THE FUTURE OF AYURVEDA: HARNESSING THE POWER OF ARTIFICIAL INTELLIGENCE FOR PERSONALIZED TREATMENT AND DIAGNOSIS

De Silva A.S

IT20166038

B.Sc. (Hons) Degree in Information Technology Specialized in Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology Sri Lanka

September 2023

DECLARATION

I declare that this is my own work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Name	**Student ID**	**Signature**
De Silva A. S	IT20166038	

Signature of the Supervisor
(Dr. Darshana Kasthurirathna)

Date

……. ..……

ABSTRACT

The rapid and busy nature of contemporary living often results in an uneven distribution of daily activities, inadequate dietary habits, insufficient physical exercise and leisure time, and excessive work-related stress, which can lead to poor health and dissatisfaction. While Ayurveda provides alternative solutions for many diseases. But people may have difficulties identifying the appropriate herbs and treatments and consulting with doctors in a timely and cost-effective manner. Additionally, the high cost of Western medicine may be a barrier, and not all illnesses are treatable. The proposed solution aims to assist users in discovering interactive Ayurvedic-based treatments for various symptoms. This solution is anticipated to be beneficial for those seeking alternatives to conventional medicine.

The identification of Ayurvedic medical herbs is a critical step in the treatment of various diseases. Traditional methods of identification involve physical examination and experience, which can be time-consuming and prone to errors. To address this issue, this study proposes a new approach that uses image processing and machine learning techniques to identify Ayurvedic medical herbs for the treatment of diseases such as diabetes, arthritis, and asthma.

The study then collected and processed large amounts of image data from various geographical areas to map the distribution of herbal plants. Finally, the model was implemented in a software application for practical use. As a novelty improvement, the study incorporated continual learning/transfer learning to improve accuracy and auto-machine learning to make it easier to add new plants. The proposed approach has the potential to significantly improve the accuracy and efficiency of Ayurvedic medical herb identification and treatment, making it a valuable tool for the healthcare industry.

TABLE OF CONTENTS

Contents

ACKNOWLEDGEMENT

First and foremost, I want to sincerely thank my mentor Dr. Darshana Kasthurirathna for his unwavering support and direction, which enabled me to successfully complete my undergraduate research. Along with my supervisor, Dr. Samantha Rajapaksha, the co-supervisor of this research project deserves my deepest gratitude for always being available to assist. Due to the fact that this research project combines technology and ayurveda, both technology specialists and ayurvedic professionals were needed for advice and support. It is very appreciated that Dr. Mrs. Janaki Bandara who provided such extensive help throughout the project to close the knowledge gap in those fields. My sincere gratitude also goes out to the other doctors of Gampaha Wikramarachchi Ayurvedic University. Last but not least, I would like to convey my thanks to everyone who has helped with this project in some way, whether directly or indirectly, including my teammates, family, and friends.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

Abbreviation	Description
AI	Artificial Intelligence
ML	Machine Learning
EHR	Electronic Health Records
AML	Auto Machine Learning
CNN	Convolutional Neural Network
SVM	Support Vector Machine
SDLC	Software Development Life Cycle
HOG	Histogram of Oriented Gradients
SIFT	Scale-Invariant Feature Transform
TP	True Positives
FP	False Positives
TN	True Negative
FN	False Negative
YOLO	You Only Look Once

Table 1: List of Abbreviations

LIST OF APPENDICES

1. INTRODUCTION

1.1 Background & Literature survey

Ayurveda is a traditional medical practice with Indian roots that places a strong emphasis on a holistic view of health and wellness. It uses organic treatments, dietary adjustments, and individualized care based on a person's dosha, or body type. Because of its efficacy and focus on prevention and individualized care, Ayurveda has become increasingly well-known on a global scale. A more specialized and individualized approach to treatment is needed for non-communicable diseases like diabetes, heart disease, and cancer because of the fast-paced modern lifestyle. In order to provide individualized and efficient therapy, this has led researchers and practitioners to investigate how Ayurveda principles might be combined with cutting-edge technology like Artificial Intelligence (AI).

Our research indicates that Sri Lankans have a long-standing cultural and traditional connection to Ayurveda. The history of the nation dates back more than 3,000 years, and Sri Lankan culture is firmly rooted in the practice of ayurveda. In Sri Lanka, different ailments have been treated and prevented through Ayurveda practices for thousands of years. The popularity of Ayurveda in Sri Lanka can be ascribed to its all-encompassing method of healthcare, which places an emphasis on the harmony of the mind, body, and spirit. The natural and organic aspects of Ayurveda, which uses herbs and other natural therapies to address a variety of health conditions, have long been valued by Sri Lankans. For minor ailments and long-term health concerns, Sri Lankans favor Ayurveda therapies over contemporary medicine.

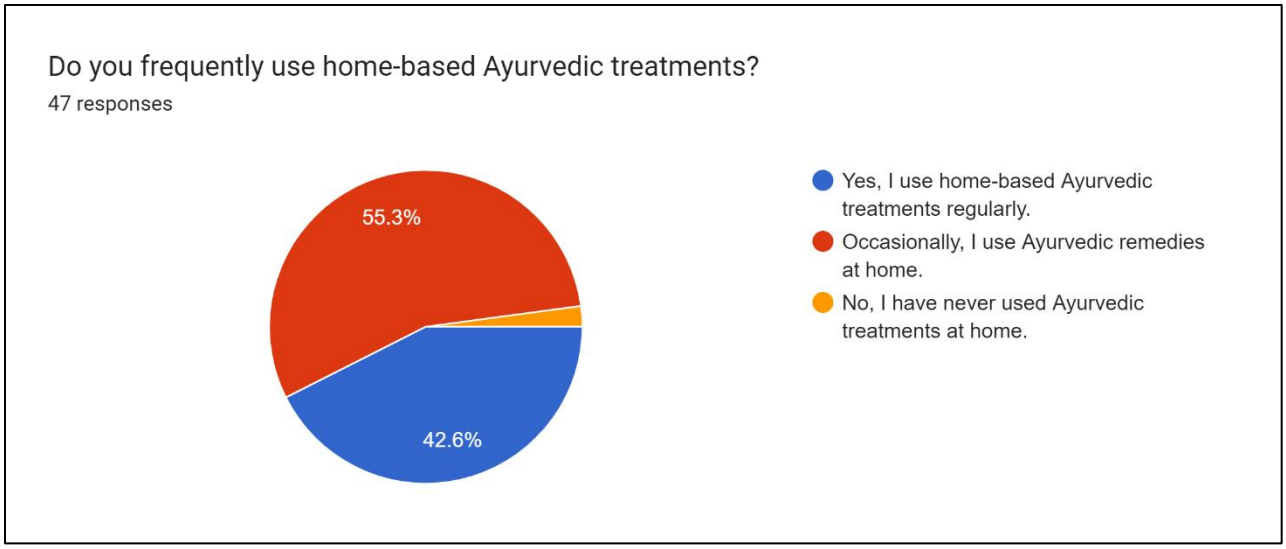

Figure 1: Survey on how frequently people use home based ayurvedic treatments.

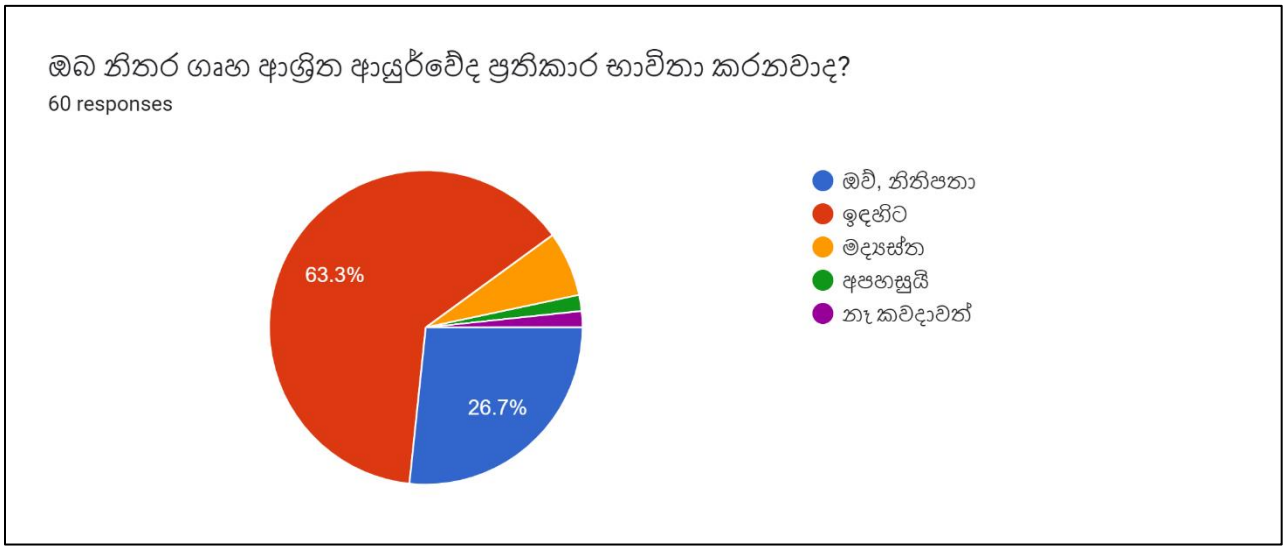

Figure 2:Survey in Sinhala on the frequency of home based ayurvedic treatments usage

Based on the survey results, it can be concluded that more than half of the respondents are frequently using ayurvedic treatments.

Ayurvedic hospitals and clinics may be found all over Sri Lanka, and Ayurveda has a significant influence on the country's healthcare system. The people of Sri Lanka have access to skilled and knowledgeable Ayurveda medical professionals who offer their patients individualized care and therapies. These medical professionals treat a variety of illnesses, from the ordinary cold to more serious disorders like cancer and diabetes, using Ayurveda treatments and conventional healing methods. As people in Sri Lanka become more health-conscious and explore for natural and

organic alternatives to modern medicine, there has been a resurgence in interest in Ayurveda recently. Ayurveda is a crucial component of Sri Lanka's healthcare system, and the government has taken steps to promote and advance it.

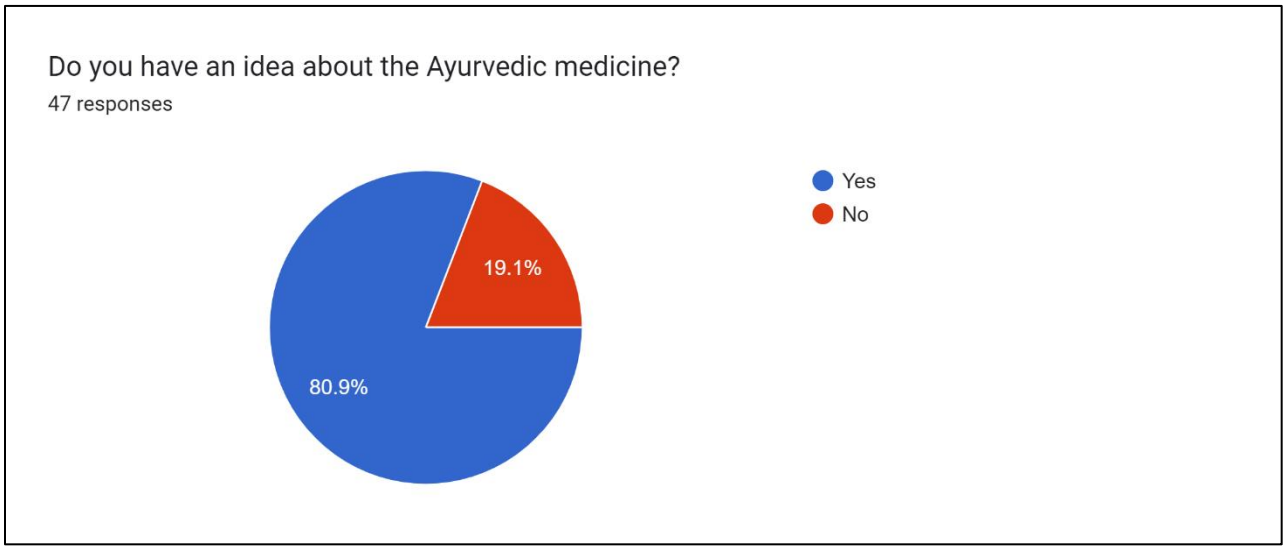

Figure 3: Survey on the idea of people about ayurvedic medicine

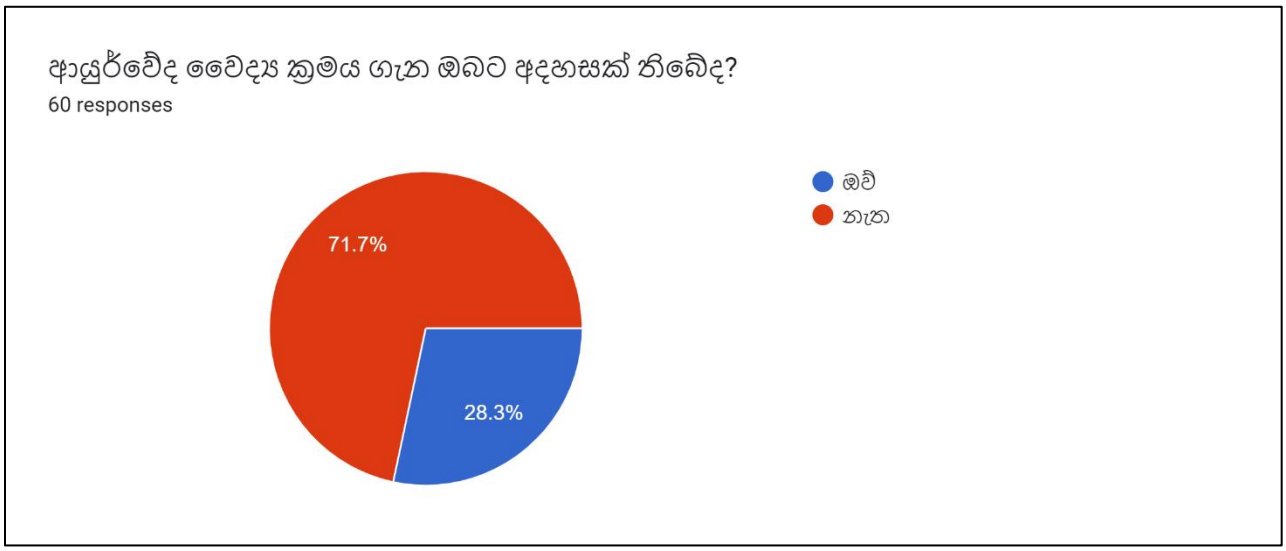

Figure 4:Survey in Sinhala on the idea of people about ayurvedic medicine

From the above two survey results it shows that most of the people in Sri Lanka are aware about the ayurvedic medicine and most people had used them even once in their lifetime. Considering the percentages, more than 80% of people are aware of ayurvedic medicine.

