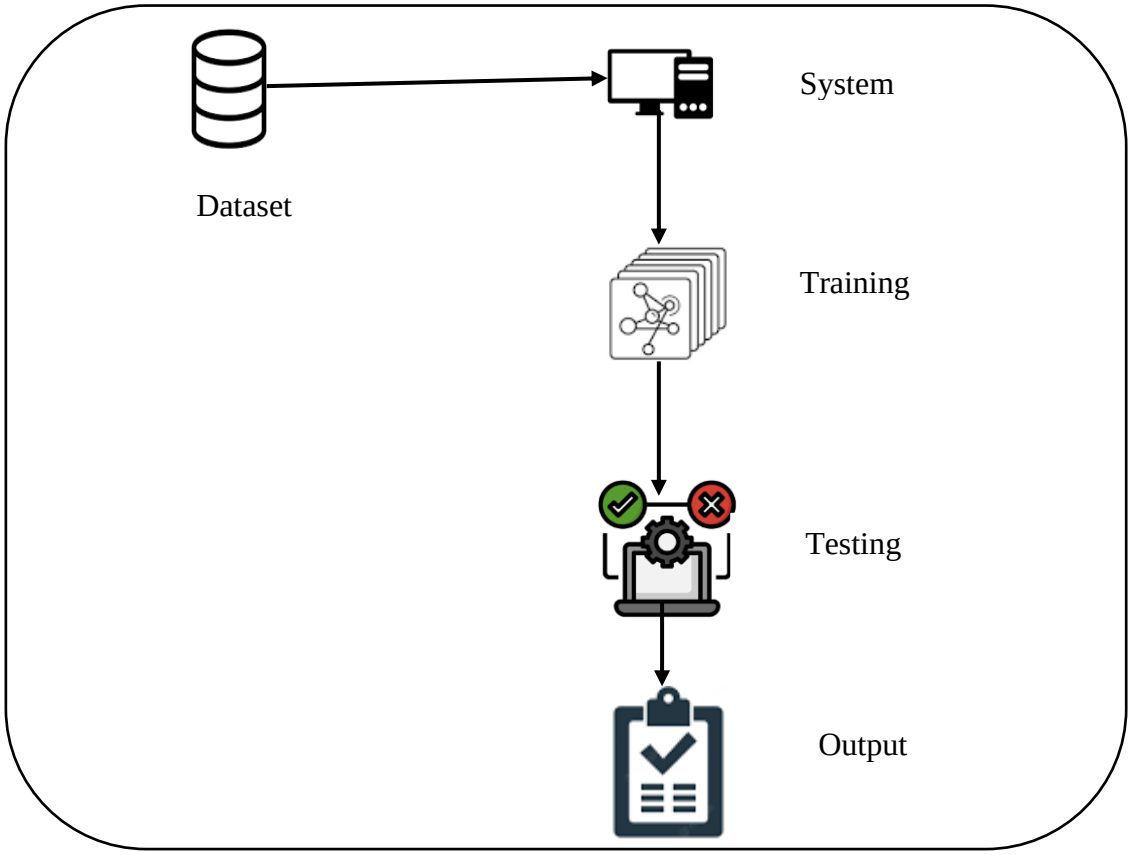

Figure 2.1 Architecture for Plant Species Classification

The roles of the participating objects are as follows:

a) **Dataset**

datasets were taken from Kaggle

b) System

Necessary pre-processing steps will be completed with the dataset before training with our algorithm.

c) Training

Once after training the model will be saved for testing and classifying.

d) Testing

Once the CNN model is trained, evaluate its performance using the testing dataset that was set aside earlier

e) Output/ Classification

User can upload the images of which to be classified and by using the saved model the images will be classified and predicted.

2.2 METHODOLOGY AND ALGORITHMS

Convolutional Neural Network

Step(1a): Convolutional Operation

The convolution operation serves as the initial cornerstone of our attack strategy. We discuss feature detectors that function as neural network filters in this section. Along with feature maps, we also talk about pattern recognition, detection layers, observational mapping, and learning the parameters of such maps. Figure 2.2 demonstrates the process for separating small pixels to obtain the feature map from the input image.

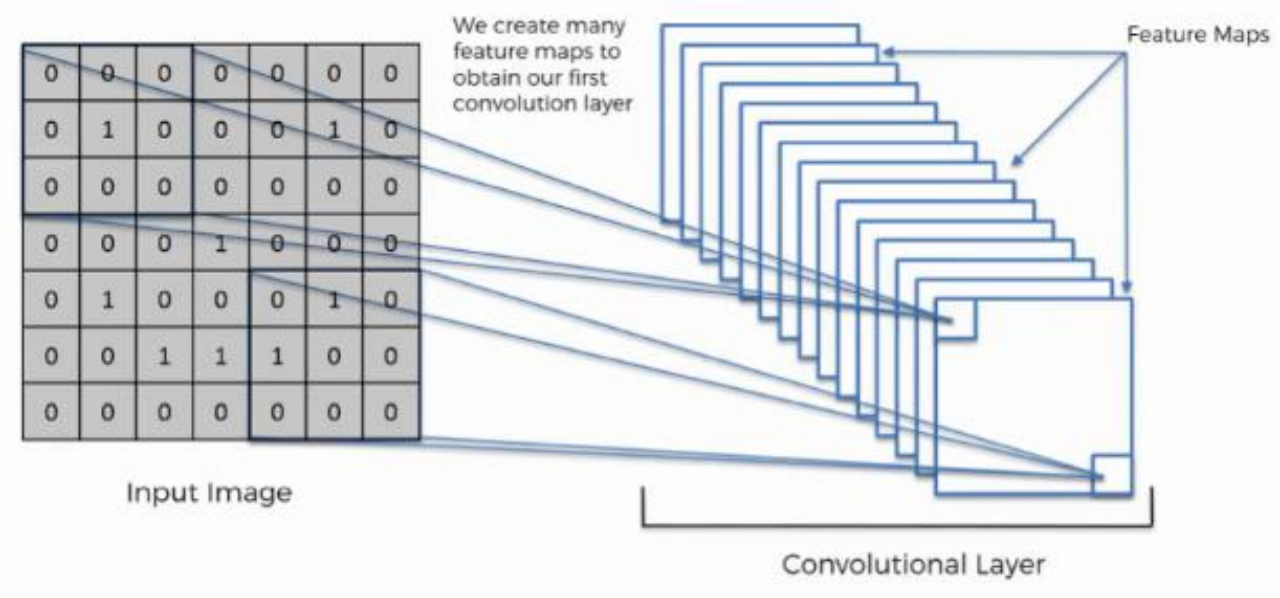

Figure 2.2 Convolutional Layer

Step (1b): ReLU Layer

The rectified or mended linear unit is used in the second half of this step. Rectified linear unit layers are discussed, and the concept of linearity is examined in the context of convolutional neural networks. Black-and-white photographs are arranged in a 2D array format, whereas color images are arranged in 3D array formats like Red, Green, and Blue, as seen in Figure 2.3.

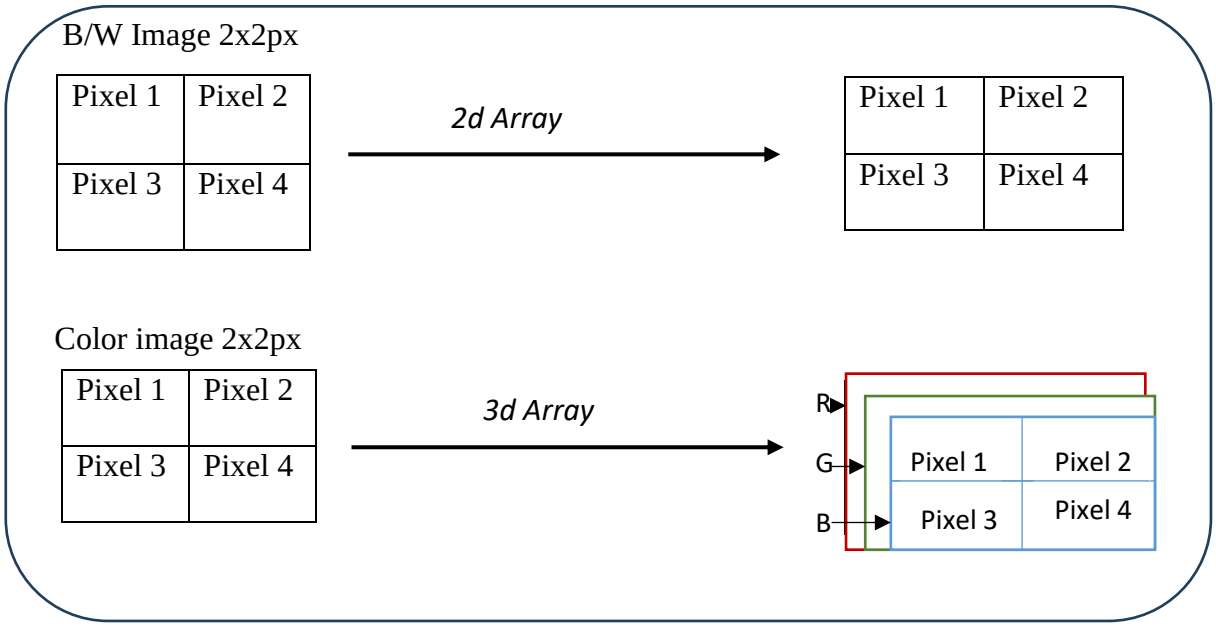

Figure 2.3 Convolutional Neural Networks Scan Images

Step 2: Max Pooling Layer

In this section, we'll talk about redundancy and look at how it typically works. We do have the strongest connection conceivable, though. However, we consider a number of approaches, including averaging (or summing). The presentation that uses a visual interactive tool to bring you through the full topic finishes this part. Figure 2.4 illustrates how to use the strides and size of the filter to extract the maximum value from the feature map.