Overall, our research shows that Sri Lankans deeply value Ayurveda and its all-encompassing approach to healing. As more individuals look for natural and organic alternatives to contemporary medicine, Ayurveda is anticipated to gain popularity in Sri Lanka.

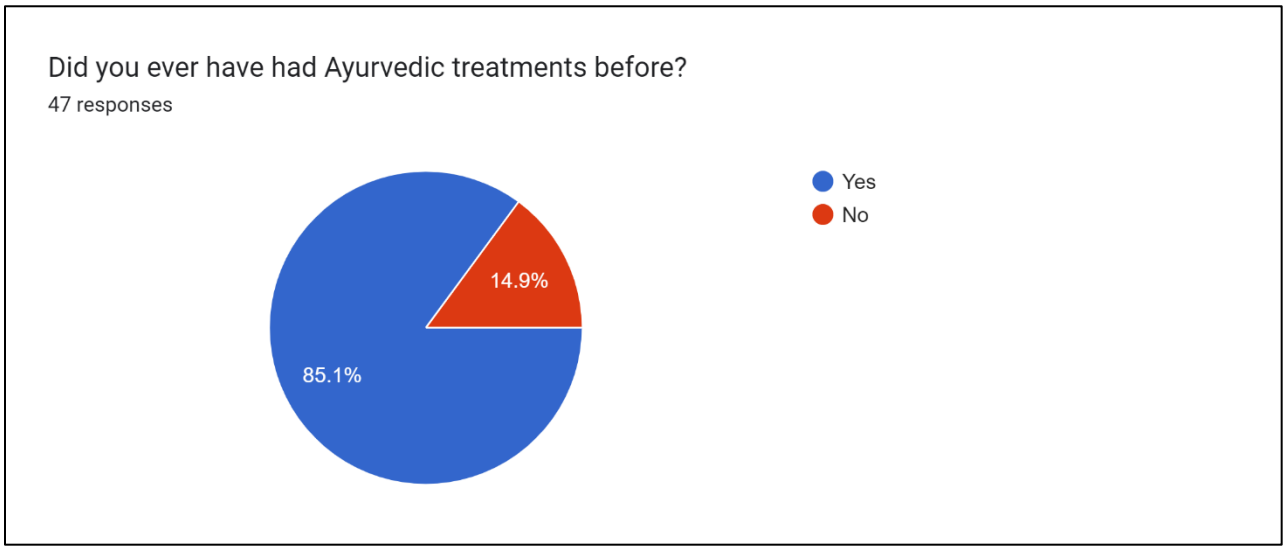

Figure 5: Survey on the people about do they had ayurvedic treatments before.

From the above two survey results it shows that most of the people in Sri Lanka had used them even once in their lifetime. Considering the percentages more than 70% of people are aware of ayurvedic medicine.

The healthcare sector is undergoing a change thanks to artificial intelligence (AI), which is providing fresh and creative approaches to the detection, treatment, and monitoring of diseases. [1] The ability of AI to evaluate enormous volumes of patient data and medical records to assist in illness detection is one of the most important benefits of AI in healthcare. Radiologists can be helped by AI algorithms in spotting potential anomalies or diseases in medical pictures like CT scans, MRIs, and X-rays. Additionally, by examining data on previous treatment outcomes and patient health records, AI can assist clinicians in developing individualized treatment regimens for patients based on their particular traits and medical history. By examining massive datasets to find trends and provide insights, AI can also help in the search for new pharmaceuticals and therapies. Moreover, chatbots and voice assistants powered by AI can offer a preliminary diagnostic and suggest next measures. Moreover, AI can be used to scan electronic health records (EHR) to find possible problems, remotely monitor patients and notify clinicians of any problems, and give individualized health advice and suggestions via virtual assistants. Overall, AI has great promise for enhancing patient outcomes and delivering more individualized and effective healthcare solutions.

Ayurveda could undergo a revolution thanks to artificial intelligence (AI), which would allow for individualized care based on each patient's particular traits, background, and symptoms. AI algorithms may find trends and offer insights by analyzing enormous volumes of patient data, which enables practitioners to make wise treatment decisions. There are already a number of Ayurveda-based applications that make individualized therapy recommendations using AI. AyurMana is one such software that employs AI algorithms to examine a patient's pulse, tongue, and symptoms in order to offer individualized treatments. Wealthy is a different program that uses AI-powered chatbots to offer individualized health tips.

The use of AI in Ayurveda offers many advantages. First of all, it gives medical professionals the ability to tailor treatment to each patient's particular needs, which can improve outcomes and lower medical expenses. Second, AI can assist in finding patterns and insights that can enhance disease diagnosis and therapy. Finally, apps and solutions driven by AI can improve patient engagement and treatment adherence. Ayurveda may make use of a number of AI technologies, including computer vision, natural language processing, and machine learning. Patterns can be found and recommendations for individual treatments can be made using machine learning. Chatbots and voice assistants can offer individualized health advice and recommendations thanks to natural language processing. Images of patients can be examined using computer vision to find symptoms and offer insights.

Here are a few of the ayurvedic apps that are already available. Yes, these smartphone applications offer Ayurveda remedies. [2]

- E-procto
- MocDocHMS
- MocDoc Clinic Management System
- MyOPD
- Vaidya Manager
- Healcon Practice
- Easy Clinic
- OptiMantra

Here are a few of the plant identification apps that are already available. [3] [4] [5]

- WhatIsThisPlant
- PlantI

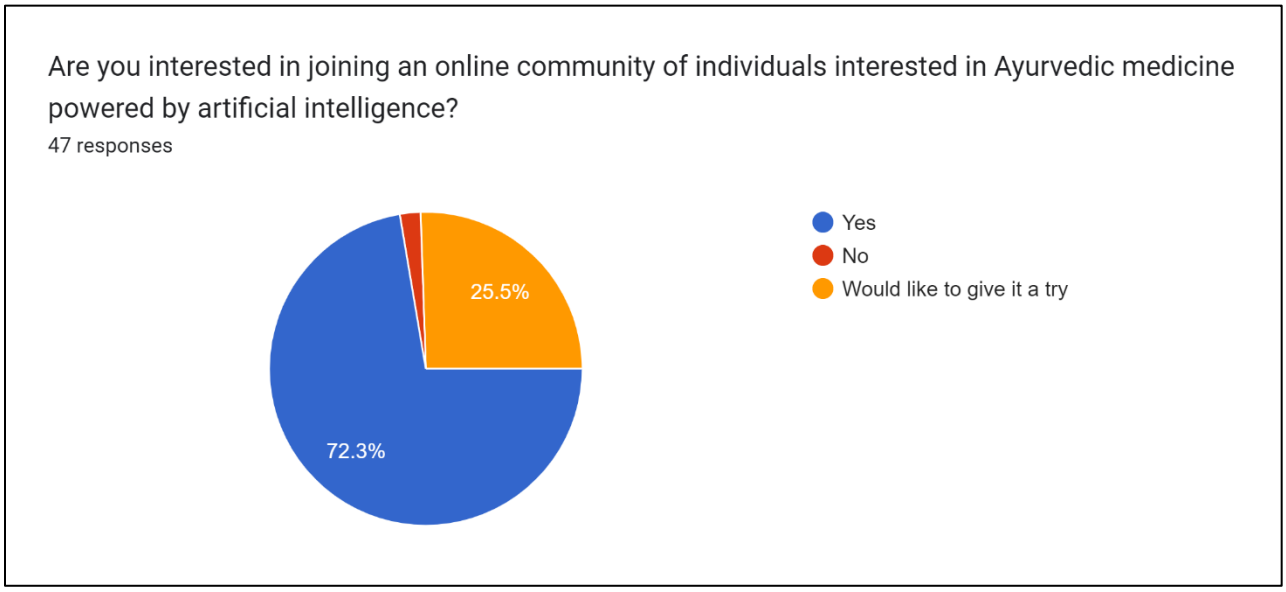

Figure 6: Survey on the interest of people to Ayurveda through AI

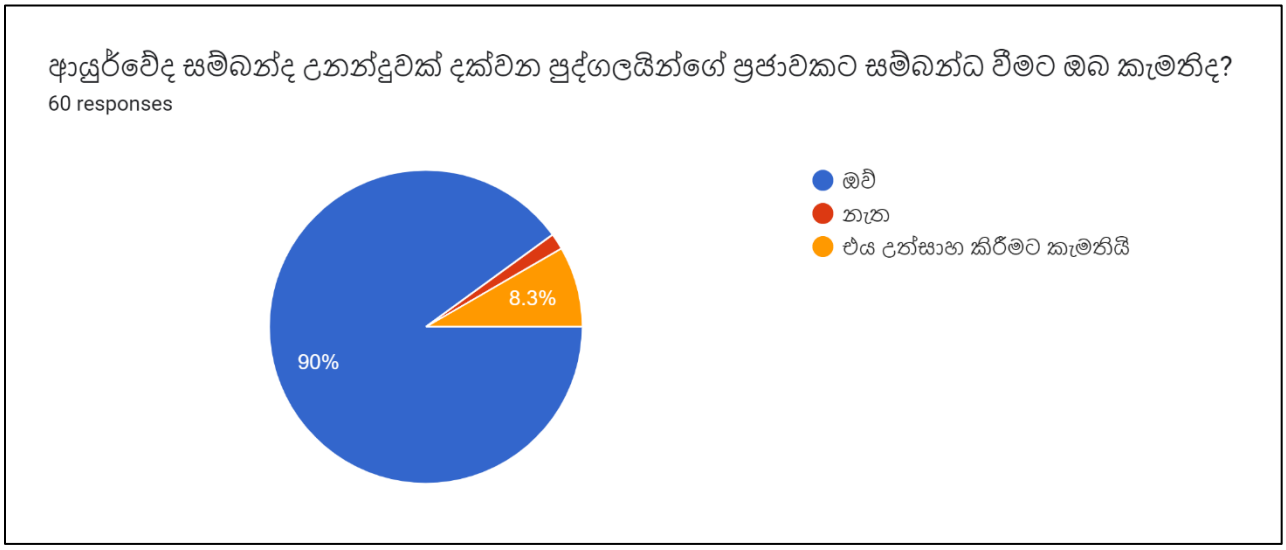

Figure 7: Survey in Sinhala on the interest of people to Ayurveda through AI

From the above two survey results it shows that most of the people in Sri Lanka are willing to join on an ayurvedic online platform.

As said in the preceding section, we suggest an original way to encourage a better lifestyle through Ayurveda in light of the **study background and survey presented above**. [6] [7]For specific

problems including arthritis, blood sugar, hair loss, infertility, obesity, paranasal sinusitis, wounds, scrapes, and swellings, our solution attempts to address Ayurvedic therapies as well as general healthy principles. A conversational AI chatbot will be part of our solution, offering a user-friendly platform for people to obtain tailored solutions and guidance via text, based on knowledge about these symptoms and their treatments.

Our suggested approach will include a chatbot in addition to an image processing element that can recognize the herbal plants required for treating various conditions. The locations of these plants will be mapped out by a geometry library, which will also link patients to Ayurvedic practitioners in a particular geographic region. Users can rate doctors to aid in the search for trustworthy medical professionals.

A social network will also be a part of our solution, allowing users to discuss information about health. Health-related content supplied by users will be gathered and kept in the knowledgebase as community knowledge for the chatbot to refer to. Auto-machine learning will keep the social network current and assist users with any new symptoms, assisting in maintaining the consistency of the solution.

We will need resources with extensive Ayurvedic-related expertise if we are to guarantee the accuracy and efficacy of our solution. For supervised machine learning algorithm-based solutions like herb identification, chatbot implementation, and social network implementation, we expect to gather data through publicly accessible social network communities relating to health and images of herbs, as well as details of Ayurvedic doctors with their locations from relevant backgrounds.

Finally, the use of AI to Ayurveda has the potential to revolutionize the industry of individualized healthcare. By utilizing AI, medical professionals can improve disease detection and treatment while also delivering tailored care. Better health outcomes can result from the creation of AI-powered apps and solutions that encourage patient participation and treatment compliance. The incorporation of AI in Ayurveda is taking on greater significance and has enormous promise for the future of healthcare as non-communicable diseases become more prevalent. [6] [7]

1.2 Research Gap

There is currently a lack of an AI-based solution that can accurately identify and map ayurvedic medical herbs needed for treatments of non-communicable diseases using image processing and machine learning models. While some image recognition models have been developed to identify herbs in general, they are not specifically trained to recognize ayurvedic medical herbs and are not optimized for identifying the herbs needed for specific non-communicable diseases. Furthermore, there is limited research on mapping the herbal plants with their respective locations, which is crucial for sustainable cultivation and conservation of these plants. Therefore, there is a need for a comprehensive AI-based solution that can accurately identify ayurvedic medical herbs needed for treatments of non-communicable diseases and map their locations for sustainable cultivation and conservation.

Existing research on the identification of medicinal herbs using image processing and machine learning has largely focused on the recognition of individual plants, with a particular emphasis on species recognition and classification. However, there is a lack of research specifically focused on the identification of Ayurvedic medical herbs that are needed for the treatment of diseases using image processing and machine learning. Furthermore, there is a lack of research that combines image processing and machine learning techniques with geographic mapping to identify the location of medicinal plants. This is a significant research gap since it would be useful to have an accurate database of the locations of Ayurvedic medical herbs for treatment purposes.

The research “A” depicts that the recognition of plants and human existence are strongly intertwined. The conventional plant identification approach operates in a convoluted manner that is unfriendly to its spread. Automatic computer recognition of plant species based on image processing is now possible thanks to the quick development of pattern recognition and computer image processing technologies. In recent years, an increasing number of academics have focused on the computer's automatic identification technology based on plant photos. We have conducted extensive study and analysis on the plant identification approach based on image processing in recent years because of this. The main technologies and steps of plant recognition are reviewed, followed by an introduction to the research significance and history of these technologies. After that, more than 30 leaf features—including 16 shape features, 11 texture features, and 4 color features—were evaluated using SVM, and 8 commonly used classifiers were described in detail. The report concludes with a finding that plant identification technologies are inadequate and a forecast for future advancement. [8]

The research “B” represents that study, they used a pattern recognition approach to demonstrate baseline automatic identification capabilities of three Ficus species based on herbarium leaf photos. To create identification models, ANN and SVM, two machine learning techniques, were

employed. Both models produced results that were satisfactory, proving their utility in identifying tasks.

According to the study described here, automated classification of certain Ficus species with similar-looking leaves is possible using leaf photos. Although the created system is not meant to take the role of human taxonomists, it may offer a quick and simple method to quickly and accurately identify plants. Since Ficus is a huge genus and species identification can be challenging, especially for non-taxonomists, we decided to focus on its species. Future expansion of the Ficus species could increase the system's resiliency. [9]

The research “C” presents evaluated automatic plant identification as a fine-grained classification challenge using the largest plant recognition datasets, up to 10,000 plant species, from the LifeCLEF [10] and CVPR-FGVC workshops.The comparison of deep neural network classifiers demonstrates the advancement in classification accuracy attained by current CNN architectures. State-of-the-art classifiers. The best model, ViT-Large/16, achieves recognition scores of 91.15% and 83.54% on the PlantCLEF 2017 and ExpertLifeCLEF 2018 test sets, respectively, before any additional post-processing like test-time augmentations and prior shift adaptation.