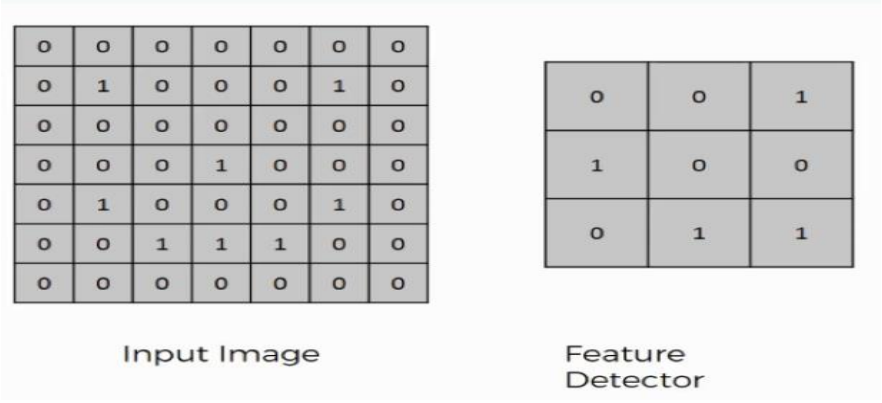

Figure 2.4 Max Polling Layer

Step 3: Flattening

Here is a brief explanation of the convolutional neural network smoothing procedure, showing how we move from connected layers to smoothed layers.

Step 4: Full Connection

This part provides a summary of everything we covered in the previous section. By learning this, you will gain a deeper comprehension of how convolutional neural networks work and how the resulting "neurons" learn to recognize images.

Let's end up the discussion by quickly summarizing the idea addressed in the chapter. You should read the accompanying tutorial on SoftMax and Cross-Entropy if you find this useful, which you probably will. Although it is not necessary for the course, becoming familiar with these ideas will be highly helpful because you will probably run into them when working with convolutional neural networks. Each neuron in the straightforward layer of neurons in Figure 2.5 receives information from every neuron in the previous layer

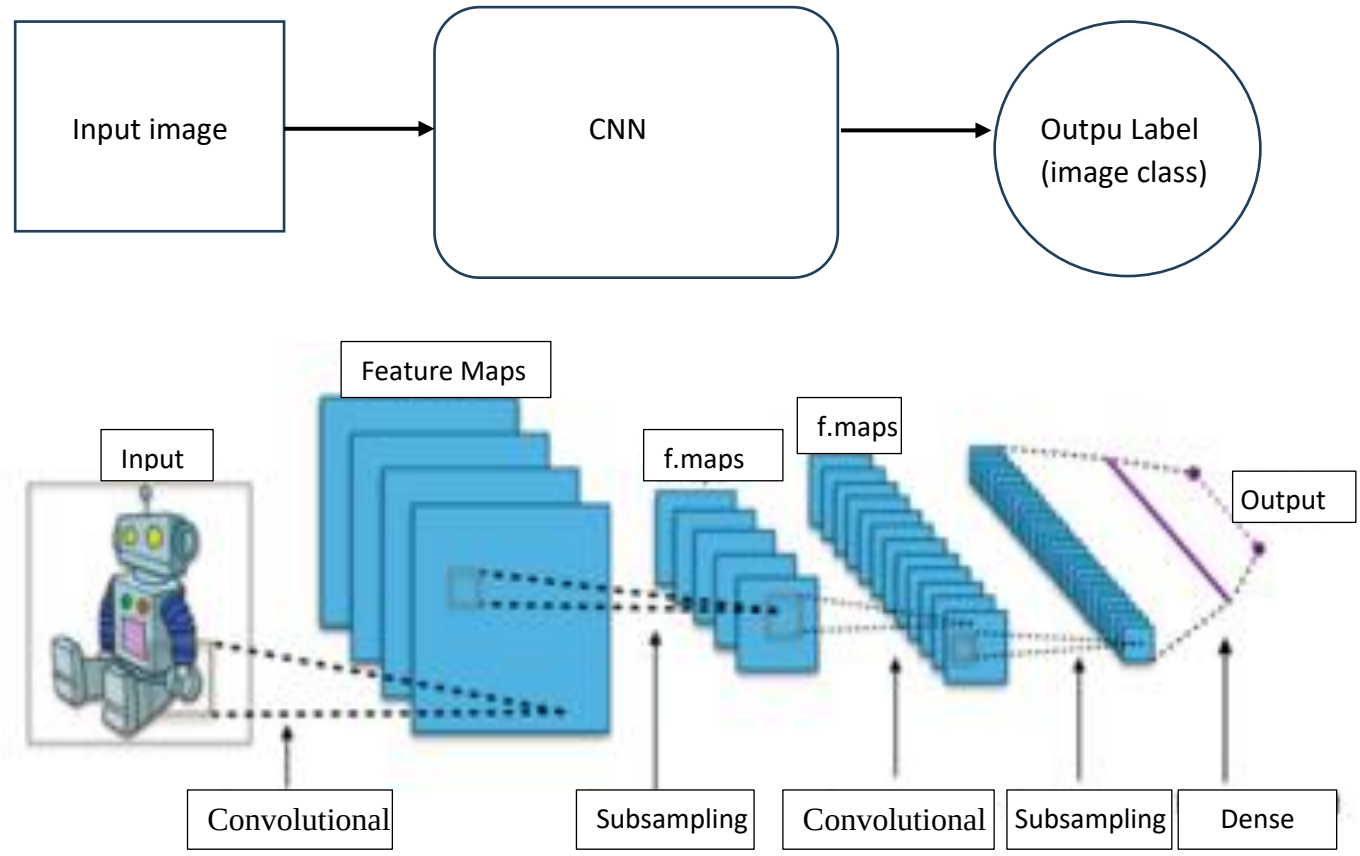

Figure 2.5 Full connected Layer or Dense layer

2.3 Feasibility Study

At this stage, the feasibility of the idea is assessed, and a business proposal is created with rudimentary cost and project plan estimates. As part of the system analysis, a feasibility analysis of the desired system must be performed. This will guarantee that the suggested approach won't put too much of a strain on the business. To carry out a feasibility study, it is crucial to know the fundamental specifications of the system.

The following three factors are crucial to the feasibility analysis:

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

2.3(a) Economic Feasibility:

This analysis aims to confirm how much money the system will cost the corporation. The company's system is being developed with the minimal resources that are now available. It's necessary to justify costs. Due to the fact that the majority of the technologies employed are publicly available, the development system is thus well within the budget. It was necessary to buy just customized items.

2.3(b) Technical Feasibility:

This study's objective is to confirm that the system's technical requirements—or requirements—can be met. The developed systems shouldn't put too much of a strain on the available technical resources. The number of technical resources required as a result is very significant. Due to this, the buyer makes excessive demands. As little or no adjustments are needed to deploy this system, the established system should only have minor demand.

2.3(c) Social Feasibility:

The research includes tracking the system's level of user acceptance. It entails instructing the user on how to use the system efficiently. The user must accept the system's necessity without feeling threatened by it. Only the techniques employed to inform and acquaint users with the system will determine the degree of user acceptance. Because he is the system's last user, his confidence must be increased so that he may offer helpful criticism as well.

2.4 Software Development Life Cycle

We use the waterfall model for our project's software development cycle due to its step-by-step implementation process.

- **Requirement Gathering and analysis** − At this point, specifications have been created and contain all potential needs for the system that will be developed.
- **System Design** − It involves reviewing the prior phase's requirements and creating the system design. This system design helps define the overall system architecture by identifying the hardware and system requirements.
- **Implementation** − The system is initially built up as a number of brief programs, or units, which are then merged using inputs from the system design. Each gadget is built and tested individually in a process known as unit testing. The waterfall model's process flow chart for the software development life cycle is shown in Figure 2.6.

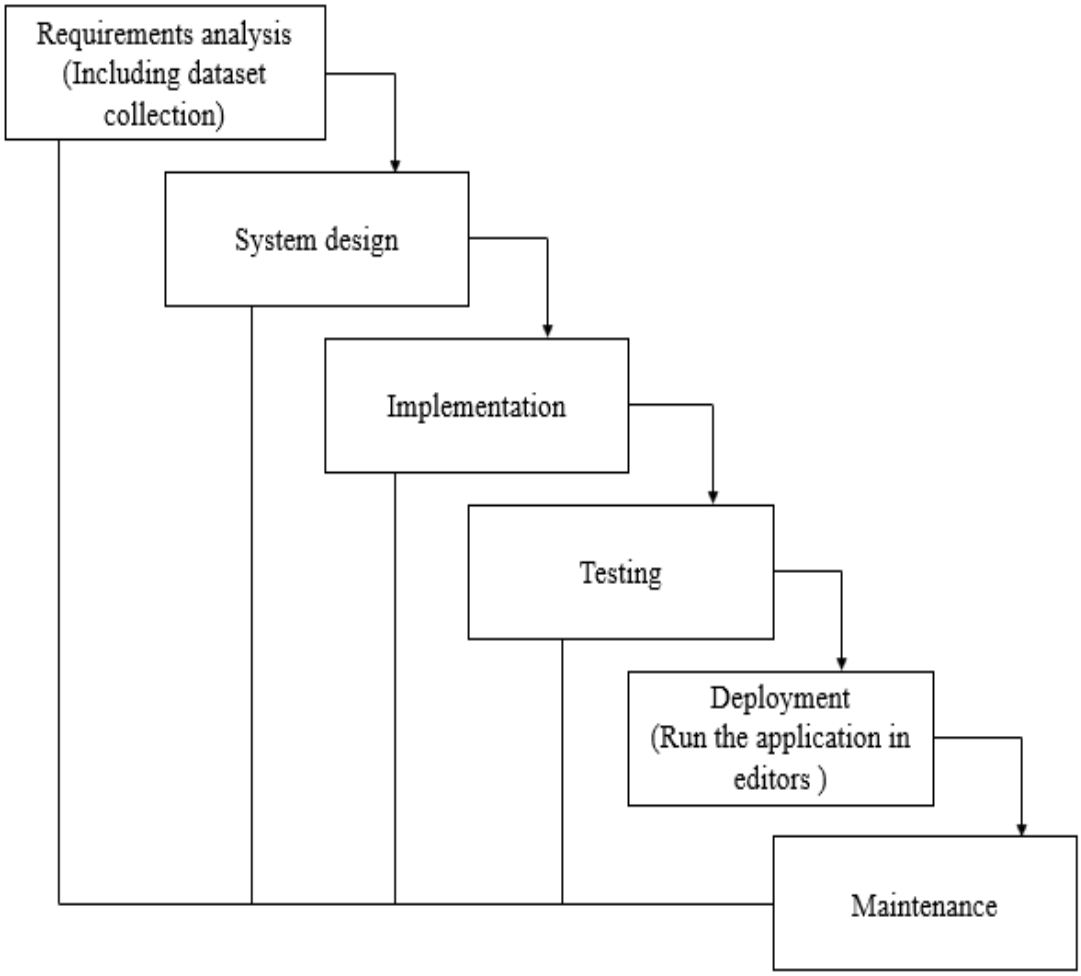

Figure 2.6 Waterfall Model

- **Integration and Testing** − During the implementation phase, each component is tested before being merged into the overall system. After integration, all system components are checked for any potential flaws or problems.
- **Deployment of system -** when the product is used in the customer environment, subjected to functional and non-functional testing, or when it is released onto the market.
- **Maintenance** – In the client environment, some issues can arise. To fix these problems, patches are released. To enhance the product, newer versions are also launched. To put these adjustments into effect in the client environment, maintenance is carried out.

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

A requirements specification is something like a software requirements specification (SRS). A software system is a comprehensive description of how the system being developed will behave. It includes a number of usage illustrations that detail each user interaction with the program. SRS also covers use cases and non-functional requirements. Non-functional specifications are those that limit the design or execution.

3.1 Functional Requirements & Non-Functional Requirements

An essential stage in establishing the viability of a system or software project is requirements analysis. The two primary types of requirements are functional and non-functional requirements.

3.1.1 Functional Requirements

These are the needs that the system must meet in order to perform the fundamental functions that the end user has specified. All of these features must unavoidably be incorporated into the system and the contract. They are portrayed or described as a system input, an action taken, and an anticipated outcome. These are user-defined specifications that, in contrast to non-functional specifications, are clearly visible in the finished product. Functional requirements examples:

1) User wanted to upload the image from various files.
2) User needs an About page to know the purpose of the project.
3) User needs the prediction of the image by comparing the trained dataset.

3.1.2 Non-functional requirements

The system must adhere to the quality standards outlined in the project contract. The importance or scope of these applications will vary from project to project.These criteria, which are also known as non-behavioral requirements, deal with things like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability

- Performance
- Reusability
- Flexibility

3.2 SYSTEM SPECIFICATIONS

Software Specifications:

- Server-side Script :Python
- IDE :Visual Studio Code
- Front end :HTML, CSS, JS
- Libraries Used :NumPy, IO, OS, Flask, TensorFlow(keras), Matplotlib, Pylab, PIL.

CHAPTER 4

SYSTEM DESIGN

The purpose of system design, also known as top-level design, is to choose the modules that should be a part of the system, their specifications, and the ways in which they interact to achieve the desired result. At the end of the system design phase, the most important system modules, file formats, and data structures are identified.

4.1 Design

4.1(a) Input Design

The raw data entered into an information system is processed to create the output. Developers must take computers into consideration while designing input methods. As a result, the system's output's quality is dependent upon the input's quality. The qualities listed below are present in well-designed input forms and screens.

- It must successfully accomplish particular tasks, such as storing, storing, and retrieving data.
- It ensures proper finishing and accuracy.
- It should be easy to complete and clear.
- User attention, consistency, and simplicity should be its main priorities.

Objectives for Input Design:

The input design goals are:

- Create procedures for entering data.
- To reduce input volume.
- To decrease the input volume.
- Develop novel methods for data collection or source materials for data collection.
- Develop screens for user interfaces, input datasets, and data entry, among other things.
- Conduct validity checks and offer effective input controls.

4.1(b) Output Design

In any system, product planning is the most crucial function. Developers choose the output categories they will use, taking into account the output controls and report layouts for prototypes.

Objectives of Output Design:

The objectives of input design are:

- Develop output designs that meet end-user requirements.
- Deliver appropriate production quantities.
- Format your output properly and send it to the right people.
- To make the output available at the right time so that we can make the right decisions.

4.2 MODULES

4.2(a) System

- Create Dataset

  The dataset, which consists of images of various plant species sorted into the five classes that need to be recognized, is split into training and testing datasets, with the test size ranging from 30% to 20%.

- Pre-processing
  To properly train our model, the photographs are scaled and altered.
- Training

  The pre-processed training dataset is utilized in order to train our model utilizing the CNN technique.

- Classification

  Our model generates a display of photos that are labeled according to their species.

4.2(b) User

- Upload Image

  The user must provide an image that needs to be categorized.

- View Results

  The viewer views the results for the categorized images.

CHAPTER 5

UML DIAGRAMS

Unified Modeling Language is known as UML. A standardized general-purpose modeling language known as UML is used in the creation of object-oriented software. The Object Management Group oversees and creates this standard.

The goal is for UML to become a widely used language for creating models of object-oriented computer programs. Metamodels and notations make up the two main parts of UML as it is right now. In the future, further techniques and procedures might be connected to or added to UML. A standard language known as UML is employed for business modeling, non-software systems, and describing, visualizing, producing, and documenting software system artifacts.

UML stands for a collection of engineering best practices that have been effective in simulating big, complicated systems. The object-oriented software development method and UML are both crucial components of these processes. To depict the design of software projects, UML typically employs graphical notation.

5.1 GOALS:

The Essential objectives within the plan of the UML are as takes after:

1. Give users a visual modeling language that is expressive and ready to use so they can create and share meaningful models.
2. Offer options for specialization and expansion of fundamental concepts.
3. Be unreliant on any one development methodology or programming language.
4. Provide a clear structure for understanding the modeling language.
5. Support the development of the OO tool market.
6. Backs advanced development ideas including teamwork, frameworks, patterns, and components.
7. Include top techniques.

5.2 USE CASE DIAGRAM

- A use case analysis defines and produces a particular kind of behavioral diagram known as a Unified Modeling Language (UML) use case diagram.

- Its objective is to convey a graphical summary of the functionality offered by the system in terms of actors, their goals (represented as use cases), and the interdependencies between these use cases.
- Use case diagrams' primary objective is to identify which system operations are carried out for which actors. The parts of the actors in your system can be expressed.

The user can add photographs, learn about the project, and view results thanks to the system's connections to upload dataset, data processing, training, and species categorization shown in Figure 5.1 of the diagram. These are thoroughly discussed.

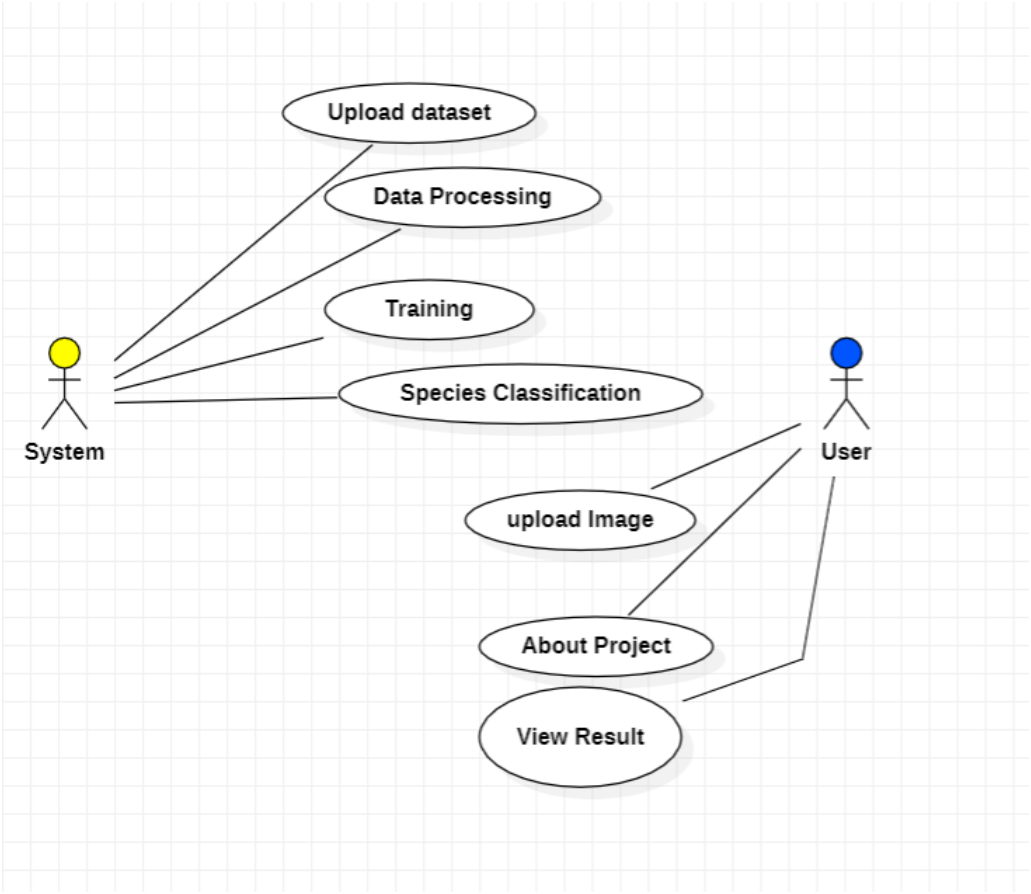

Figure 5.1 System & User Use Case Diagram

5.3 CLASS DIAGRAM

An example of a static structural diagram used in software development is the Unified Modeling Language (UML) class diagram, which displays the classes, attributes, operations, and connections between the classes to explain how a system is organized. Specifies the information's class. Figure 5.2 in this figure illustrates the link between two distinct classes, namely system and user, each with a unique property.

Figure 5.2 Class Diagram

5.4 SEQUENCE DIAGRAM

- A specific kind of interaction diagram that depicts how processes interact with one another and in what order is a Unified Modeling Language (UML) sequence diagram.
- The message sequence diagram is organized as follows. The terms event diagram, event scenario, and timing diagram are all variations of the term sequence diagram.

The relationship between the lifeline of the system, the user, and their communications is explained in this diagram in Figure 5.3.