Prior shift adaptation: The prior shift in datasets, or the distinction in the class distribution of training and test data, is an important and pervasive phenomenon. We investigate the effects of several prior shift adaptation techniques on classification precision. Approach to fine-grained categorization using retrieval: A competitive alternative that outperforms direct classification is to train an image retrieval system and then classify the results using nearest neighbors.

Overall, there are definite benefits to employing image retrieval, such as recovering pertinent nearest-neighbor labeled samples, offering ranking class predictions, and enabling users or experts to visually confirm the species based on the k-nearest neighbors. Additionally, the retrieval approach naturally accommodates open-set recognition problems, allowing for the extension or modification of the collection of recognized classes following the training phase. The list of classes could alter, for instance, as a result of changes to biological taxonomy. In contrast to the conventional approach, which requires retraining the classification head, the introduction of new classes only requires the addition of training images with the new label. On the downside, the retrieval strategy necessitates the efficient execution of the nearest neighbor search in addition to running the deep net to extract the embedding, raising the overall complexity of the fine-grained recognition system. [11]

Features	Research A	Research B	Research C	Proposed System
Ayurvedic Plants Identification	X	X	X	✓
Usage of image processing	✓	✓	✓	✓
Plant Identification	✓	✓	✓	✓
Representative datasets	X	X	✓	✓
Noise sensitivity	X	X	✓	✓
Using of auto machine Learning	X	✓	✓	✓
Technology Used	SVM [12]	SVM/ANN [12] [13]	KNN/DNN [14]	CNN/SVM [12] [15]

Table 2: Research comparisons

Therefore, there is a need for further research that focuses specifically on the identification of Ayurvedic medical herbs for the treatment of diseases using image processing and machine learning, while also incorporating geographic mapping and addressing the challenges posed by variable plant morphology, environmental conditions, and lighting conditions.

2.RESEARCH PROBLEM

1. How to identify herbal medicinal plants through their photos?

2. How an ayurvedic medical practice can add plants to train the model without high technological knowledge?

The research problem is the lack of an efficient and accurate solution for identifying Ayurvedic medical herbs that are required for treating diseases using image processing and machine learning models. The current research gap lies in the lack of studies that combine the principles of Ayurveda with modern AI techniques to develop a robust solution for identifying medicinal plants. [3] [4] [5]

Based on the limited availability of herbal plants and medicines, combined with the lack of knowledge and resources for identifying and locating these plants, there is a need for an AI-based solution that can accurately identify Ayurvedic medical herbs used for the treatment of non-communicable diseases through image processing, while also mapping their locations. Additionally, to improve the accuracy of the system, continual learning and transfer learning techniques can be employed, and the system can be made more accessible through the use of Auto Machine Learning to enable the addition of new plants. Therefore, the research problem is to develop an AI-based solution that can accurately identify Ayurvedic medical herbs and their locations while utilizing continual learning and Auto Machine Learning (AML) techniques to improve accuracy and accessibility.

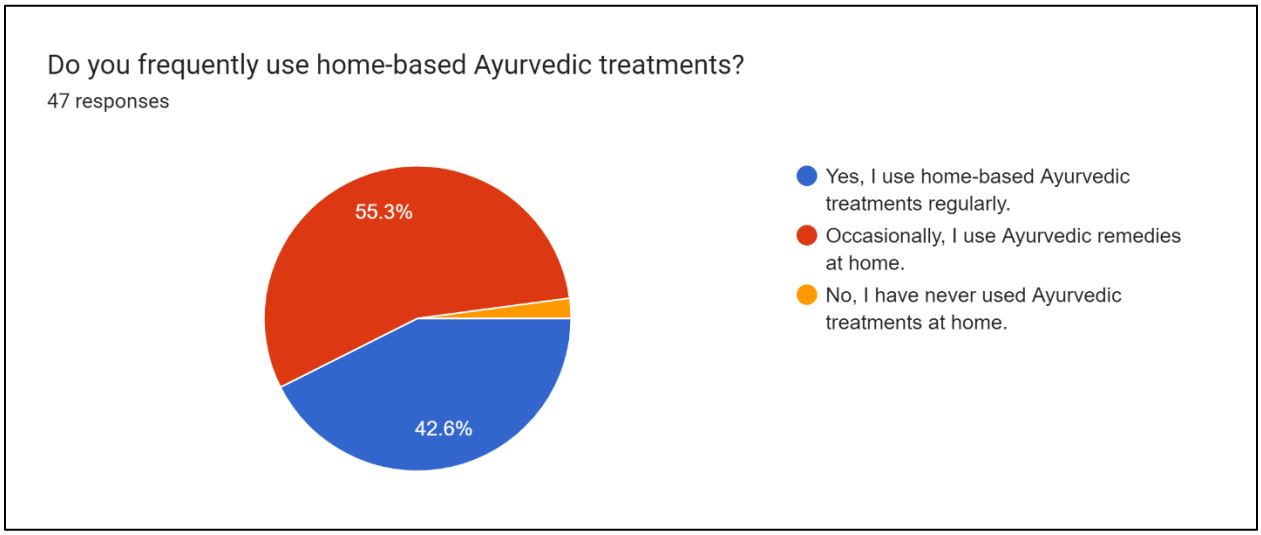

Figure 8:Survey on how frequently use ayurvedic treatments.

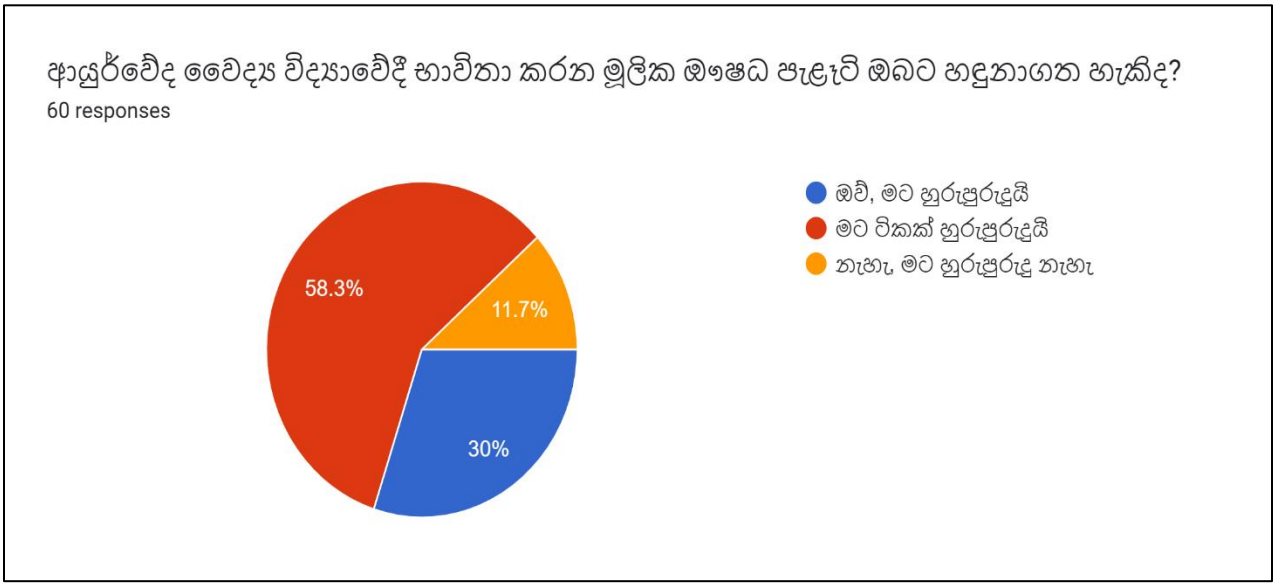

Figure 9:Survey in Sinhala on how frequently use ayurvedic treatments.

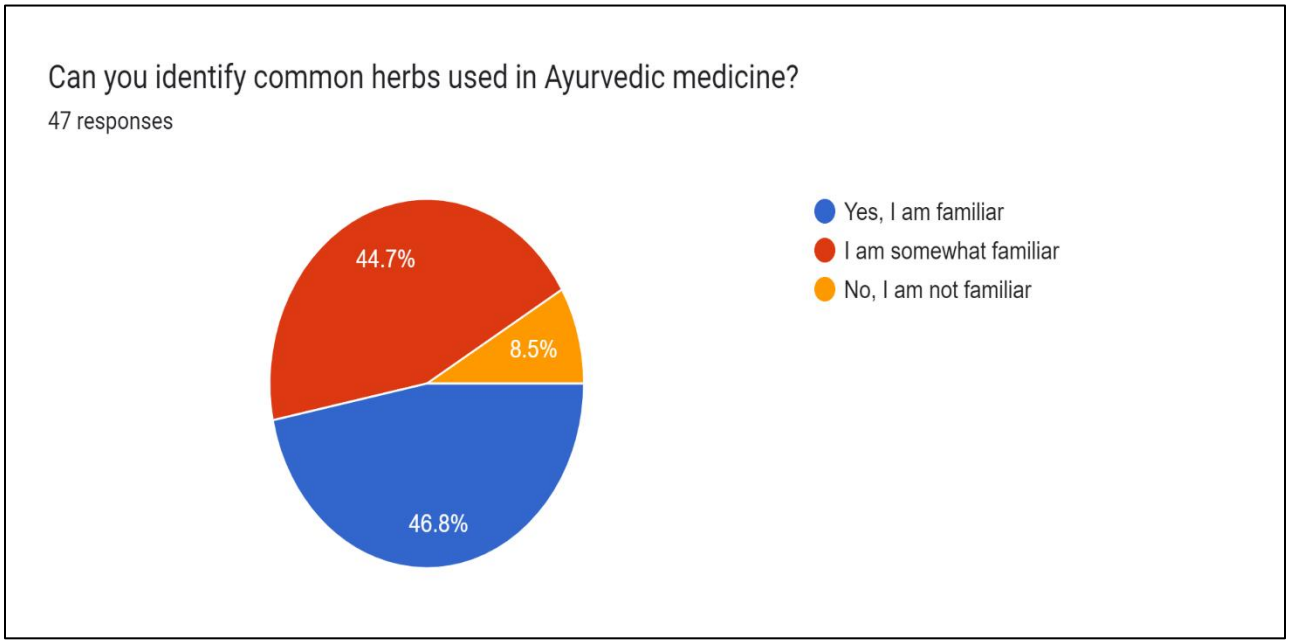

Figure 10: Survey on the capability of people to identify medicinal home plants.

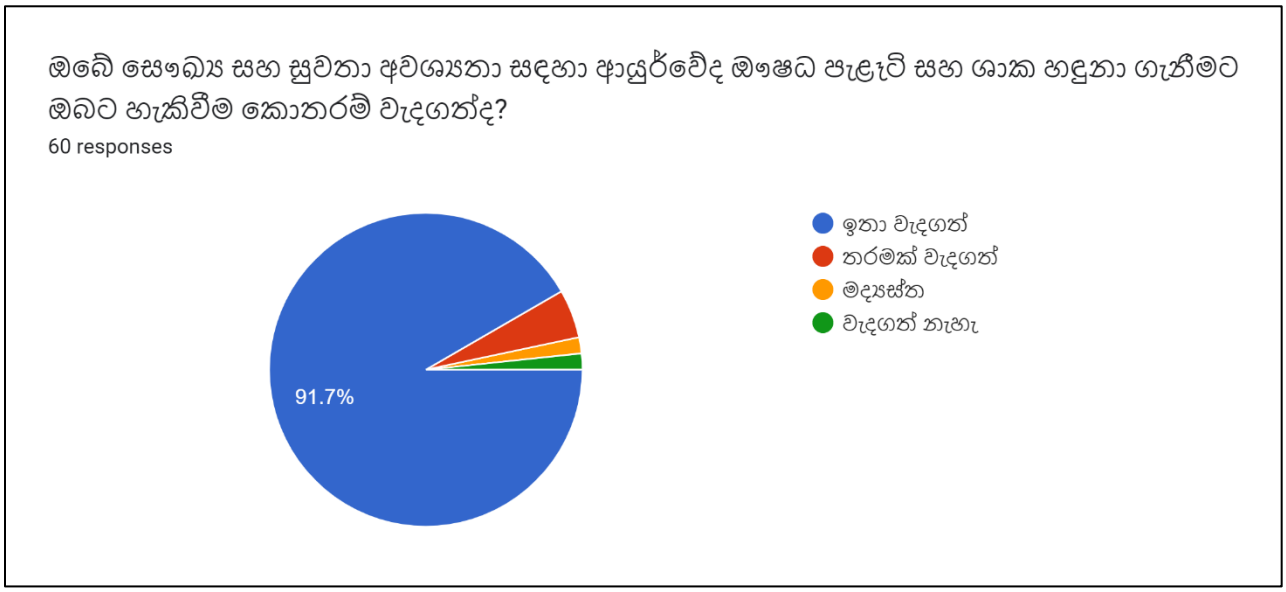

Figure 11:Survey in Sinhala on the capability of people to identify medicinal home plants

3.RESEARCH OBJECTIVES

3.1 Main Objectives

The main objective of this research is to develop a creative solution to promote a healthier lifestyle by utilizing the principles of Ayurveda by addressing the common symptoms for some diseases through identifying appropriate Ayurvedic medical herbs. Additionally, a component based on a geometry library will be implemented to map out the locations of these identified herbs which will make easier for the patient to find herbs.

To achieve this objective, the proposed solution will consist of an image processing-based component to identify the necessary herbal plants for treating these diseases. Furthermore, the solution will incorporate continual learning/transfer learning techniques to enhance the accuracy of the system, and an Auto Machine Learning component to facilitate the addition of new plants to the system. Ultimately, the objective of this research is to provide individuals with a more accessible and comprehensive approach to achieving a healthier lifestyle through Ayurveda.

3.2 Specific Objectives

Collection of images of different ayurvedic herbs and labeling them.

- Pre-processing of the collected images, such as resizing and normalization
- Splitting the data into training, validation, and testing sets.
- Training a machine learning model using pre-processed image data.
- Evaluating the model on the validation set to identify areas of improvement.
- Collecting and processing large amounts of image data from various geographical areas to map the distribution of herbal plants.
- Fine-tuning the model based on the evaluation results.
- Testing the final model on the testing set to measure its accuracy and robustness.
- Implementing the model in a software application for practical use.