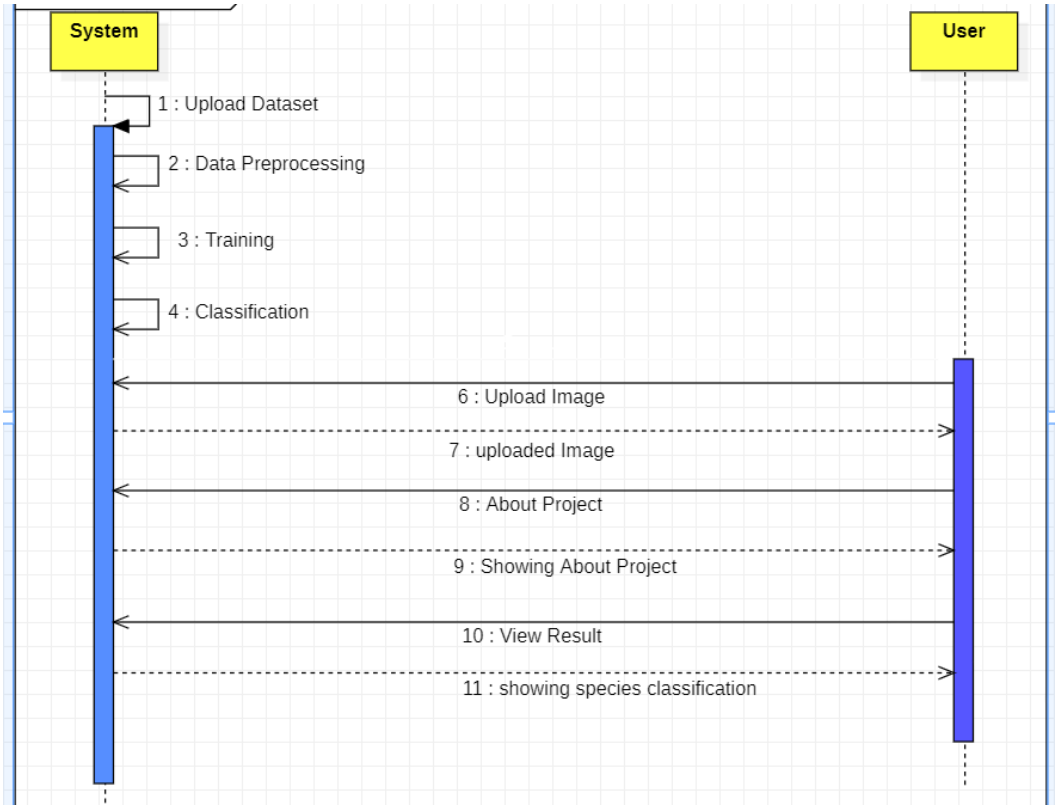

Figure 5.3 Sequence Diagram

5.5 COLLABORATION DIAGRAM:

A numbering scheme, like illustrated below, is used in collaboration diagrams to specify the order of method calls. The numbers show how the methods are called one after the other. The collaboration diagram was demonstrated using the same order management system. Sequence diagram calls are comparable to method calls. Sequence diagrams do not describe how things are organized, whereas collaboration diagrams display how objects are organized. Figure 5.4 of the schematic explains how two lifelines will communicate with one another.

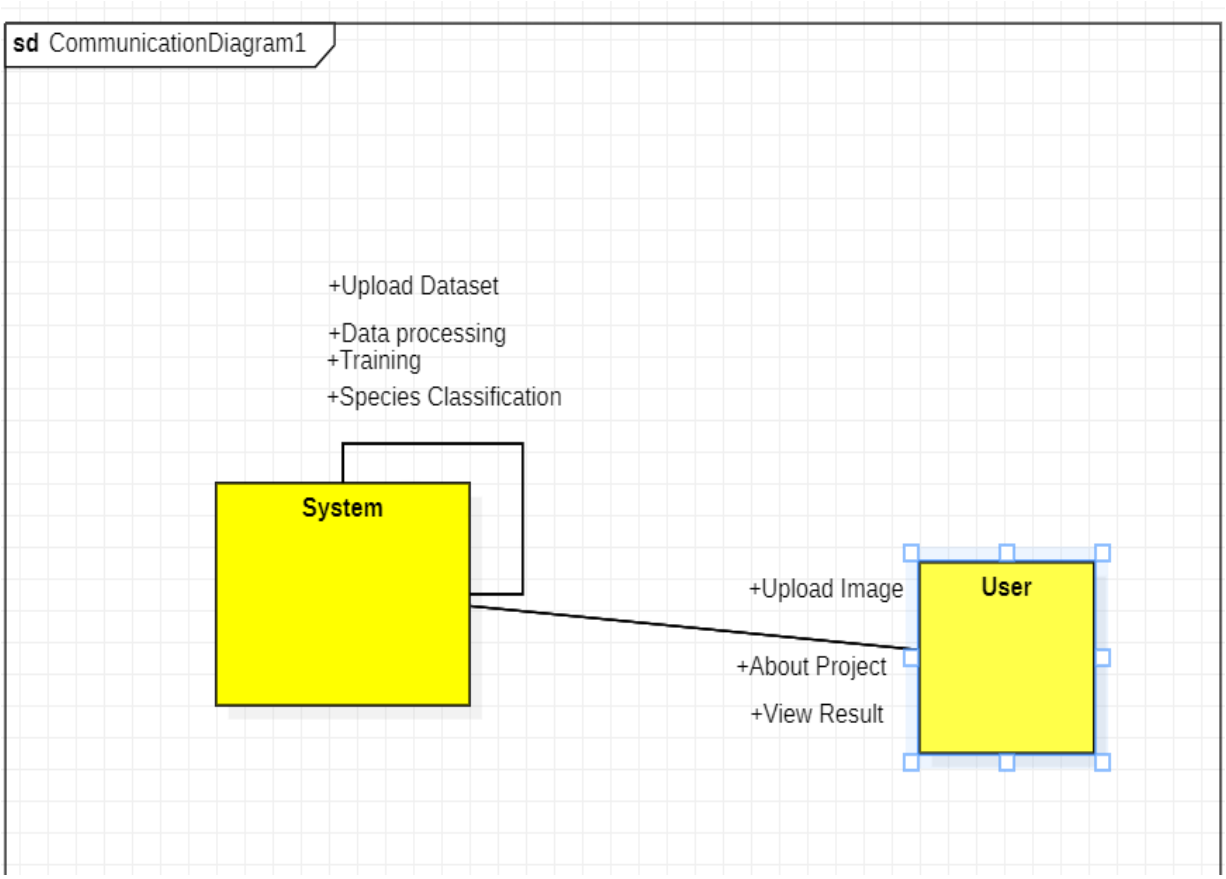

Figure 5.4 Deployment Diagram

5.6 DEPLOYMENT DIAGRAM

The Component Diagram is similar to the Exploded Diagram, which depicts an exploded picture of the system. This is so that components may be seen in an exploded view. Nodes are found in a deployment diagram. Nodes are merely the actual equipment required to distribute applications. The system and user perspectives for deployment are shown in diagram Figure 5.5.

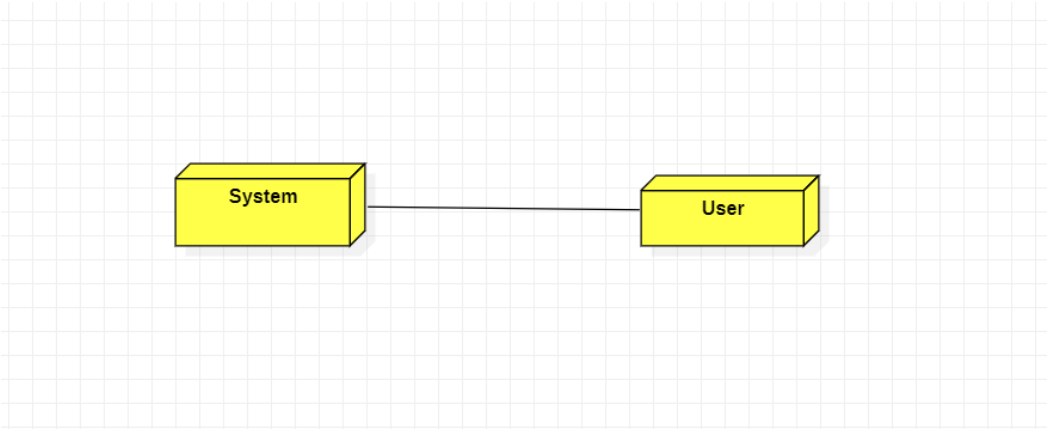

Figure 5.5 Deployment Diagram

5.7 ACTIVITY DIAGRAM

A flowchart of sequential tasks and actions that supports selection, repetition, and concurrency is called an activity diagram. Activity diagrams let you step-by-step define the operational and business workflows of the different system components using the Unified Modeling Language. Activity flow diagrams demonstrate the entire control flow in Figure 5.6.

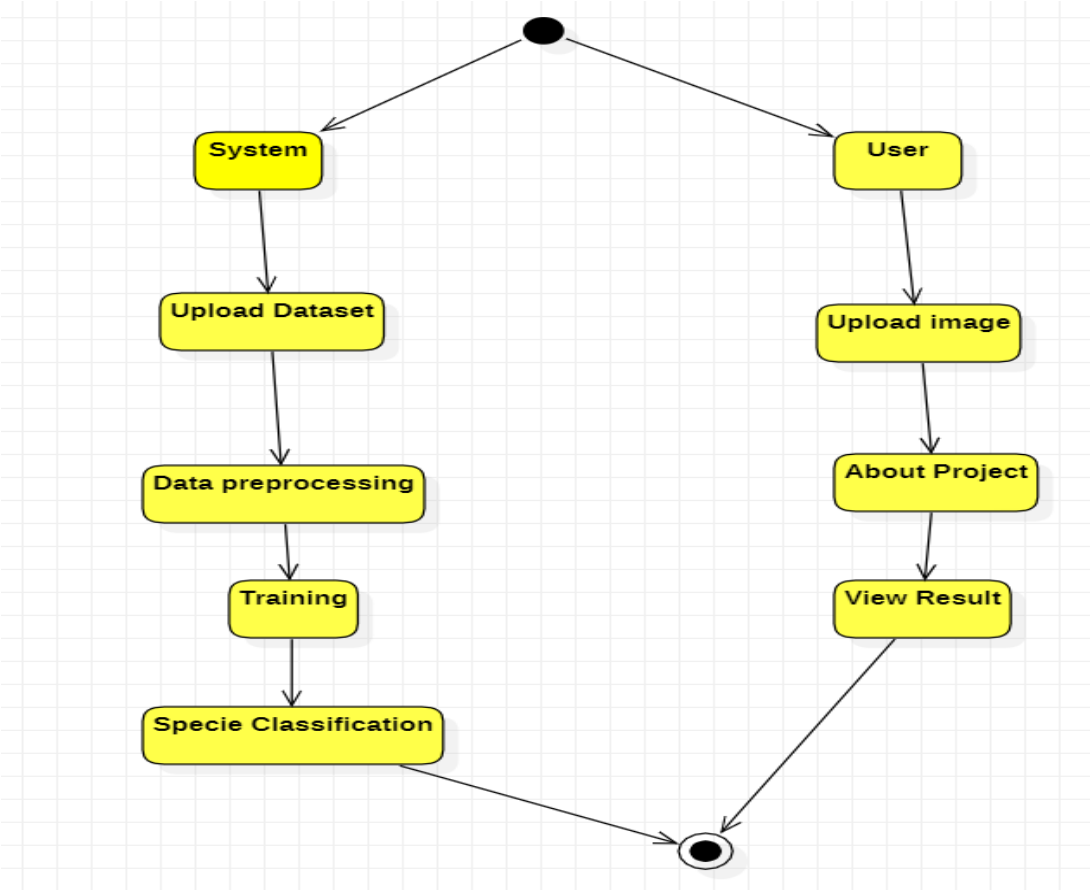

Figure 5.6 Activity Diagram

5.8 COMPONENT DIAGRAM

Component diagrams, commonly referred to as UML component diagrams, show how the physical components of a system are configured and wired together. To represent implementation specifics and make sure that all system requirements are addressed in the planned development, component diagrams are frequently generated in Figure 5.7.

Figure 5.7 Component Diagram

5.9 ER DIAGRAM

A database's structure is represented using an entity-relationship model (ER model), which is frequently displayed visually through an entity-relationship diagram (ER diagram). An ER model is a database design or blueprint that can be used to create a database in the future. A unit and a relation set are the two major elements of an E-R model.

A set is a collection of related objects that may also contain properties. Since a database table or attribute is referred to as an entity in DBMS terminology, an ER diagram depicts the relationships between tables and their attributes to depict the entire logical structure of a database. Let's look at a basic ER diagram to better understand this idea. Figure 5.8 illustrates the relationship between the entity and attribute used to build plant species classification.

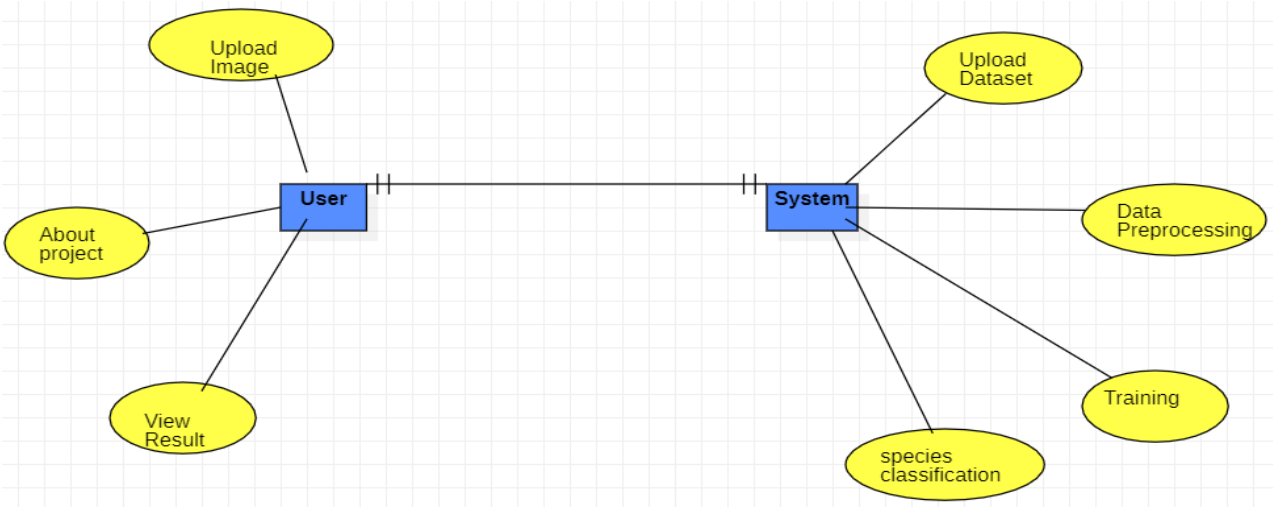

Figure 5.8 Entity Relationship Diagram

5.10 DFD DIAGRAM

A data flow diagram (DFD) is a common tool for illustrating how information moves through a system. A large percentage of the system requirements can be graphically represented by a well-organized DFD. Either manual, automatic or a hybrid of the two is possible. It demonstrates how data enters and exits the system, what changes, and where it is kept. The DFD is used to describe the boundaries and scope of the entire system. It can be utilized as a means of communication between the system analyst and everyone involved with the system as well as a place to start when rebuilding it**.** In the diagram, Figure 5.9 shows how the external entities are connected to the process to flow their data to the dataset to build the plant species classification.

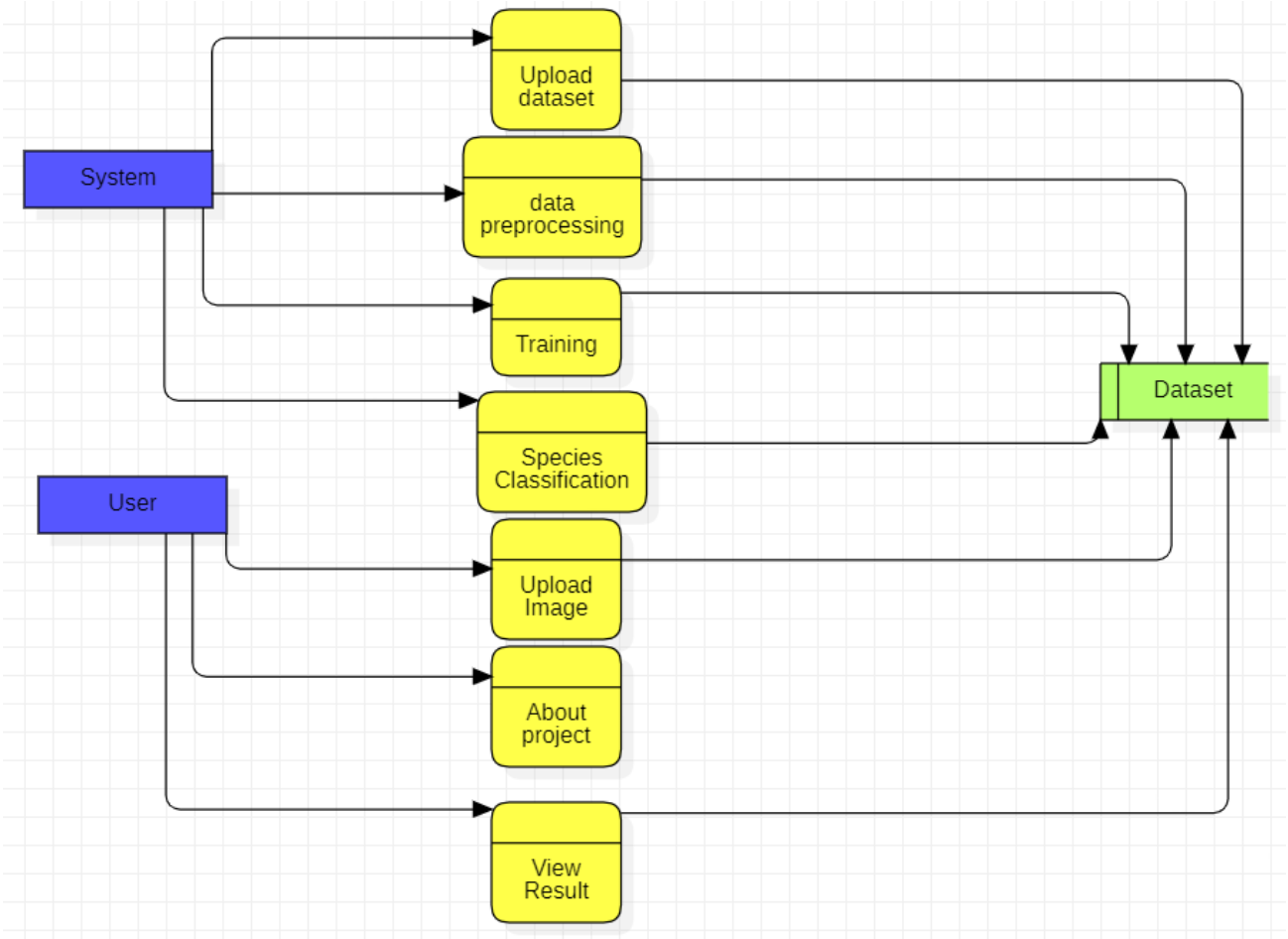

Figure 5.9 DFD diagram

CHAPTER 6

SOFTWARE TESTING

Testing serves the essential goal of uncovering errors within a system. This comprehensive process involves systematically identifying all possible flaws or vulnerabilities in a given work product. It offers a means to assess the performance of individual components, sub-assemblies, assemblies, or the final product. Testing is the practice of rigorously evaluating software to confirm its alignment with specified requirements and user expectations while ensuring it doesn't exhibit unacceptable failures. Different test types cater to distinct testing needs and criteria.

6.1 Testing Activities

The testing process employs various testing levels, each of which tries to test a distinct component of the system.

6.1.1 Unit Testing

Unit testing entails the creation of test cases designed to confirm the proper functioning of the internal program logic, as well as the generation of valid outputs from program inputs. It aims to validate all decision branches and the internal code flow. This form of testing focuses on assessing individual software units within the application and occurs after the completion of each unit but before integration. Unit testing is categorized as a structural test and requires an understanding of the unit's construction, making it an intrusive testing approach. These tests primarily examine fundamental aspects at the component level and ensure that specific business processes, applications, or system configurations adhere to documented specifications, with well-defined inputs and expected outcomes.

6.1.2 Integration Testing

Integration testing is crafted to evaluate the functionality of integrated software components, ensuring they operate seamlessly as a unified program. This testing approach is driven by events and primarily focuses on verifying the fundamental outcomes of screens or fields. Its key purpose is to showcase that while individual components may have passed unit testing satisfactorily, the amalgamation of these components is both accurate and harmonious. Integration testing is specifically tailored to uncover issues that may arise from the interaction and integration of these components.

6.1.3 Functional Testing

Functional tests serve as structured demonstrations, confirming that the functions under examination adhere to the stipulated criteria outlined in business and technical requirements, system documentation, and user manuals.

Functional testing is centred on the following items:

Valid Input : Ensuring that recognized categories of valid input are accepted as expected.

Invalid Input : Verifying that recognized categories of invalid input are appropriately rejected.