As novelty improvement,

- Using continual learning/transfer learning to improve accuracy. (**Also, can incorporate Auto Machine Learning to make it easier to add new plants.)**

4.METHODOLOGY

4.1 Requirement gathering and Analysis.

- **Collecting information from Gampaha Wickramarachchi Ayurvedic University**
  To collect information on ayurveda and diseases we met **Dr. Janaki Wickramarachchi**, who is the Dean of Chikisthsaka Faculty at **Gampaha Wickramarachchi Ayurvedic University** and had conducted some online meetings conducted with her, with participation of our group members. She agreed to provide us the necessary information related to the ayurveda and the research gap which is having when connecting with modern technologies such as Artificial Intelligence and Machine Learning. She highlighted several main diseases which are fine for the research. She gave us legal approval for the continuation of the research and gave advice about the things we need to focus on in the future while continuing the project.

- **Data gathering**
  Firstly, we read a dozen published research for initial understanding and got some basic idea by reading and browsing through few articles. Our supervisors had few meetings with us to discuss the initial methods for data gathering and the external supervisor connected us with a few ayurvedic specialties and pointed out the diseases and the data we are needed continue with. In future the other necessary data and images will be collected from the University of Gampaha as necessities.

- **Conducting a survey**
  To get an idea about the knowledge of people about ayurvedic treatments and diseases and their knowledge about the connection between AI/ML with it, we have conducted a survey was conducted with both closed and open-ended questions by distributing a questionnaire.

4.2 Feasibility studies

Economic feasibility is a critical aspect of any project's success, as it determines whether or not the project is financially viable. The economic feasibility report analyzes the development costs and benefits of the project, and if a proper economic feasibility plan is not in place, the project is likely to fail. Therefore, it is crucial that the proposed system is both cost-effective and efficient in order to ensure its success. [18]

- **Scheduled feasibility.**
  Scheduled feasibility is another essential factor to consider when undertaking a project. A schedule feasibility assessment examines the timelines for the planned project, and any delays or missed deadlines can have a significant impact on the project's success. Therefore, it is vital that the proposed system completes each task within the allotted time period as specified to ensure that the project stays on schedule. [18]

- **Technical feasibility**
  Technical feasibility planning is also crucial in the development of any system. It involves evaluating the required skills and expertise necessary for mobile and web application development, as well as the ability to understand software architectures and communicate effectively with stakeholders to obtain the necessary

information. Without proper technical feasibility planning, it is unlikely that the proposed system will be successfully developed and implemented. Therefore, it is essential to have the necessary technical skills and communication abilities to move forward with the system's development. [18]

4.3 System Diagrams

4.3.1 Overall System Diagram

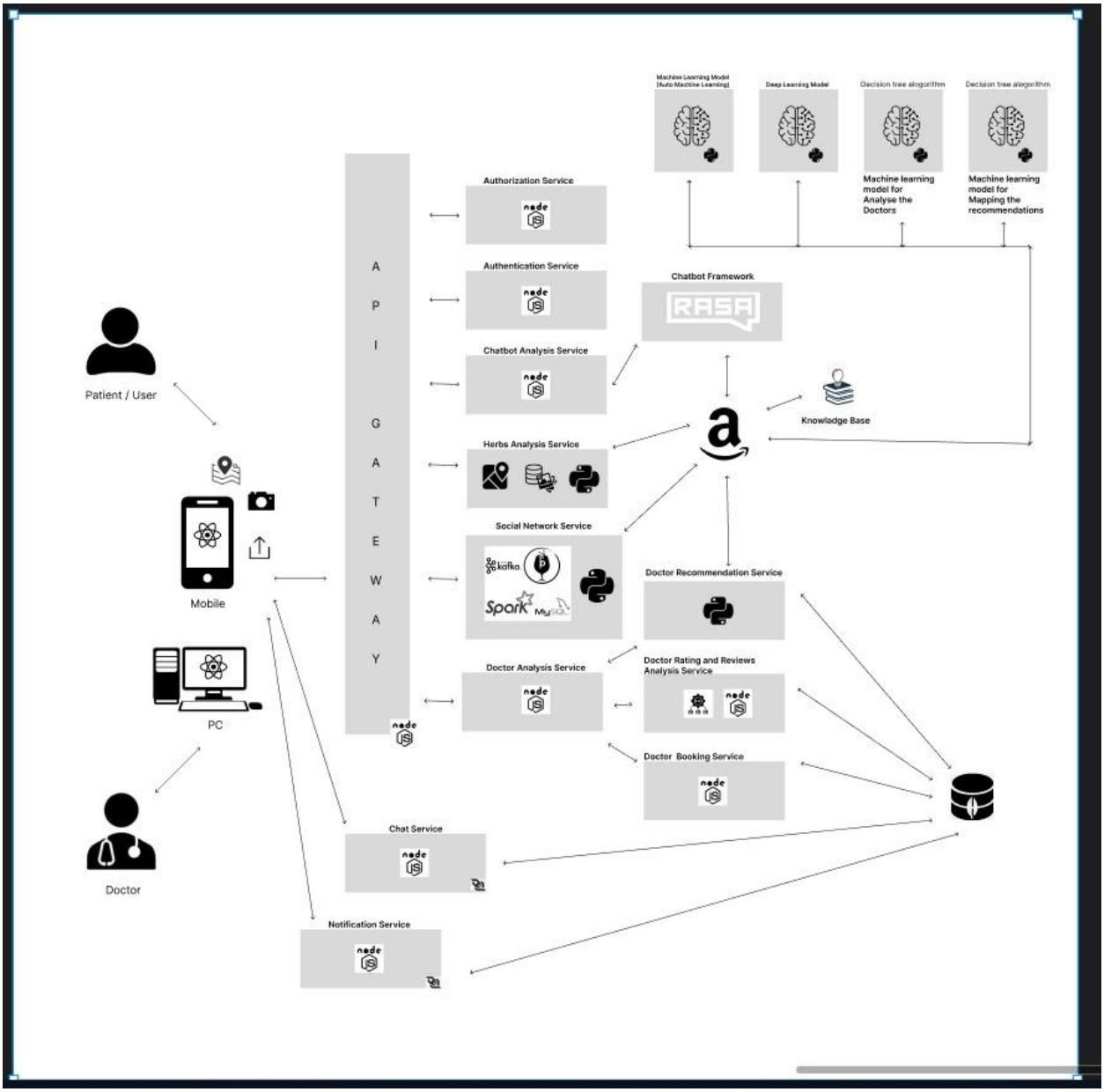

Figure 12: Overall System Architecture Diagram

4.3.2 Component System Diagram

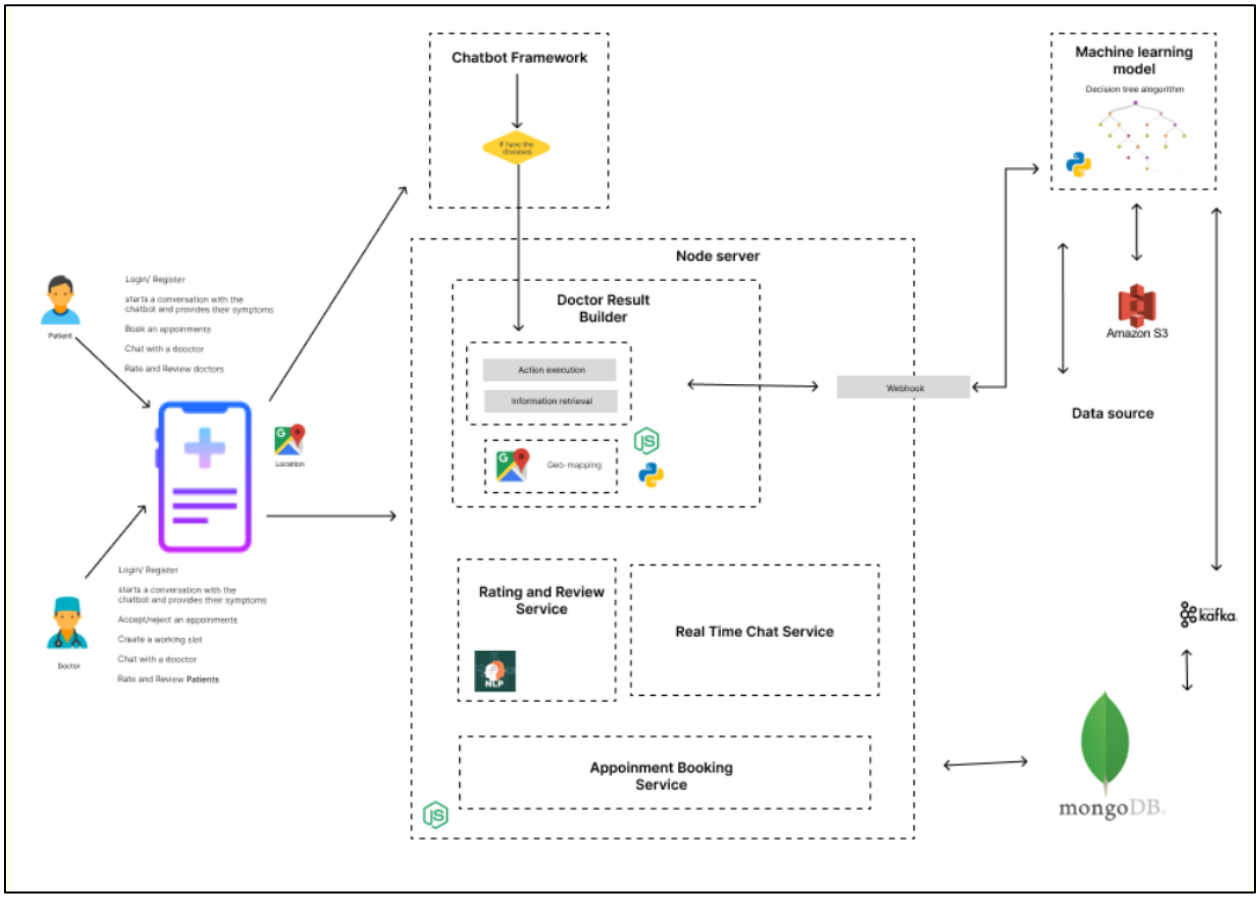

Figure 13: Individual Component Diagram

4.4 Understanding the Key Pillars of the Research Domain

4.4.1 Machine Learning

Machine learning (ML) represents a paradigm shift in problem-solving, offering a solution to complex challenges where the manual development of algorithms by human programmers is economically unfeasible. Instead, ML empowers machines to autonomously uncover algorithms without explicit human guidance. Recent advancements in the field have seen generative artificial neural networks surpassing previous methods across various domains. ML techniques find applications in diverse fields, including large language models, computer vision, speech recognition, email filtering, agriculture, and medicine, addressing tasks that would otherwise be prohibitively costly to tackle through conventional algorithmic development.

The mathematical foundation of ML is closely tied to mathematical optimization techniques, providing the essential framework for its methodologies. In parallel, data mining explores the realm of exploratory data analysis through unsupervised learning, contributing to the broader landscape of data-driven insights.

Within the realm of business problem-solving, ML assumes the guise of predictive analytics. While ML encompasses a range of approaches, not all of which rely on statistical principles, computational statistics plays a significant role in shaping the methodologies applied in the field.

The roots of machine learning trace back to the pursuit of artificial intelligence (AI). In the early stages of AI's academic development, researchers sought ways to enable machines to learn from data. They explored a variety of symbolic methods and what were initially known as "neural networks," primarily focusing on perceptrons and models that later evolved into generalized linear models in statistics. The use of probabilistic reasoning, especially in fields like automated medical diagnosis, was also prevalent.

Over time, a divergence emerged between the logical, knowledge-based AI approach and machine learning. Challenges related to data acquisition and representation plagued probabilistic systems. By the 1980s, expert systems had taken center stage in AI, sidelining statistics. Symbolic and knowledge-based learning persisted within AI, resulting in inductive logic programming, while statistical research in pattern recognition and information retrieval branched away from AI. Neural networks research was simultaneously abandoned in both AI and computer science domains. However, researchers like Hopfield, Rumelhart, and Hinton continued their work, coining the term "connectionism" and achieving significant breakthroughs, such as the reinvention of backpropagation.

The 1990s marked a renaissance for machine learning, as it emerged as an independent discipline. The field shifted its focus from the grand goal of achieving artificial intelligence to solving practical, tangible problems. It departed from the symbolic AI methods inherited from the past, embracing techniques borrowed from statistics, fuzzy logic, and probability theory. Machine learning models can be broadly categorized into three paradigms based on the nature of available signals or feedback.

Supervised learning involves the computer learning a general rule that maps inputs to outputs, guided by examples provided by a "teacher." Unsupervised learning, on the other hand, unfolds when no explicit labels are given to the learning algorithm, prompting it to autonomously uncover underlying patterns in the input data. Reinforcement learning introduces dynamic interaction with an environment, where the program strives to achieve specific objectives, akin to navigating and maximizing rewards. Each learning paradigm has its strengths and limitations, with no single algorithm capable of solving all conceivable problems.

4.4.2 Convolutional Neural Network (CNN)

CNNs are subsets of machine learning. It is one of the many varieties of artificial neural networks used for CNNs, or Convolutional Neural Networks, constitute a specific network architecture within the realm of deep learning algorithms, primarily tailored for tasks related to image recognition and the processing of pixel data. While deep learning encompasses various types of neural networks, CNNs emerge as the preferred architecture for the precise identification and recognition of objects. This inherent capability positions them as the optimal choice for tasks in the domain of computer vision (CV), particularly in applications where robust object recognition plays a pivotal role, such as in autonomous vehicles and facial recognition systems.

Deep learning algorithms extensively rely on artificial neural networks (ANNs) to model complex patterns and relationships. Among the diverse ANN types, recurrent neural networks (RNNs) stand out, specifically designed for processing sequential or time series data. RNNs find their niche in natural language processing (NLP), machine translation, speech recognition, and image captioning applications, where sequential information is paramount.

What sets CNNs apart is their versatility in extracting critical insights from both time series and image data. This versatility significantly benefits tasks involving images, including but not limited to image recognition, object classification, and pattern identification. CNNs leverage fundamental principles of linear algebra, such as matrix multiplication, to discern intricate patterns within images. Moreover, they extend their classification prowess to audio and signal data domains.

The structural arrangement of a CNN closely mimics the connectivity pattern observed in the human brain. Like the brain, CNNs comprise billions of neurons strategically organized. Remarkably, the neurons within a CNN bear resemblance to the frontal lobe of the human brain, responsible for processing visual stimuli. This specific configuration ensures comprehensive coverage of the entire visual field, eliminating the issue of fragmented image processing that plagues conventional neural networks, requiring low-resolution image inputs in disjointed segments. Consequently, CNNs exhibit superior performance when presented with image inputs, as well as when dealing with speech or audio signal inputs, surpassing the capabilities of older network architectures.