Functions : Exercising identified functions to validate their functionality.

Output : It is necessary to exercise the identified classes of application output.

Systems : Invoking interfacing systems or procedures as required.

Effective organization and preparation of functional tests are primarily driven by a focus on requirements, key functions, and special test cases. Moreover, a methodical approach is essential to cover business process flows, data fields, predefined processes, and sequential processes comprehensively during testing. Prior to concluding functional testing, additional tests are identified, and the current tests are assessed for their effectiveness in delivering value.

6.1.4 System Testing

System testing is a comprehensive assessment that verifies whether the integrated software system aligns with the established requirements. It scrutinizes system configurations to confirm expected and foreseeable outcomes, in configuration-oriented system integration testing, for instance. This type of testing emphasizes integration junctions and predetermined process links by relying on process flows and descriptions.

6.1.5 White Box Testing

White Box Testing is a testing method where the software tester possesses insights into the internal mechanisms, structure, and coding of the software, or at the very least, its intended functionality. It is employed to examine aspects of the software that are not accessible through black box testing techniques.

6.1.6 Black Box Testing

Black Box Testing is a testing approach where the software is assessed without any insight into its internal workings, structure, or codebase. These tests, like many others, are developed based on a well-defined source document, such as a specification or requirements document. In this type of testing, the software being evaluated is treated as an opaque entity, much like a "black box," where the focus is solely on inputting data and evaluating the corresponding outputs, without delving into the software's internal operations.

6.2 Unit Testing

Unit testing typically occurs within a unified code and unit testing phase during the software development lifecycle, although it's not unusual for coding and unit testing to be carried out as separate, discrete phases.

Test strategy and approach

Manual field testing will be conducted, and comprehensive functional test cases will be documented in detail.

Test objectives

- Each field entry must function properly.
- The designated link must be used to activate the pages.

Features to be tested

- It should be checked that the entries follow the proper structure.
- That duplicate entries are not permitted.

6.3 Integration Testing

Software integration testing involves the gradual integration testing of multiple software components on a single platform. Its purpose is to uncover issues stemming from interface defects, leading to the identification of failures in the integrated system.

Integration testing aims to verify that various components or software applications, whether within a software system or at a broader company level, can interact seamlessly and without encountering errors.

Test Results: All of the aforementioned test cases were successful. There were no problems.

6.4 Acceptance Testing

User Acceptance Testing (UAT) represents a pivotal project phase that necessitates active involvement from end users and serves as a key step in confirming the system's alignment with its functional requirements. Table 6.1 will represent all of the aforementioned test cases that were successfully passed.

Table 6.1 Test Case

TEST CASES:

Input	Output	Result
Input text	Tested for the different plant species classification using five different species	Success

A variety of test cases are used to evaluate the model. We employed 4 test cases: reading the dataset, preprocessing the dataset, creating the model, and classification. As demonstrated in Table 6.2, all test cases were successfully completed.

Table 6.2 Test Case Model Building

Test cases Model building:

S.NO	Test cases	I/O	Expected O/T	Actual O/T	Pass/Fail
1	Read the dataset.	Dataset path.	The dataset needs to be read successfully.	Dataset was fetched successfully.	Pass
2	Performing pre-processing on the dataset	The pre-processing takes place	Pre-processing should be performed on a dataset.	Pre-processing successfully completed.	Pass
3	Model Building	Model Building for the clean data	Need to create a model using required algorithms.	The model created successfully.	Pass
4	Classification	Input image provided.	The output should be the types of plant species.	Model classified successfully	Pass

CHAPTER 7

IMPLEMENTATION

In the software development process, the implementation phase assumes a paramount significance. This stage involves the actual crafting of the system's source code, where programmers utilize the collected requirements and project documentation as their blueprint. Their approach is rooted in their expertise and tried-and-true software development methodologies.

7.1 Pseudocode

Pseudocode represents a comprehensive yet clear explanation of the intended actions and logic of a computer program or algorithm. It adopts a formal, yet easily readable format, employing natural language syntax and structure, which promotes accessibility and understanding for both programmers and other stakeholders participating in the development process.

Here we wrote a pseudocode for the plant species classification model by file name model.py.

Model.py

```
# Import necessary libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot as plt
# Build the deep learning model
model = Sequential([
Conv2D(512, (3, 3), activation='relu', input_shape=(length, width, 3)),
MaxPooling2D(3, 3),
Conv2D(256, (2, 2), activation='relu'),
MaxPooling2D(1, 1),
Flatten(),
Dense(5, activation='relu'),
```

```
Dense(num_classes, activation='softmax')  # num_classes is the number of plant species ])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Load and preprocess the dataset
# Assuming you have a directory structure with subdirectories for each class
train_data_dir = 'path_to_train_data_directory'
image_size = (64,64)  # Adjust according to your model's input size
batch_size = 12
train_datagen = ImageDataGenerator(
rescale=1.0 / 255,
shear_range=in between 0 -1,
zoom_range= in between 0 -1,
horizontal_flip=True )
train_generator = train_datagen.flow_from_directory(
train_data_dir,
target_size=image size,
batch_size=batch size,
class_mode='categorical',)
validation_generator = train_datagen.flow_from_directory(
train_data_dir,
target_size=image_size,
batch_size=batch_size,
class_mode='categorical',)
# Train the model
epochs = 50 # Adjust based on your training needs
history = model.fit(
train_generator,
```

```
epochs=epochs,
validation_data=validation_generator,
verbose=0, 1 or 2)
# Save the model for future use
model.save('plant_species_model.h5')
# Plot the accuracy graph
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy Over Training Epochs')
plt.legend()
plt.savefig(r"visualizations\model_1_loss.png")
plt.show()
# Plot the loss graph
plt.plot(history.history['loss'], label='Training Accuracy')
plt.plot(history.history['val_loss'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.title('loss Over Training Epochs')
plt.legend()
plt.savefig(r"visualizations\model_1_loss.png")
plt.show()
# showing total accuracy of the model
history.history['val_accuracy']
```

This pseudocode included code to generate an accuracy graph using Matplotlib. The graph will show how the training accuracy and validation accuracy change over the training

epochs. Remember to replace 'path_to_train_data_directory' and other placeholders with the actual paths and values specific to our project.

App.py

Here's a pseudocode outline for a Python Flask web application that includes multiple HTML pages (index.html, about.html, upload.html) along with a shared template (template.html) using the Jinja2 templating engine. These files will be included in a single app.py file.

```
# Import necessary libraries

import os

from flask import Flask, request, render_template, send_from_directory,flash

from PIL import Image

from pylab import *

import numpy as np

from tensorflow.keras.preprocessing import image

from tensorflow.keras.models import load_model

# Create a Flask app

app = Flask(__name__)

create the classes for the datasets

classes=['dateset_name','dataset_name','dataset_name']

# Define routes and view functions

# Route for the home page (index.html)

@app.route('/')

def index():

return render_template('index.html')

# Route for the about page (about.html)

@app.route('/about')

def about():

return render_template('about.html')
```

```
# Route for the upload page (upload.html)
@app.route("/upload")
def upload():
return render_template("upload.html")
@app.route('/upload1', methods=['GET', 'POST'])
def upload():
if request.method == 'POST':
# Handle file upload logic here
# Example: Save uploaded file and process it
uploaded_file = request.files['file']
# Process the uploaded file
filename = myfile.filename
mypath = os.path.join('images/', fn) # path of image location
myfile.save(mypath)
return redirect(url_for('index'))
# Redirect to the home page after processing
return render_template('upload.html')
#load the image in model.h5
load_model("model.h5")
#load the path with certain size(height, width)
image.load_img(mypath, target_size=(64,64))
#arrange in array(numpy)
image.img_to_array(test_image)
# Expand the dimension of image axis must be at 0 level
np.expand_dims(test_image, axis)
# by tensorflow library we use to predict the imported image using classes allocated
new_model.predict(test_image)
```

return render_template("template.html", image_name= image_name, predication=prediction)

Run the app

if __name__ == '__main__':

app.run(debug=True)

This pseudocode included code connecting various HTML pages, which render themselves to generate output from a template file based on the Jinja2 engine that is found in the application's templates folder. Even this app.py include the loading of new test image from the path given by the user, loads in the Hierarchical Data Formats then later it will resize the image, resized images are push in array and classifies the images and its specie name by classes names.

CHAPTER 8

OUTPUT RESULT

8.1 Home

In our project, we are classifying the different plant species with the help of deep learning, Figure 8.1, is home page which is connected to many other pages.

Figure 8.1 Home page

8.2 About Us

The page depicted in Figure 8.2, shows the project's concept is briefly explained.

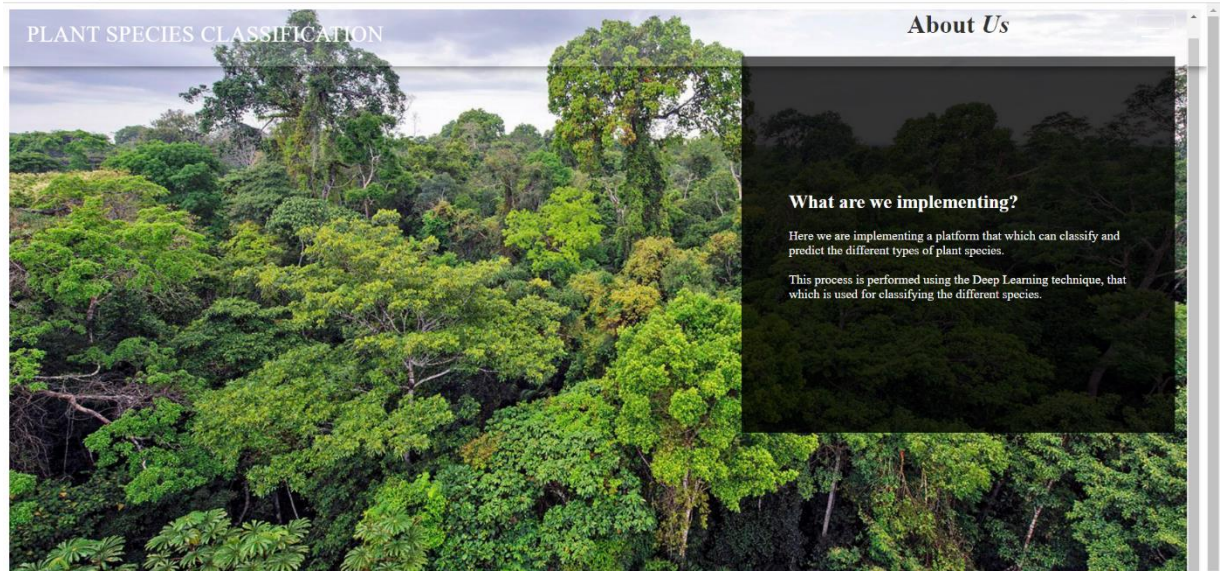

Figure 8.2 About page

8.3 Upload Page

The user typically inserts the image into a preprocessor to obtain the classification result of plant species classification, as shown in Figure 8.3, which is an image upload page.

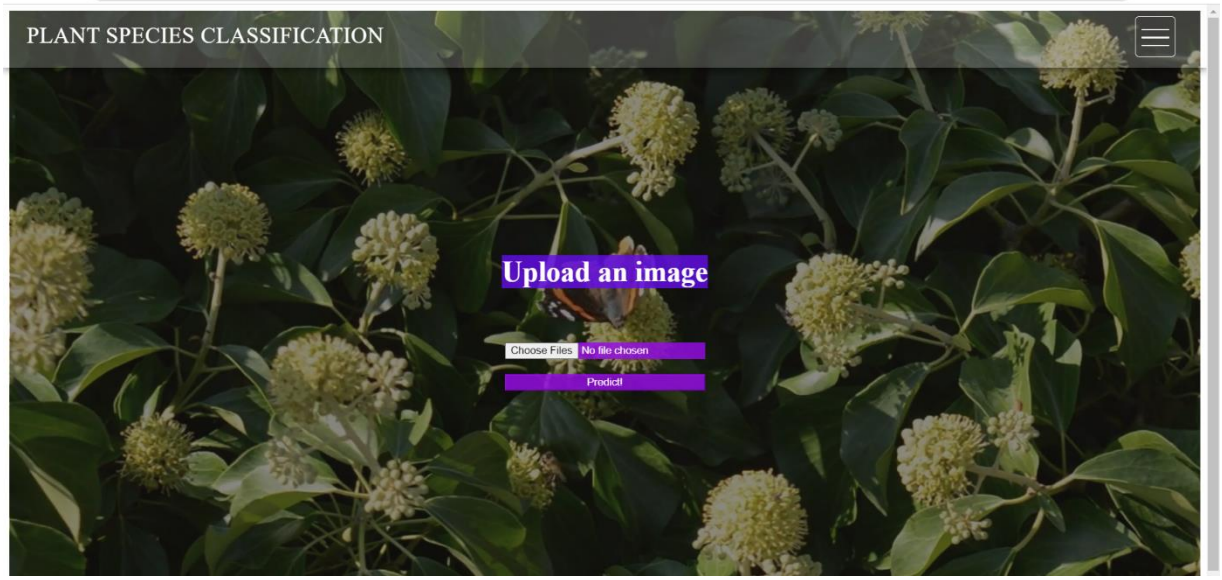

Figure 8.3 Upload Page

8.4 Classified Output

The project's final page, shown in Figure 8.4, displays the species name and the image that the user chose and added to the upload page, as seen in Figure 8.3. The tomato leaf image has been inserted here by the user.

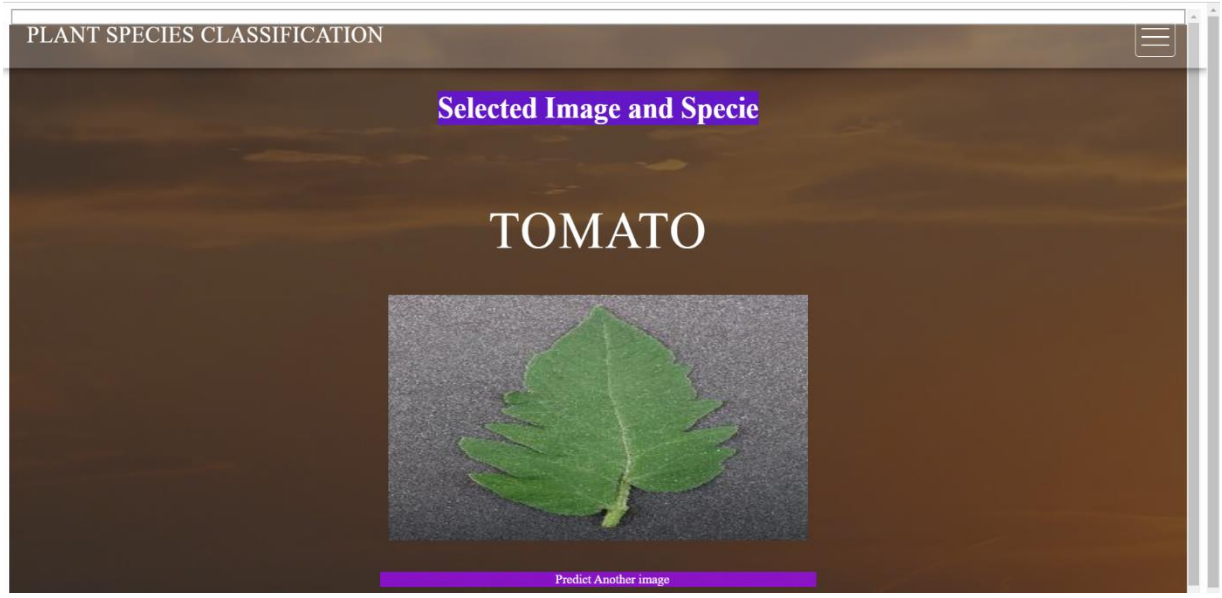

Figure 8.4 Tomato Classified Output

8.5 Classified Output

The upload page result is displayed as shown in Figure 8.5 after the user entered an image of an apple leaf.

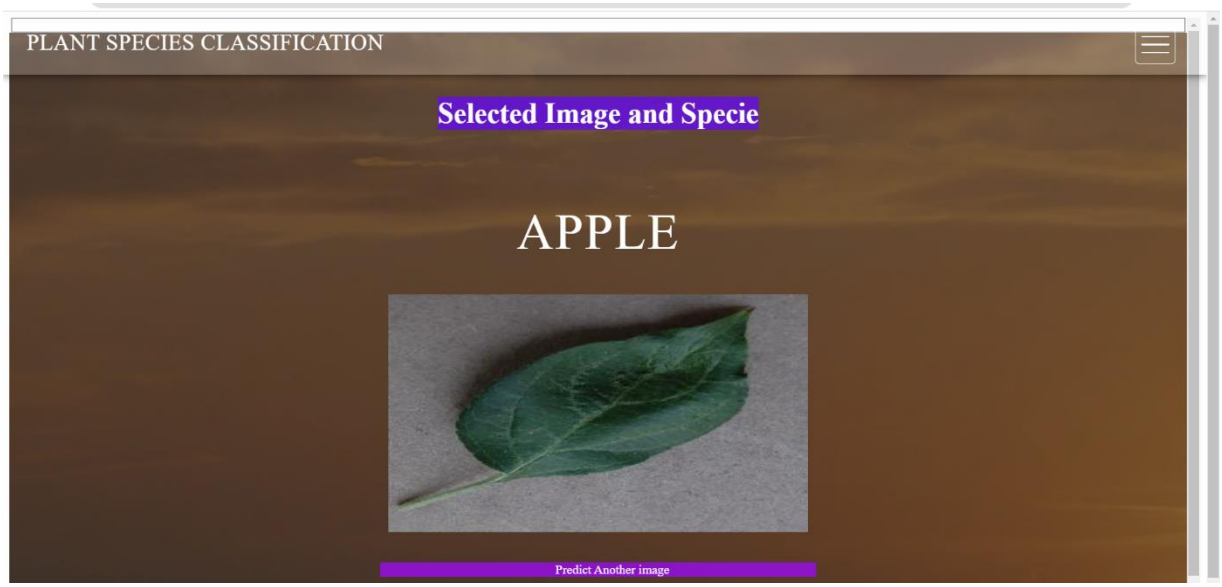

Figure 8.5 Apple Classified Output

8.6 Classified Output

The upload page result is displayed as shown in Figure 8.6 after the user entered an image of a corn leaf.

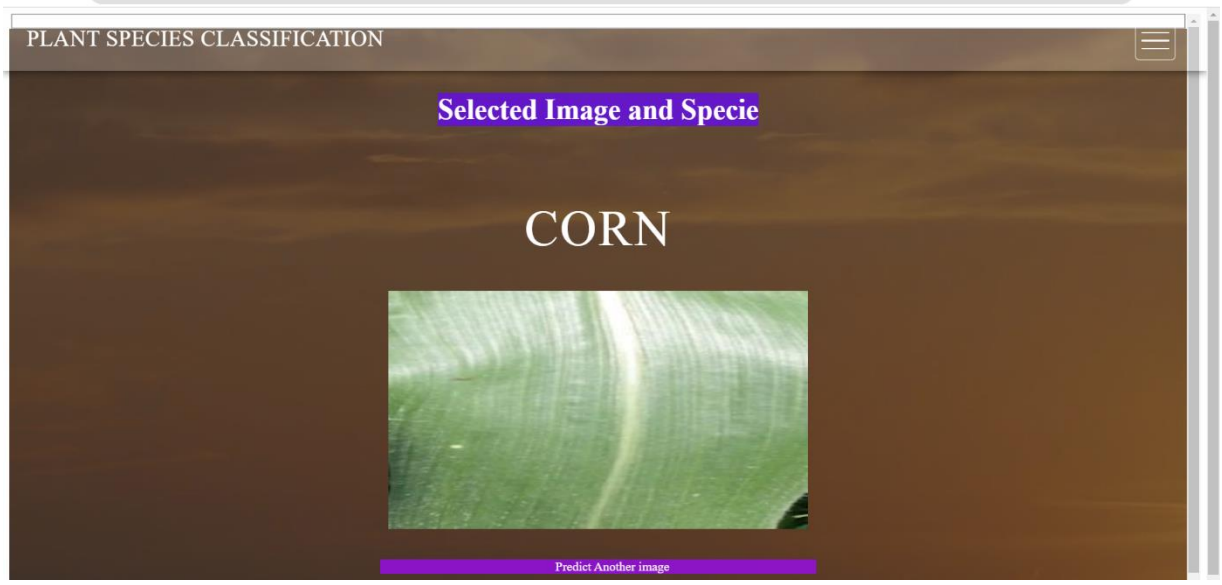

Figure 8.6 Corn Classified output

8.7 Classified Output

The upload page result is displayed as shown in Figure 8.7 after the user entered an image of a grape leaf.