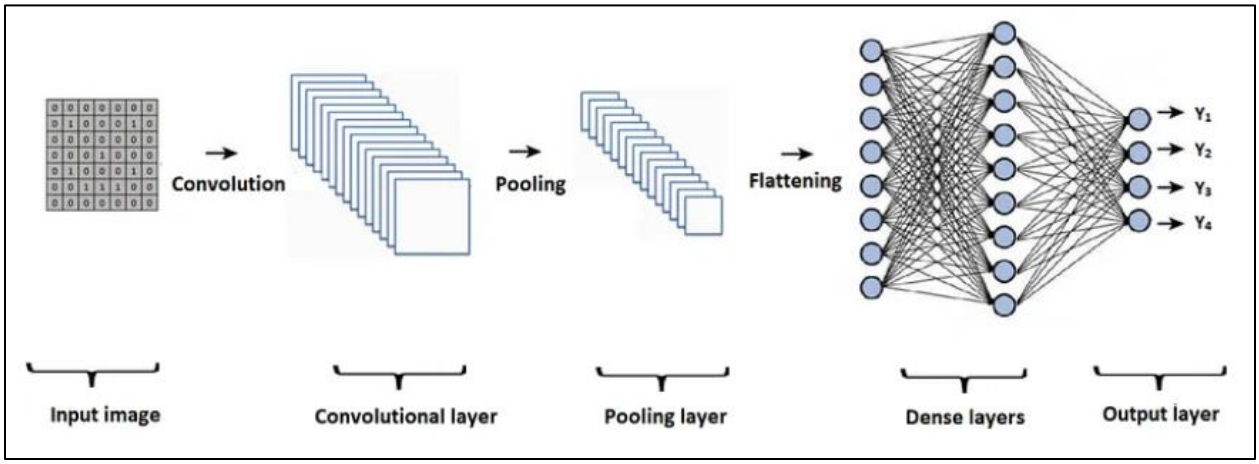

Figure 14: CNN Architecture

- **CNN stacks**

  A deep learning Convolutional Neural Network (CNN) is structured with three distinct layers: the convolutional layer, the pooling layer, and the fully connected (FC) layer. The convolutional layer is the initial layer, while the FC layer serves as the final layer in the network.

As the CNN progresses from the convolutional layer towards the FC layer, its complexity steadily increases. This incremental complexity empowers the CNN to progressively recognize larger segments and intricate characteristics within an image, ultimately leading to the complete identification of objects.

The convolutional layer serves as the cornerstone of the CNN, responsible for a significant portion of computations. This layer often includes multiple convolutional operations, and it all begins with the initial convolutional layer. During the convolution process, a kernel or filter within this layer traverses the image's receptive fields to assess the presence of specific features. Over multiple iterations, this kernel systematically explores the entire image, calculating the dot product between input pixels and the filter after each pass. The cumulative result of these computations is known as a feature map or convoluted feature, effectively converting the image into numerical values. This numerical representation enables the CNN to comprehend the image and discern relevant patterns.

The pooling layer, similar to the convolutional layer, employs a kernel or filter to traverse the input image. However, in contrast to the convolutional layer, the pooling layer serves to reduce the number of input parameters and introduces some degree of information loss. On the positive side, this layer simplifies the network's structure and enhances its efficiency.

The CNN's FC layer is where the pivotal task of image classification takes place, leveraging the extracted features from preceding layers. In this context, "fully connected" implies that all inputs or nodes from one layer establish connections with every activation unit or node in the subsequent layer.

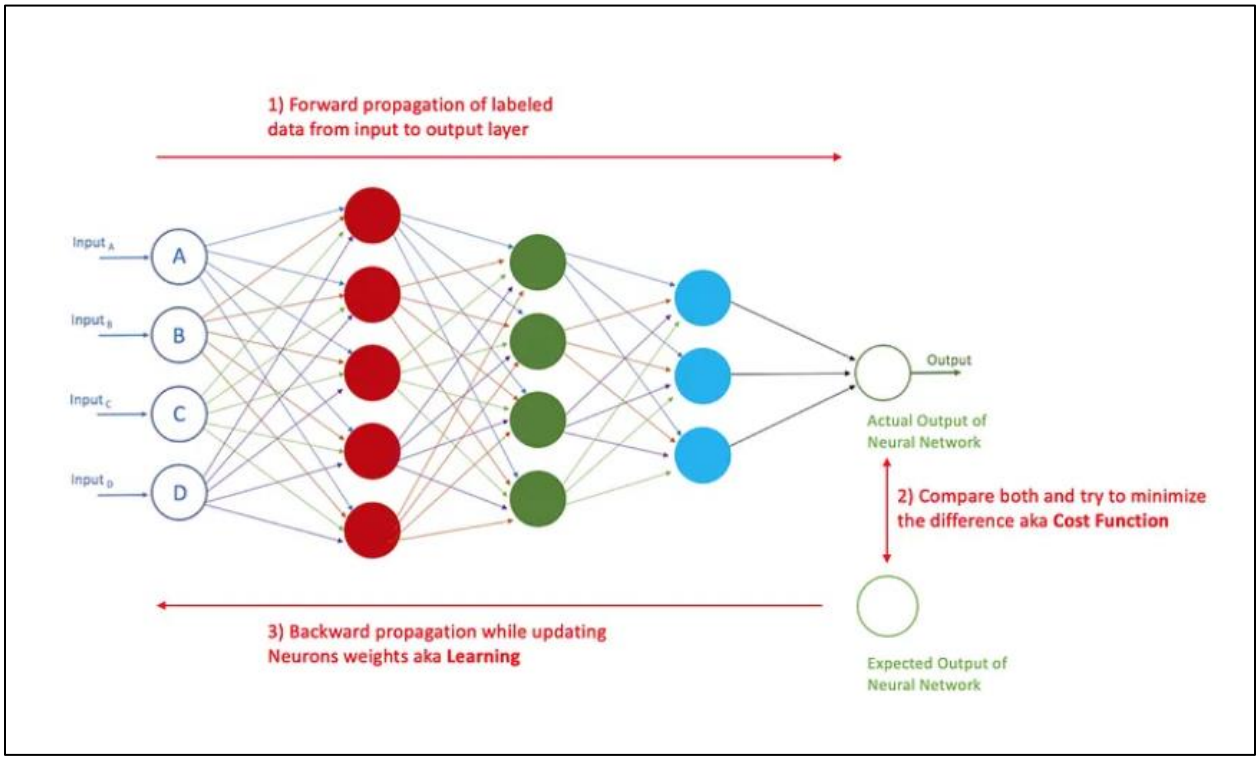

Figure 15:Process of Model Training

CNN avoids integrating all of its layers as such integration would lead to an unnecessarily dense network, potentially resulting in increased attrition, reduced output quality, and elevated computational demands.

A CNN can encompass multiple layers, and each of these layers is dedicated to learning and detecting distinct image features. In this process, a filter or kernel is applied to the input image, gradually refining and enhancing the output as it progresses through successive layers. In the earlier layers, these filters typically begin by detecting fundamental features.

With each subsequent layer, the complexity of the filters intensifies to thoroughly scrutinize and pinpoint the distinguishing characteristics that define the input object uniquely. Consequently, the output of each convolved image, representing a partially recognized image after each layer's operation, serves as the input for the subsequent layer in the hierarchy. Ultimately, in the final layer, referred to as the Fully Connected (FC) layer, the CNN achieves the identification of the image or object it represents.

The convolution process entails the application of these filters to the input image, with each filter selectively activating specific image features while transmitting its output to the subsequent layer's filter. Each layer specializes in recognizing distinct features, leading to the repetitive execution of these operations across numerous layers. The cumulative information from all the image data that has traversed through the multiple layers enables the CNN to accomplish the holistic identification of the entire object.

4.4.3 VGG based on Convolutional Neural Networks (CNN)

VGG is a standard architecture for deep Convolutional Neural Networks (CNNs) with multiple layers. "deep" refers to the number of convolutional layers, with VGG-16 and VGG-19 containing 16 and 19 convolutional layers, respectively. The VGG architecture serves as the foundation for innovative object recognition models. Designed as a deep neural network, the VGGNet outperforms benchmarks on numerous tasks and datasets beyond ImageNet. In addition, it remains one of the most prominent image recognition architectures.

4.4.3.1 VGG 19 vs VGG16

The Convolutional Neural Network model introduced by A. Zisserman and K. Simonyan, affiliated with the University of Oxford, is detailed in their study titled "Very Deep Convolutional Networks for Large-Scale Image Recognition." The VGG16 model, which stems from their work, demonstrates an impressive accuracy of nearly 92.7% in ImageNet's top five evaluation tests. ImageNet, an extensive dataset comprising over 14 million images, spans nearly a thousand distinct classes. Notably, VGG16 was among the most frequently submitted models for the ILSVRC-2014 competition. It distinguishes itself from its predecessors, like AlexNet, by substituting large kernel-sized filters with a series of 3x3 kernel-sized filters, yielding substantial improvements. The training of the VGG16 model was a protracted process spanning several weeks and leveraged Nvidia Titan Black GPUs. As previously mentioned, VGGNet-16 boasts 16 layers and possesses the capability to classify images into 1000 object categories, encompassing a wide range, including items like keyboards, animals, styluses, and mice. Additionally, the model accepts input images of dimensions 224 by 224 pixels.

The VGG19 model, also known as VGGNet-19, closely resembles the VGG16 model, with the exception that it incorporates 19 layers. The numerical labels "16" and "19" correspond to the quantity of weight layers, specifically convolutional layers, present in each model. Essentially, VGG19 encompasses three additional convolutional layers when compared to VGG16. In the subsequent section of this article, we will delve deeper into the distinctive characteristics of the VGG16 and VGG19 networks.

VGGNets are constructed on the fundamental principles of Convolutional Neural Networks (CNNs). These networks are characterized by their utilization of remarkably small convolutional filters. The VGG-16 architecture consists of a total of 13 convolutional layers and three fully connected layers. Let's take a brief overview of the architectural facets of VGG: Input: VGGNet accepts input images of size 224 by 224 pixels. To maintain uniformity for the ImageNet competition, the model's creators systematically cropped the central 224x224 portion of each image.

Convolutional Layers: VGG's convolutional layers are characterized by their small receptive field, precisely 3x3, the minimum dimension necessary to capture both vertical and horizontal information. There are 11 convolution filters applied to the input, followed by a ReLU (Rectified Linear Unit) activation function, a significant innovation introduced by AlexNet. This ReLU unit accelerates the training process considerably by efficiently handling the input. The ReLU function outputs the input value if it's positive and zero otherwise. To maintain spatial resolution post-convolution, the convolution stride is consistently set to 1 pixel (where stride indicates the number of pixel movements across the input matrix).

Hidden Layers: All hidden layers within the VGG network utilize the ReLU activation function. Notably, VGG typically avoids using Local Response Normalization due to its tendency to increase memory consumption and training time, with negligible impact on overall accuracy.

Fully-Connected Layers: The VGGNet comprises three fully connected layers. The initial two layers are equipped with 4096 channels each, while the third layer encompasses 1000 channels, one for each class within the dataset.

4.4.3.2 Usages

The VGG16 network architecture adheres to a fundamental design principle where the number of filters utilized increases twofold at each level of the convolutional layer. This design characteristic, however, comes with a notable drawback in that the VGG16 network is quite extensive, demanding a substantial amount of time for training its parameters. Due to its depth and the inclusion of fully connected layers, the VGG16 model's size exceeds 533MB, making the implementation of a VGG network a labor-intensive process. While many deep learning image classification tasks make use of the VGG16 model, smaller network architectures like GoogLeNet and SqueezeNet are often preferred due to their more manageable size and comparable performance. Nonetheless, the VGGNet serves as an excellent foundational component for learning due to its straightforward implementation.

In terms of its performance, the VGG16 model significantly surpasses previous models in the ILSVRC-2012 and ILSVRC-2013 competitions. It competes admirably in the classification task, closely rivalling the winning GoogleNet model, which achieved a 6.7% error rate, while surpassing the ILSVRC-2013 winner, Clarifai, by a substantial margin. The VGG16 model achieved an error rate of approximately 11.7% without external training data and 11.2% with such data. Notably, when considering the performance of single networks, the VGGNet-16 model emerges as the top performer with a test error rate of approximately 7.0%, surpassing a single GoogLeNet by about 0.9%.

4.4.4 Data Augmentation

The precision of a deep learning model depends on the quantity, quality, and relevance of training data. However, a scarcity of data is one of the most common obstacles in machine learning applications. In many situations, the collection of such information can be expensive and time-consuming.

Data augmentation is the process of producing additional data points from extant data in order to augment the size of an existing dataset.

Creating an extended dataset can be accomplished by making minor modifications to existing data or by employing deep learning models to generate additional data points. Companies can use data augmentation to reduce their need to acquire and prepare training data. This allows them to construct more accurate machine learning models more quickly.

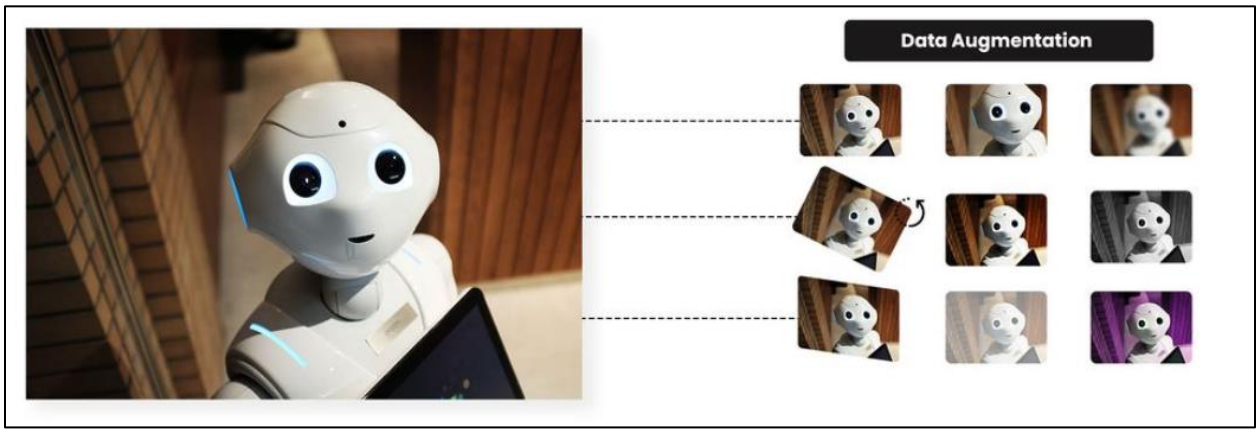

Figure 16: Data Augmentation

• Tools for Enhancing Image Data

Several Python libraries are available for improving image datasets. Libraries like Augmentor, Albumentations, Imgaug, AugLy, and Solt, along with frameworks like TensorFlow, Keras, and PyTorch, offer data augmentation capabilities. Depending on your project's requirements and convenience, you can choose any of these libraries or frameworks to work with. Explore these tools and techniques further in this section.

• Methods for Enhancing Image Data

Various techniques can be employed to manipulate image parameters such as size, color intensity, brightness, contrast, orientation, background, and more to augment existing datasets. The aforementioned open-source Python libraries make it feasible to apply these image manipulation techniques. Let's take a look at some of these techniques using an example image of a vehicle.

One augmentation technique is "flipping." By horizontally and vertically flipping the original image, we can generate two additional perspectives of the same object, expanding the dataset. For instance, a vertically flipped image of a vehicle might assist in detecting overturned vehicles during object recognition.

Similarly, rotating the image exposes the model to different object perspectives. Using a rotation angle of 180 degrees, you can also achieve a vertical flip effect.

Scaling is another data augmentation technique that allows you to create various object sizes, aiding the model in handling dimensional variations.

Additionally, you can generate more observations for the deep learning model by adjusting image parameters such as brightness, contrast, or hues.

Furthermore, with image augmentation libraries like Albumentations, you can introduce atmospheric effects like rain, clouds, or snow into an image. The resulting augmented image represents a transformed version of the original. Depending on your dataset's requirements, you can explore and implement numerous other techniques, considering that certain techniques may be better suited for specific datasets. Selecting the right set of techniques is crucial for successful data augmentation.

• How Does Data Augmentation Benefit Image Classification?

As evident from the aforementioned techniques, data augmentation enables the rapid expansion of an existing dataset by introducing more relevant observations. Deep learning models require extensive training data, and data augmentation helps these models generalize effectively across a diverse range of images with distinct labels. Consequently, it enhances the overall classification accuracy of the model. However, it's essential to be cautious about overfitting when applying data augmentation. Adding irrelevant or repetitive images can be detrimental, so it's crucial not to excessively enhance the dataset.

• Conclusion and Future Prospects for Data Enhancement

In recent years, data augmentation has shown tangible benefits in various domains, including image restoration, medical imaging for disease identification, environmental monitoring, autonomous vehicle control, and traffic management, among others. With continuous advancements in computer vision, data augmentation will remain a valuable tool for advanced data analytics, supporting tasks related to identification, classification, and prediction.

4.4.5 Image Processing

• Image Processing Overview

Image processing refers to the procedure of converting an image into a digital format and applying specific operations to extract useful information from it. In this context, image processing systems typically treat all images as 2D signals when implementing signal processing techniques.

• Types of Image Processing

There are five primary categories of image processing:

1. Visualization: Detecting previously invisible objects within an image.

2. Object Identification: Distinguishing and recognizing objects within an image.

3. Sharpening and Restoration: Enhancing an image's quality compared to the original.

4. Pattern Recognition: Analyzing patterns in the surroundings of objects within an image.

5. Retrieval: Searching and retrieving digital images similar to a reference image from a vast database.

• Hardware for Image Processing

Image processing systems rely on general-purpose computers, which can range from personal computers to supercomputers. In specialized applications, custom-built computers may be used to achieve specific performance requirements.

• Specialized Image Processing Hardware

Specialized hardware includes components like digitizers and hardware capable of performing fundamental operations, such as Arithmetic Logic Units (ALUs) that can execute simultaneous arithmetic and logical operations on entire images.

• Key Components

1. Large Storage: Essential for storing image data, including short-term storage, online storage for quick access, and archival storage for infrequent access.

2. Image Sensors: Devices like CCD and CMOS sensors used in digital cameras to capture and convert light into electrical signals.

3. Visual Display: Necessary for viewing processed images.

4. Printing Equipment: Instruments such as laser printers, film cameras, and inkjet printers for recording images.

5. Networking: Required for transmitting visual data through networked computers, with bandwidth being a critical factor due to the substantial data requirements of image processing applications.

• Fundamental Image Processing Steps

1. Image Collection: The initial stage involving image retrieval from a source, often hardware-based.

2. Image Enhancement: The process of highlighting obscured features of interest within an image, which may involve adjustments to luminance, contrast, and more.

3. Image Restoration: Enhancing the appearance of an image using mathematical or probabilistic models.

4. Color Image Processing: Techniques for processing color images, especially relevant due to their widespread use on the internet.

5. Multiresolution and Wavelet Processing: The use of wavelets to represent images at varying resolutions, facilitating data compression and pyramidal representation.

6. Compression: Reducing storage requirements for images, particularly for internet use.

7. Morphological Operations: Operations that transform images based on their structures.

8. Segmentation: The challenging process of dividing an image into its individual components or objects.

9. Description and Representation: Representing segmented regions with suitable descriptors for further computer processing.

• Deconvolution Challenges in Image Processing

Blind image deconvolution involves restoring a clear image from a distorted and noisy one without precise knowledge of the blurring process. This blurring can result from various factors like defocus, camera movement, or scene motion. Image deconvolution is particularly challenging when dealing with motion distortion or out-of-focus objects.

• Importance of Phase in Image Processing

Phase in image processing holds information about feature positions. Combining phase-only and magnitude-only images cannot recreate the original image. Instead, multiplying these images in the Fourier domain and then reversing the transformation is necessary to recover the original. Phase information is crucial for defining timing or relative positions in signals, often more significant than absolute phases in various applications.

4.5 Specific Methodology

The methodology to achieve these objectives involves:

- Collect a dataset of plant images: You will need a dataset of labeled images of plants for training your machine learning model.
- Preprocess the data: You will need to preprocess the data by resizing the images, normalizing the pixel values, and splitting the dataset into training and validation sets.
- Feature extraction: You will need to extract features from the images that can be used to train your machine learning model. There are several feature extraction techniques available, such as Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and Convolutional Neural Networks (CNNs).
- Train the machine learning model: Once you have extracted the features, you can train your machine learning model. There are several algorithms you can use for classification, including convolutional Neural Networks.
- Test and Evaluate: After training the model, you will need to test it using a separate dataset and evaluate its performance. You can use metrics such as accuracy, precision, recall, and F1 score to evaluate the performance of the model.
- Deploy the system: Finally, you can deploy the system and make it accessible to users.

This element has been incorporated into the system to enhance user interactions related to herbal remedies. Its primary aim is to assist users in comprehending and identifying Ayurvedic herbs. This feature enables users to obtain detailed information about herbs recommended by the chatbot, empowering them to explore these herbs independently. When users receive herbal suggestions from the chatbot, they can access an interface that provides extensive details about these recommended herbs. Users can also manually search for herbs based on the information provided. Additionally, a unique functionality allows users to capture images of herbs through their device's camera if they encounter a specific herb that is challenging to identify. This capability is implemented through a combination of backend and frontend functionalities.

On the backend side, the Fast API framework is employed to create an interface that accepts image uploads and initiates the herb prediction process. Uploaded images are processed, and a trained model is employed to predict the herb that best matches the user's input. This model utilizes a pre-trained VGG19 architecture that has been fine-tuned using a dataset comprising images of herbs. The integration with Firebase ensures that comprehensive herb details, including images and additional information, are stored and retrieved seamlessly. Users can access this comprehensive herb information stored in the database, aiding them in identifying the encountered herbs.

On the frontend aspect, users interact with the image upload interface. Users have the option to upload images of herbs they've found, and the system provides predictions based on these images. If a user cannot find an exact match for the identified herb, they can use the camera feature to capture images for analysis. Additionally, users can access a dedicated image upload area, offering a straightforward and intuitive method for submitting herb images. The backend model processes these uploaded images, generating predictions that help users identify the herbs they've discovered. This interactive feature bridges the gap between the chatbot's recommendations and user-driven exploration, fostering an empowering and educational experience.

1) Model Training Phase:

- Dataset Creation: A diverse and comprehensive dataset of Ayurvedic herb images is curated, encompassing various herb species and perspectives.

- Data Preprocessing: Images are standardized by resizing them to a standard dimension (e.g., 224x224 pixels) while maintaining the aspect ratio. Pixel values are normalized to a specific range (usually [0, 1]).

- Dataset Split: The dataset is divided into training and validation sets for effective model evaluation.

- Model Architecture: A pre-trained neural network architecture, such as VGG19, is selected as the base model due to its robust feature extraction capabilities.

- Transfer Learning: The pre-trained model's convolutional base serves as a feature extractor, and new dense layers are added on top for classification.

- Fine-tuning and Training: The entire model is fine-tuned on the Ayurvedic herb dataset. Convolutional layer weights are frozen, while dense layer weights are updated during training.

- Validation and Hyperparameter Tuning: Model performance is evaluated on the validation dataset, and hyperparameters like learning rate, batch size, and epochs are fine-tuned for optimal results.

2) Herb Prediction (Inference) Phase:

- Image Preprocessing: When a user uploads an herb image, it undergoes preprocessing steps to align with the model's requirements.

- Model Inference: The pre-trained and fine-tuned herb identification model predicts the herb species from the preprocessed image, providing a probability distribution over herb classes.

- Class Selection: The herb class with the highest predicted probability is chosen as the final prediction, providing the identified herb's name.

- Herb Details Retrieval: Using the predicted herb class label, detailed information about the identified herb is retrieved from the Firebase database, including textual details and related images.

- Presentation to Users: The identified herb and its associated data are displayed through the frontend interface, allowing users to explore the details and properties of the recommended herb based on the provided information.

This research component follows a structured approach encompassing the development of the backend herb identification model, the creation of a user-friendly image upload interface, seamless integration with Firebase for herb information storage, and the provision of comprehensive herb details to users. Evaluation of this component involves assessing the accuracy of herb predictions and measuring user satisfaction with the feature.

4.6 Commercialization Aspects of the System

This study offers a creative answer in the form of a potential filling of one of the research gaps that were discovered by our research. Everyone who has a stake in the matter, including patients, ayurveda practitioners working in the ayurvedic industry, and any other parties that are pertinent, will be informed about the application known as "AYUR MANA." The Gampaha Wickramarachchi Ayurvedic University holds events on a regular basis for university students and practitioners. Because the institution has its own medicinal supply systems across the nation, these events also feature the presentation of products in an effort to boost sales in the local community.

In addition to that, promotion of AYUR MANA will take place on the official website of the Gampaha Wickramarachchi Ayurvedic University. There is already an application available in the Google Play store that customers who are interested in downloading can use on their portable technology.

Patients, ayurvedic practitioners, and members of the general public who have an interest in ayurveda are the primary target stakeholders that this system intends to accommodate.

4.7 Consideration of the aspect of the System

During the process of coding each individual component, coding standards were adhered to.

4.7.1 Social Aspects

The smartphone application known as 'AYUR MANA' can be utilized by anyone with varying degrees of familiarity and expertise with technology. The early diagnosis and management of diseases and pests will help save enormous economic losses, which in turn will contribute to the reduction of global poverty and the improvement of people's access to nutritious food.

The process of early infestation control has been made more difficult since there are not enough tools that are based on information and communications technology to assist in real-time collaboration between producers and researchers. Therefore, by the use of this cutting-edge technology, we can provide a way for early detection and quick communication while also ensuring the education of the farmers, which will ultimately assist in the survival of the stakeholders for sustainable ayurvedic solutions.

4.7.2 Security Aspects

The smartphone application known as "AYUR MANA" provides users with the ability to safeguard the confidentiality of their private information, detection details, and user authentication. Concerns regarding the level of security were taken into account right from the beginning of the page. For the purpose of ensuring the user's authenticity, JWT tokens were utilized. While communicating over the network, several ways for encrypting data were utilized in order to circumvent the security checks. Requests for user rights were made in order to improve the system's integrity and security.

4.7.3 Ethical Aspects

The design of the system takes into consideration several ethical factors. There is no upper age limit for users of this platform.

Children are able to successfully use this program as well. The culture is respected, and the application has not yielded any negative feedback in that regard.

5. IMPLEMENTATION AND TESTING

5.1 Implementation

The research project offers a comprehensive suite of software solutions encompassing both web and mobile applications, designed to address critical aspects of image identification, classification, and progression level assessment in herb plants related to ayurveda. These applications are collectively known as "AYUR MANA."

1. AYUR MANA Mobile Application:

- Purpose: The mobile application, "AYUR MANA," serves as a versatile tool for the identification, classification, and evaluation of disease progression in coconut palms.
- Functionality:

✓ Disease Identification: AYUR MANA allows users to identify plant leaves affecting various diseases,

providing valuable insights into potential threats to the humans.

- ✓ Classification: The application enables the classification of identified leaves, aiding in the categorization and differentiation.
- ✓ Progression Level Calculation: Users can assess the severity and progression level of diseases, facilitating timely intervention and management strategies.
- Implementation Technology: The front end of the mobile application is implemented using React Native, a cross-platform framework known for its efficiency and ability to deliver consistent user experiences across different mobile platforms.

2. AYUR MANA Web Application:

- Purpose: The web application is designed to cater to the specific needs of researchers at the Gampaha Wikramarachchi Ayurvedic University, offering continuous monitoring capabilities for disease-infected patients across the country.
- Functionality:
- ✓ Monitoring: practitioners can monitor the status of plant herbs in real-time, facilitating data-driven decision-making.
- Data Analysis: The web application supports the analysis of leaves and patterns, aiding researchers in gaining valuable insights.
- Implementation Technology:
- ✓ Frontend Development: The frontend of the web application is implemented using React.js, a popular JavaScript library known for its flexibility and performance in creating dynamic user interfaces.
- ✓ Backend Development: Node.js is employed for the backend development of the web application. Node.js is well-suited for handling real-time data updates and providing a robust server-side environment.

In summary, AYUR MANA is a comprehensive software solution comprising a mobile application for on-the-ground plant leaf identification and assessment and a web application for remote monitoring and data analysis. The technologies chosen for implementation, such as React Native, React.js, and Node.js, ensure efficiency, cross-platform compatibility, and real-time data handling, making AYUR MANA a powerful tool for researchers and stakeholders involved in the management and preservation of Ayurvedic Medicine palm lands.

5.1.1 Preprocessing, Augmentation and Model Implementation

The code imports a set of Python libraries and modules that are commonly used for tasks related to deep learning, image processing, file operations, time management, and data visualization. These libraries and modules will likely be used throughout the script for different tasks and functionalities.

```
import pyrebase
import time
import os
import numpy as np
import shutil
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19
from glob import glob
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
import matplotlib.pyplot as plt
```

Figure 17: Code Snippet to Import Libraries

1) import pyrebase: This line imports the "pyrebase" library, which is often used to interact with the Firebase platform in Python. Firebase is a popular backend service that provides various features, including real-time databases, authentication, and cloud storage.

2) import time: The "time" module is a standard Python library that allows you to work with time-related functions, such as measuring code execution time or adding delays in your code.

3) import os: The "os" module provides functions for interacting with the operating system, including tasks like reading files, creating directories, and more.

4) import numpy as np: This imports the "numpy" library, which is widely used for numerical and array operations in Python. The alias "np" is a common convention when importing numpy.

5) import shutil: The "shutil" module is used for file operations, such as copying, moving, and deleting files and directories.

6) from tensorflow.keras.layers import Input, Lambda, Dense, Flatten: These lines import specific layers from the TensorFlow Keras library. These layers are commonly used when building neural network models.