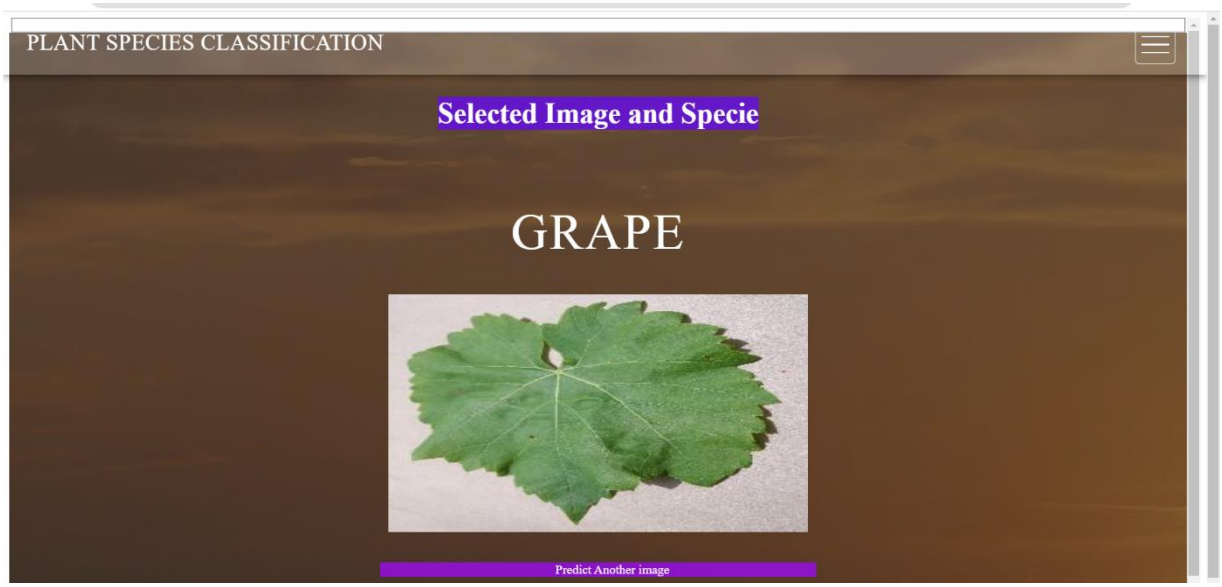

Figure 8.7 Grape Classified Output

8.8 Classified Output

The upload page result is displayed as shown in Figure 8.8 after the user entered an image of a peach leaf.

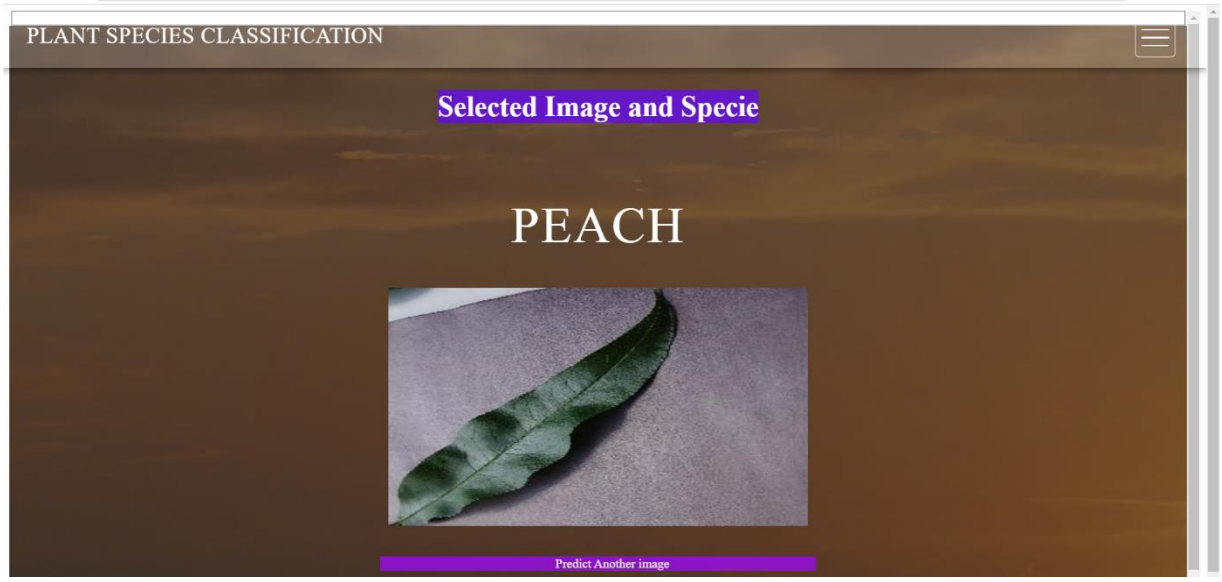

Figure 8.8 Peach Classified Output

8.9 Training Accuracy and Validation Accuracy

Training accuracy is a metric used in deep learning to measure how accurately a model predicts the target values for the data it was trained on. Validation accuracy assesses a machine learning model's correctness when predicting on a separate dataset not used in training, gauging its generalization performance. In Figure 8.9, we can see the accuracy based on epochs, where we used two values, one for training accuracy and one for validation accuracy, with training accuracy receiving 0.9833 and validation accuracy receiving 0.9929.

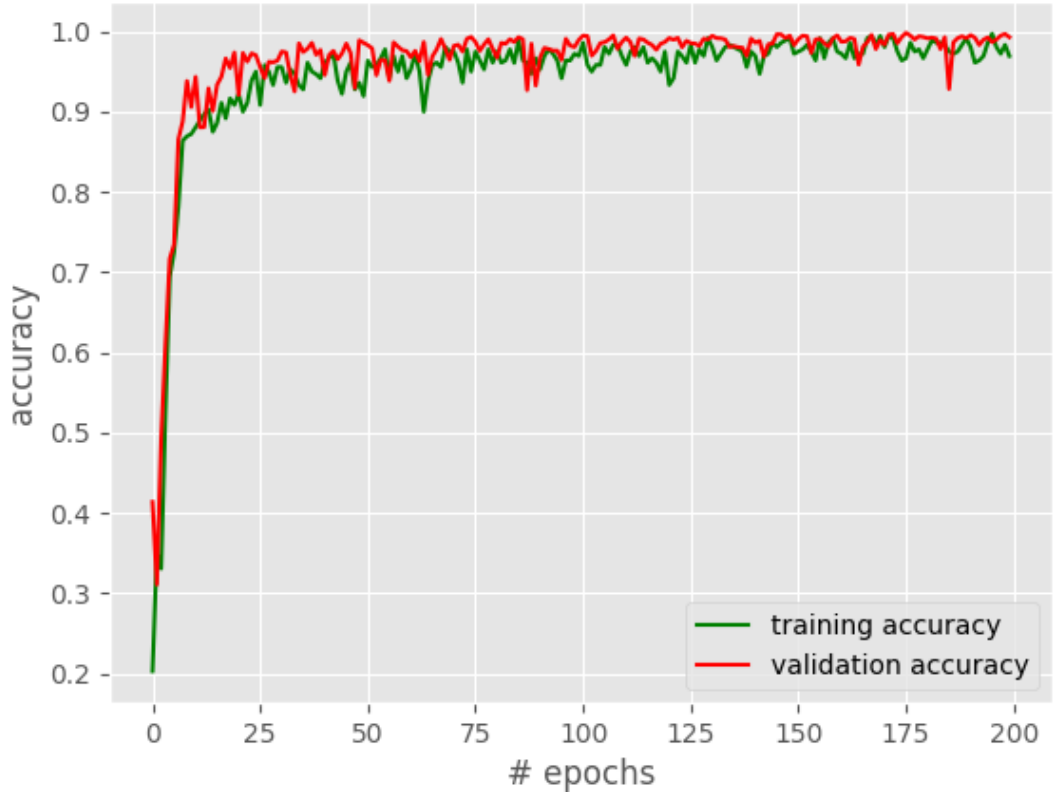

Figure 8.9 Training Accuracy & Validation Accuracy

8.10 Training Loss and Validation Loss

Training loss, in deep learning, quantifies the error between predicted and actual values during training, guiding model optimization to minimize errors and enhance learning. Validation loss in deep learning measures the error between model predictions and actual values on a separate validation dataset, helping to assess generalization and model performance on unseen data during training. In Figure 8.10, we can see the loss based on epochs, where we used two values, one for training loss and one for validation loss, with training loss receiving 0.0944 and validation loss receiving 0.0211.

Figure 8.10 Training Loss & Validation Loss

CHAPTER 9

CONCLUSION & FUTURE SCOPE

In this study, employed a computer vision and Deep learning approach to classify plant leaf images, conducted in distinct phases encompassing image pre-processing, segmentation, feature extraction, and image classification. A combination of both texture and color features was harnessed, followed by the utilization of a linear CNN classifier for the categorization task. The system was rigorously tested on a comprehensive dataset, yielding an accuracy exceeding 99.289%, all accomplished within the TensorFlow framework. Remarkably, our model demonstrated the capability to autonomously discern among 5 diverse plant species. The proposed methodology stands out for its ease of implementation and remarkable efficiency. Nevertheless, while our model achieved commendable accuracy surpassing 99%, it still falls behind approaches that leverage advanced neural networks and deep learning techniques. Looking ahead, the ultimate objective is to advance the concept of automatic plant species identification by embarking on the exciting journey of working with real-time, live datasets.

In the future, this method might be expanded to categorize more plant categories with more species using a huge dataset, which could result in a simple method for classifying and forecasting various plant species without the need for additional human labour.

CHAPTER 10

REFERENCES

[1] D. Bisen, ‘‘Deep convolutional neural network-based plant species recognition through features of leaf,’’ Multimedia Tools Appl., vol. 80, no. 4, pp. 6443–6456, 2020, Doi: 10.1007/s11042-020-10038-w

[2] S. Patil, B. Patra, N. Goyal, and K. Gupta, “Recognizing Plant species using Digitized leaves- A comparative study,” International Conference on Trends in Electronics and Informatics (ICOEI). IEEE Xplore Part Number: CFP21J32-ART; ISBN:978-1-6654-1571-2., 2021.