7) from tensorflow.keras.models import Model: This line imports the "Model" class from TensorFlow Keras, which is used to create and manipulate deep learning models.

8) from tensorflow.keras.applications.vgg19 import VGG19: This imports the VGG19 model architecture from TensorFlow Keras applications. VGG19 is a popular deep neural network architecture often used for image-related tasks.

9) from glob import glob: The "glob" function is used for searching and retrieving files matching a specified pattern in a directory. It is often used to load datasets of images.

10) from tensorflow.keras.preprocessing.image import ImageDataGenerator: This line imports the "ImageDataGenerator" class, which is used to generate augmented versions of images, typically used for data augmentation during training deep learning models.

11) from PIL import Image: The "PIL" library (Python Imaging Library) is used for opening, manipulating, and saving various image file formats.

12) import matplotlib.pyplot as plt: This imports the "matplotlib" library, specifically the "pyplot" module, which is used for creating various types of plots and charts, including displaying images

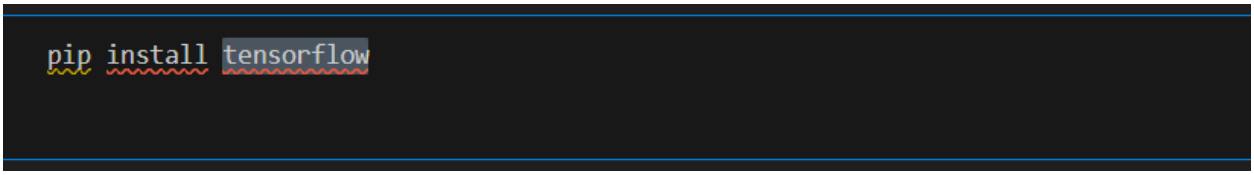

Figure 18: Code Snippet to install TensorFlow.

When you run pip install tensorflow, the pip package manager will connect to the Python Package Index (PyPI), download the TensorFlow package, and install it on your system. This enables you to use TensorFlow in your Python projects and scripts, allowing you to leverage its powerful capabilities for machine learning and deep learning tasks.

Figure 19: Code Snippet to install NumPy.

NumPy (Numerical Python) is a fundamental Python library for numerical and array operations. It provides support for creating, manipulating, and performing operations on arrays and matrices of data. NumPy is widely used in scientific computing, data analysis, and machine learning because of its efficient data structures and mathematical functions.

When you run pip install numpy, the pip package manager will connect to the Python Package Index (PyPI), download the NumPy package, and install it on your system. This enables you to use NumPy in your Python projects and scripts, allowing you to work with numerical data and perform various mathematical and data manipulation tasks efficiently.

```
pip install fastapi
```

Figure 20: Code Snippet to install Fast API

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints. It is designed for building RESTful APIs quickly and efficiently. FastAPI provides features like automatic validation of request and response data, automatic generation of API documentation (using tools like Swagger and ReDoc), and support for asynchronous programming, making it a popular choice for building web APIs.

When you run pip install fastapi, the pip package manager will connect to the Python Package Index (PyPI), download the FastAPI package, and install it on your system. This enables you to use FastAPI in your Python projects, allowing you to create robust and high-performance web APIs with minimal code.

```
pip install unicorn
```

Figure 21: Code Snippet to install Unicorn.

The command pip install unicorn is used to install the Unicorn library using the Python package manager, pip. Let's break down the meaning of this command:

pip: pip is a package manager for Python, which is used to install and manage Python packages (libraries and modules).

install: This is the command to tell pip that you want to install a package.

unicorn: This is the name of the package you want to install. In this case, you are installing Unicorn.

```
pip install matplotlib
```

Figure 22: Code Snippet to install matplotlib.

Matplotlib is a widely used Python library for creating high-quality, static, animated, and interactive plots and charts. It provides a flexible and user-friendly interface for generating a wide range of 2D and 3D plots, making it an essential tool for data visualization in scientific computing, data analysis, and various fields of research and industry.

When you run pip install matplotlib, the pip package manager will connect to the Python Package Index (PyPI), download the Matplotlib package, and install it on your system. This enables you to use Matplotlib in your Python projects, allowing you to create a wide variety of plots and visualizations to convey data and insights effectively.

```
pip install pyrebase5
```

Figure 23: Code Snippet to install pyrebase

When you run pip install pyrebase5, the pip package manager will connect to the Python Package Index (PyPI), download the "pyrebase5" package, and install it on your system. This allows you to use "pyrebase5" in your Python projects, enabling interactions with Firebase services and integration of Firebase functionality into your applications.

```
firebaseConfig = {
  "apiKey": "AIzaSyAOoxCKY--vhHPXTi5VBClBgzJpSghElUk",
  "authDomain": "aurveda-3e60f.firebaseapp.com",
  "databaseURL": "https://aurveda-3e60f-default-rtdb.firebaseio.com",
  "projectId": "aurveda-3e60f",
  "storageBucket": "aurveda-3e60f.appspot.com",
  "messagingSenderId": "619349165409",
  "appId": "1:619349165409:web:ca7a53485ee82faa1af120",
  "measurementId": "G-LFBLN9CB2F",
  "serviceAccount":"Firebase_Service_Account_Keys.json"
}

firebase = pyrebase.initialize_app(firebaseConfig)
storage = firebase.storage()
db = firebase.database()
```

Figure 24: Code Snippet to firebase Configuration

1. Firebase Configuration:

firebaseConfig is a Python dictionary that contains the configuration settings required to connect to various Firebase services. These settings include:

"apiKey": An API key used for authentication and access control.

"authDomain": The authentication domain or URL of the Firebase project.

"databaseURL": The URL of the Firebase Realtime Database.

"projectId": The unique identifier of the Firebase project.

"storageBucket": The URL of the Firebase Storage bucket.

"messagingSenderId": The sender ID for Firebase Cloud Messaging (FCM).

"appId": The unique application ID for the Firebase project.

"measurementId": The measurement ID for analytics (if enabled).

"serviceAccount": The path to a service account JSON file (Firebase Service Account Keys) that contains credentials for server-to-server interactions.

2. Initializing Firebase:

firebase = pyrebase.initialize_app(firebaseConfig) initializes a connection to Firebase using the provided configuration settings (firebaseConfig). This line creates a Firebase app instance (firebase) that can be used to access various Firebase services.

Firebase Storage and Database:

storage = firebase.storage() initializes a connection to Firebase Storage, allowing you to interact with the cloud storage service. You can use storage to upload, download, and manage files in Firebase Storage.

db = firebase.database() initializes a connection to the Firebase Realtime Database. You can use db to read and write data to the Firebase database in real-time.

Overall, this code sets up a Python environment to interact with Firebase services, such as Firebase Storage and the Realtime Database, using the Pyrebase library. Once initialized, you can use storage and db to perform various operations related to cloud storage and database interactions within your Python application.

```
storage.child("Main_Data").child("Alpinia Galanga (Rasna)/AG-S-001.jpg").get_url(None)
```

```
'https://firebasestorage.googleapis.com/v0/b/aurveda-3e60f.appspot.com/o/Main_Data%2FAlpinia%20Galanga%20%28Rasna%29%2FAG-S-001.jpg?alt=media'
```

Figure 25: Code Snippet to retrive data from Firebase

It will retrieve the URL of the file "AG-S-001.jpg" located in the "Alpinia Galanga (Rasna)" directory within the "Main_Data" directory in Firebase Storage. The URL can then be used to access and download the file from the Firebase Storage location.

```
herb_name = "test"


db.child("herbs").child(herb_name).child("img").set("url")

url'


db.child("herbs").child(herb_name).child("info").set("info about it")
```

Figure 26: Creating Real Time Database in Fire Base

This code is creating or updating a Firebase Realtime Database entry for an herb with the name "test." It's setting two pieces of information: the URL of an image associated with the herb under the "img" sub-node and some general information about the herb under the "info" sub-node. This structure allows you to organize and store information about different herbs in the database with each herb having its own entry under the "herbs" node.

```
filesx = storage.list_files()
main_file = "main"
for file in filesx:
    file_names = file.name.split('/')[1]
    # print(file_names)
    if main_file != file_names:
        print(file_names)
        os.mkdir("./new_data/"+file_names)
        main_file = file_names
    if file.name.endswith('.jpg'):
        file_img = file.name.split('/')[-1]
        file.download_to_filename("new_data/"+file_names+"/"+file_img)
```

Figure 27: Code Snippet to list from firebase storage

Here it is working with Firebase Storage using the Pyrebase library to list files, organize them into directories, and download JPEG files to a local directory. Let's break down what this code is doing:

1) filesx = storage.list_files(): This line retrieves a list of files and objects stored in Firebase Storage. The list_files() method allows you to access the files stored in a specific storage bucket.

2) main_file = "main": This initializes a variable main_file with the string "main." This variable will be used

to keep track of the current main directory being processed.

3) for file in filesx:: This initiates a loop to iterate through the list of files obtained from Firebase Storage.

4) file_names = file.name.split('/')[1]: This line extracts the name of the file from its full path by splitting the path using '/' as the separator. It takes the second element (index 1) of the resulting list as the file_names, which should correspond to the name of the current directory.

5) if main_file != file_names:: This condition checks if the main_file is not equal to the file_names. If they are different, it means a new main directory is encountered in the list of files.

6) os.mkdir("./new_data/"+file_names): If a new main directory is encountered, this line creates a new directory with the name file_names in the local "./new_data/" directory.

7) main_file = file_names: It updates the main_file variable to the current file_names value.

8) if file.name.endswith('.jpg'):: This condition checks if the file has a ".jpg" extension.

9) file_img = file.name.split('/')[-1]: It extracts the filename from the full path by splitting the path using '/' as the separator and taking the last element (index -1) of the resulting list.

10) file.download_to_filename("new_data/"+file_names+"/"+file_img): If the file has a ".jpg" extension, it downloads the file to the local directory "./new_data/" under the appropriate subdirectory (based on file_names) with the same filename.

In summary, this lists files from Firebase Storage, organizes them into directories based on their names, and downloads JPEG files to a local directory structure. It ensures that files with the same file_names are grouped in the same local subdirectory.

```
def is_folder_empty(folder_path):
    items = os.listdir(folder_path)
    if len(items) == 0:
        return True
    else:
        return False
```

Figure 28: Code Snippet to check a specified folder

The provided Python function `is_folder_empty` checks whether a specified folder (directory) is empty or not. Here's an explanation of how this function works:

1. `def is_folder_empty(folder_path):`: This line defines a Python function called `is_folder_empty` that takes one argument, `folder_path`, which should be the path to the folder you want to check.

2. `items = os.listdir(folder_path)`: Inside the function, it uses the `os.listdir()` function to obtain a list of items (files and subdirectories) contained within the specified `folder_path`. This list is stored in the `items` variable.

3. `if len(items) == 0:`: This condition checks if the length (number of items) in the `items` list is equal to zero. If the list is empty, it means there are no files or subdirectories in the specified folder.

4. `return True`: If the folder is empty, the function returns `True`, indicating that the folder is indeed empty.

5. `else:`: If the condition in step 3 is not met, meaning there are items in the folder, the function proceeds to the `else` block.

6. `return False`: In the `else` block, the function returns `False`, indicating that the folder is not empty.

So, when you call this function with a folder path as an argument, it will return `True` if the folder is empty and `False` if it contains files or subdirectories.

```python
def add_data():
    rootdir= 'new_data'
    sub_class = os.listdir(rootdir)

    for classes in sub_class:
        try:
            os.mkdir("./Dataset/train/"+classes)
            os.mkdir("./Dataset/test/"+classes)
            os.mkdir("./Dataset/resize train/"+classes)
            os.mkdir("./Dataset/resize test/"+classes)
            print("Data Added")
        except:
            print("Already in the system")

    for classes in sub_class:
        print(classes)
        if is_folder_empty("Dataset/train/"+classes):
            print(f"The folder '{classes}' is empty.")
            path_to_class = rootdir +'/'+classes
            allimgs = os.listdir(path_to_class)
            np.random.shuffle(allimgs)
            test_ratio = 0.25
            train_FileNames, test_FileNames = np.split(np.array(allimgs),[int(len(allimgs)* (1 - test_ratio))])

            train_FileNames = [path_to_class+'/'+ name for name in train_FileNames.tolist()]
            test_FileNames = [path_to_class+'/' + name for name in test_FileNames.tolist()]

            for name in train_FileNames:
                shutil.copy( name ,'./Dataset/train/' + classes)

            for name in test_FileNames:
                shutil.copy(name,'./Dataset/test/' + classes)
        else:
            print(f"The folder '{classes}' is not empty.")
```

Figure 29: Code Snippet to splitting data into training and test data

The `add_data` function appears to perform the following tasks related to organizing and splitting data into training and testing datasets for multiple classes:

1. Creating Directories-

- It first lists the subdirectories (classes) within the "new_data" directory using `os.listdir(rootdir)`.

- For each class, it attempts to create several subdirectories inside the "Dataset" directory:

  - "./Dataset/train/" + class_name

  - "./Dataset/test/" + class_name

  - "./Dataset/resize train/" + class_name

  - "./Dataset/resize test/" + class_nam

- If the subdirectories already exist, it prints "Already in the system." If they don't exist, it prints "Data Added."

2. Data Splitting-

- For each class, it checks if the "Dataset/train/class_name" directory is empty using the `is_folder_empty` function. If it's empty, it proceeds to split the data into training and testing sets.

- It shuffles the list of image filenames within the class directory and defines a test_ratio (e.g., 0.25, indicating a 25% test set).

- It then splits the list of image filenames into two lists: `train_FileNames` and `test_FileNames`, according to the specified test ratio.

- For each filename in `train_FileNames`, it copies the image file to "./Dataset/train/class_name."

- For each filename in `test_FileNames`, it copies the image file to "./Dataset/test/class_name."

3. Handling Non-Empty Directories-

- If the "Dataset/train/class_name" directory is not empty (indicating that data splitting may have already occurred), it prints "The folder 'class_name' is not empty."

This code is essentially organizing and splitting data into training and testing datasets for each class, assuming that each class's data is stored in separate subdirectories under the "new_data" directory. It also handles cases where the data splitting has already been performed for some classes.

```python
def data_pre_procesing():
    IMG_SIZE = (227,227)
    src_train_path = './Dataset/train/'
    dst_train_path = './Dataset/resize train/'
    src_test_path = './Dataset/test/'
    dst_test_path = './Dataset/resize test/'

    train_dir = os.listdir(src_train_path)
    test_dir = os.listdir(src_test_path)

    for img_list in train_dir:

        x = src_train_path+img_list

        img_folders = os.listdir(x)

        for img_name in img_folders:
            img_dst_train_path = dst_train_path + '/'+ img_list +'/' + img_name
            img_src_train_path = src_train_path + '/' + img_list + '/' + img_name
            img = Image.open(img_src_train_path)
            img = img.resize(IMG_SIZE,Image.LANCZOS)
            img.save(img_dst_train_path)

    for img_list in test_dir:

        x = src_test_path+img_list
        img_folders = os.listdir(x)

        for img_name in img_folders:
            img_dst_test_path = dst_test_path + '/'+ img_list +'/' + img_name
            img_src_test_path = src_test_path + '/'+ img_list +'/' + img_name
            img = Image.open(img_src_test_path)
            img = img.resize(IMG_SIZE,Image.LANCZOS)
            img.save(img_dst_test_path)
```

Figure 30: Code Snippet to Data Processing

The `data_pre_processing` function appears to perform the following tasks related to image data preprocessing:

1. Image Size Configuration-

- It sets the desired image size to (227, 227) pixels as `IMG_SIZE`. This indicates that all images will be resized to this specified dimension.

2. Source and Destination Paths Configuration-

- It defines source and destination paths for both training and testing datasets:

  - `src_train_path`: The source directory containing the original training images.

  - `dst_train_path`: The destination directory where resized training images will be saved.

  - `src_test_path`: The source directory containing the original testing images.

  - `dst_test_path`: The destination directory where resized testing images will be saved.

3. Listing Directories-

- It lists the directories (classes) present in the training and testing datasets:

  - `train_dir`: List of classes in the training dataset.

  - `test_dir`: List of classes in the testing dataset.

4. Resizing Images-

- It iterates through each class in the training and testing datasets (`train_dir` and `test_dir`).

- For each class, it lists the images (`img_folders`) present in the respective source directory.

- It then iterates through each image in the class.

- For each image, it constructs the source path (`img_src_train_path` for training and `img_src_test_path` for testing) and destination path (`img_dst_train_path` for training and `img_dst_test_path` for testing) based on the class and image name.

- It opens the original image using the Python Imaging Library (PIL), resizes it to the specified `IMG_SIZE` using Lanczos interpolation, and saves the resized image to the destination path.

In summary, the `data_pre_processing` function resizes all the images in the training and testing datasets to a specified size (227x227 pixels) using Lanczos interpolation and saves the resized images in separate directories within the "resize train" and "resize test" directories, respectively. This preprocessing step is common in image data preparation for machine learning tasks.

```
def model_building():
    IMAGE_SIZE = [224, 224]
    vgg19 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
    for layer in vgg19.layers:
        layer.trainable = True
    folders = glob('Dataset/train/*')
    x = Flatten()(vgg19.output)
    prediction = Dense(len(folders), activation='softmax')(x)
    model = Model(inputs=vgg19.input, outputs=prediction)
    model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
    print("Image Data Generator")
    train_datagen = ImageDataGenerator(rescale = 1./255,
                                       shear_range = 0.2,
                                       zoom_range = 0.2,
                                       horizontal_flip = True)

    test_datagen = ImageDataGenerator(rescale = 1./255)
    training_set = train_datagen.flow_from_directory('Dataset/train',
                                                     target_size = (224, 224),
                                                     batch_size = 50,
                                                     class_mode = 'categorical')
    test_set = test_datagen.flow_from_directory('Dataset/test',
                                                target_size = (224, 224),
                                                batch_size = 50,
                                                class_mode = 'categorical')
    print("model training ")
    r = model.fit(
        training_set,
        validation_data=test_set,
        epochs=30,
        steps_per_epoch=len(training_set),
        validation_steps=len(test_set)
    )
    model.save('model_3.h5')
```

Figure 31: Code Snippet to model building

The model_building function appears to define and train a convolutional neural network (CNN) model for image classification using the VGG19 architecture and Keras. Here's an explanation of what this function is doing:

1) **Image Size Configuration:**
IMAGE_SIZE = [224, 224]: It sets the desired input image size for the model to 224x224 pixels.

Loading the VGG19 Model:

vgg19 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False): This line loads the VGG19 model pretrained on the ImageNet dataset. It specifies the input shape, loads pretrained weights, and excludes the fully connected layers (include_top=False).

2) **Setting VGG19 Layers as Trainable:**
for layer in vgg19.layers: layer.trainable = True: It sets all layers in the VGG19 model to be trainable.

3) **Gathering Class Labels:**
folders = glob('Dataset/train/*'): It gathers the class labels from the training dataset by listing subdirectories under "Dataset/train."

4) **Building the Custom Model:**
x = Flatten()(vgg19.output): It flattens the output of the VGG19 base model.

prediction = Dense(len(folders), activation='softmax')(x): It adds a fully connected layer with softmax activation for classification.

model = Model(inputs=vgg19.input, outputs=prediction): It constructs the custom model by defining the input and output layers.

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']): It compiles the model, specifying the loss function, optimizer, and evaluation metric.

5) **ImageDataGenerator for Data Augmentation:**
It sets up data augmentation for the training dataset and rescaling for the testing dataset using ImageDataGenerator.

6) **Loading Training and Testing Data:**
training_set = train_datagen.flow_from_directory('Dataset/train', ...): It loads the training data using ImageDataGenerator and specifies the target size, batch size, and class mode.

test_set = test_datagen.flow_from_directory('Dataset/test', ...): It loads the testing data in a similar manner.

7) **Model Training:**
model.fit(...): It trains the model on the training data and validates it on the testing data. The training includes multiple epochs, steps per epoch, and validation steps.

8) **Saving the Model:**
model.save('model_3.h5'): It saves the trained model to a file named "model_3.h5."

In summary, this function builds, compiles, and trains a CNN model for image classification using transfer learning with the VGG19 architecture. The training data is augmented for better generalization, and the trained model is saved to a file for future use.

```
add_data()
```

```
Already in the system
Already in the system
Already in the system
Already in the system
Already in the system
Already in the system
Already in the system
Alpinia Galanga (Rasna)
The folder 'Alpinia Galanga (Rasna)' is not empty.
Amaranthus Viridis (Arive-Dantu)
The folder 'Amaranthus Viridis (Arive-Dantu)' is not empty.
Artocarpus Heterophyllus (Jackfruit)
The folder 'Artocarpus Heterophyllus (Jackfruit)' is not empty.
Azadirachta Indica (Neem)
The folder 'Azadirachta Indica (Neem)' is not empty.
Basella Alba (Basale)
The folder 'Basella Alba (Basale)' is not empty.
Brassica Juncea (Indian Mustard)
The folder 'Brassica Juncea (Indian Mustard)' is not empty.
test
The folder 'test' is not empty.
```

```
data_pre_procesing()
```

```
model_building()
```

Figure 32: Code Snippet to image classification model using deep learning

These functions appear to be part of a larger pipeline for building and training an image classification model using deep learning techniques. By executing these functions in sequence, you are organizing your data, preparing it for model training, and then training the model itself.

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image
```

With these imports, you have access to TensorFlow's functionality for deep learning and image processing, making it possible to work with image data and build and train neural network models for tasks like image classification, object detection, and more.

```
model = tf.keras.models.load_model("Test_model_ep30.h5")
```

Figure 33: Code Snippet to loading the pretrained model

After running this line of code, the model variable will hold the pre-trained model loaded from the file. You can use this model for various tasks, such as making predictions on new data, fine-tuning, or further evaluation.

```
def preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Rescale the pixel values to [0, 1]
    return img_array
```

Figure 34: Code Snippet togetting the image file path

The `preprocess_image` function appears to be a Python function that preprocesses an image for use in a machine learning model, particularly a deep learning model. Let's break down what this function does:

1. `def preprocess_image(image_path):`: This line defines a Python function called `preprocess_image` that takes one argument, `image_path`, which should be the path to the image you want to preprocess.

2. `img = image.load_img(image_path, target_size=(224, 224))`: This line loads an image from the specified `image_path` using the `image.load_img` function from the Keras `image` module. It also resizes the image to a target size of (224, 224) pixels. This resizing is common when working with many pre-trained deep learning models that expect a fixed input size.

3. `img_array = image.img_to_array(img)`: This line converts the loaded image `img` into a NumPy array. The image is now represented as a multi-dimensional array of pixel values.

4. `img_array = np.expand_dims(img_array, axis=0)`: It adds an extra dimension to the array to make it compatible with models that expect input data in batch format. The `axis=0` argument specifies that the new dimension should be added as the first dimension, effectively creating a batch of size 1.

5. `img_array /= 255.0`: This line rescales the pixel values in the image array to be in the range [0, 1] by dividing all values by 255.0. This step is often necessary to normalize the pixel values since many deep learning models expect input data in this range.

6. `return img_array`: Finally, the preprocessed image data, represented as a NumPy array, is returned from the function.

In summary, this function takes an image file path, loads the image, resizes it to a specified target size, converts it to a NumPy array, adds a batch dimension, and rescales the pixel values. This preprocessed image data is then ready to be used as input to a deep learning model for tasks like image classification, object detection, or other computer vision tasks.

```
new_image_path = 'Dataset/test/Alpinia Galanga (Rasna)/AG-S-011.jpg'
new_image = preprocess_image(new_image_path)
```

Figure 35:Code Snippet to contains the preprocessed image data

After running this code, the new_image variable contains the preprocessed image data, which can be used as input for a machine learning model, such as a deep neural network, for tasks like image classification, object recognition, or any other computer vision task for which the model was designed.

```
predictions = model.predict(new_image)
```

```
1/1 [==============================] - 0s 118ms/step
```

```
predicted_class_index = np.argmax(predictions[0])
```

```
predicted_class_index
```

Figure 36:Code Snippet to calculate and return the index

The code you've provided is used to obtain the predicted class label for an image based on the model's predictions. Here's what this code does step by step:

1. `predictions = model.predict(new_image)`: This line of code uses the loaded machine learning model (`model`) to make predictions on the preprocessed image (`new_image`). The result is stored in the `predictions` variable. The `predictions` variable typically contains class probabilities for each possible class label.

2. `predicted_class_index = np.argmax(predictions[0])`: This line of code uses NumPy's `argmax` function to find the index of the class with the highest predicted probability. It does this by taking the maximum value in the `predictions` array, and `np.argmax` returns the index corresponding to that maximum value.

3. `predicted_class_index`: Finally, the `predicted_class_index` variable will hold the index of the class that the model predicts to be most likely for the input image. This index can be used to identify the predicted class label if you have a mapping between class indices and class labels.

In summary, this code calculates and retrieves the index of the predicted class label for the input image based on the model's predictions. You can use this index to map it to the corresponding class label for further interpretation or reporting.

```
class_labels = list(test_set.class_indices.keys())
```

Figure 37: Code Snippet to class labels in training datasets

After running this code, the class_labels list will contain the class labels used in your testing dataset. These class labels can be useful for interpreting the model's predictions, especially when you want to associate numeric class indices with human-readable labels.

```
    predicted_class_label = class_labels[predicted_class_index]
    print("Predicted class label:", predicted_class_label)

Predicted class label: Alpinia Galanga (Rasna)
```

Figure 38: Code Snippet toclass labels used in test data

After running this code, the class_labels list will contain the class labels used in your testing dataset. These class labels can be useful for interpreting the model's predictions, especially when you want to associate numeric class indices with human-readable labels.

```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('Dataset/test',
                                            target_size = (224, 224),
                                            batch_size = 50,
                                            class_mode = 'categorical')
```

```
Found 644 images belonging to 30 classes.
```

Figure 39: Code Snippet toevaluating the ML model

After executing this code, the test_set object is ready to be used for evaluating a machine learning model. It can be passed to the model's evaluation function to assess the model's performance on the test data.

```
# Predict using test data
test_imgs, test_labels = next(test_set)  # get a batch of test images and labels
predictions = model.predict(test_imgs)

# The output, 'predictions' is a probability distribution over the classes
# To convert these probabilities into class labels, you can get the class with the highest probability:

predicted_classes = np.argmax(predictions, axis=1)
```

Figure 40: Code Snippet to predictions on test images

The provided code snippet is used to make predictions on a batch of test images using a machine learning model (`model`). Here's what each part of the code does:

1. `test_imgs, test_labels = next(test_set)`: This line of code retrieves a batch of test images and their corresponding labels from the `test_set` data generator. The `next` function is used to obtain the next batch of data. `test_imgs` will contain the batch of images, and `test_labels` will contain the corresponding labels

2. `predictions = model.predict(test_imgs)`: This line of code uses the pre-trained model (`model`) to predict class probabilities for the batch of test images (`test_imgs`). The `model.predict` method takes the input images and returns predicted probabilities for each class.

- `predictions`: This variable holds the predicted class probabilities for each image in the batch. Each row corresponds to an image, and each column corresponds to a class, with values representing the predicted probability of belonging to each class.

3. `predicted_classes = np.argmax(predictions, axis=1)`: This line of code calculates the predicted class for each image in the batch. It uses `np.argmax` to find the index (class) with the highest predicted probability along the specified axis, which is set to `1`. This means that for each image, it selects the class with the highest probability.

After running this code, the `predicted_classes` variable will contain the predicted class labels (as integers) for each image in the batch. These predictions can be compared to the true labels (`test_labels`) to evaluate the model's performance on the test data.

```
# Get class labels
class_labels = list(test_set.class_indices.keys())

# Display some images with their predictions
for i in range(10):  # change the range value if you want to see more or less images
    img = test_imgs[i]
    actual_label = class_labels[np.argmax(test_labels[i])]
    predicted_label = class_labels[predicted_classes[i]]

    plt.imshow(img)
    plt.title(f'Actual: {actual_label}, Predicted: {predicted_label}')
    plt.show()
```

Figure 41: Code Snippet to display test images

The provided code snippet is used to display some test images along with their actual and predicted class labels. Here's how the code works:

1. `class_labels = list(test_set.class_indices.keys())`: This line creates a list of class labels based on the class indices obtained from the `test_set` data generator. These class labels are used to map integer class labels to human-readable labels.

2. The code then enters a loop to display a certain number of images (in this case, 10):

   - `for i in range(10):`: This loop runs 10 times to display 10 images. You can adjust the range value if you want to see more or fewer images.

   - `img = test_imgs[i]`: It retrieves the `i`-th test image from the batch of test images (`test_imgs`).

   - `actual_label = class_labels[np.argmax(test_labels[i])]`: It calculates the actual class label for the `i`-th test image by using the `np.argmax` function to find the index (class) with the highest true label probability in `test_labels`.

   - `predicted_label = class_labels[predicted_classes[i]]`: It calculates the predicted class label for the `i`-th test image by using the `predicted_classes` array, which contains the predicted class indices, and mapping them to the class labels using `class_labels`.

   - `plt.imshow(img)`: This line displays the test image using matplotlib's `imshow` function.

   - `plt.title(f'Actual: {actual_label}, Predicted: {predicted_label}')`: It sets the title of the displayed image to include both the actual and predicted class labels.

   - `plt.show()`: This command displays the image with the specified title.

This code will loop through the specified number of images and display them one by one, along with their actual and predicted class labels, allowing you to visually inspect how well the model's predictions match the true labels for these images.

5.1.2 Initially used Datasets of Herbs

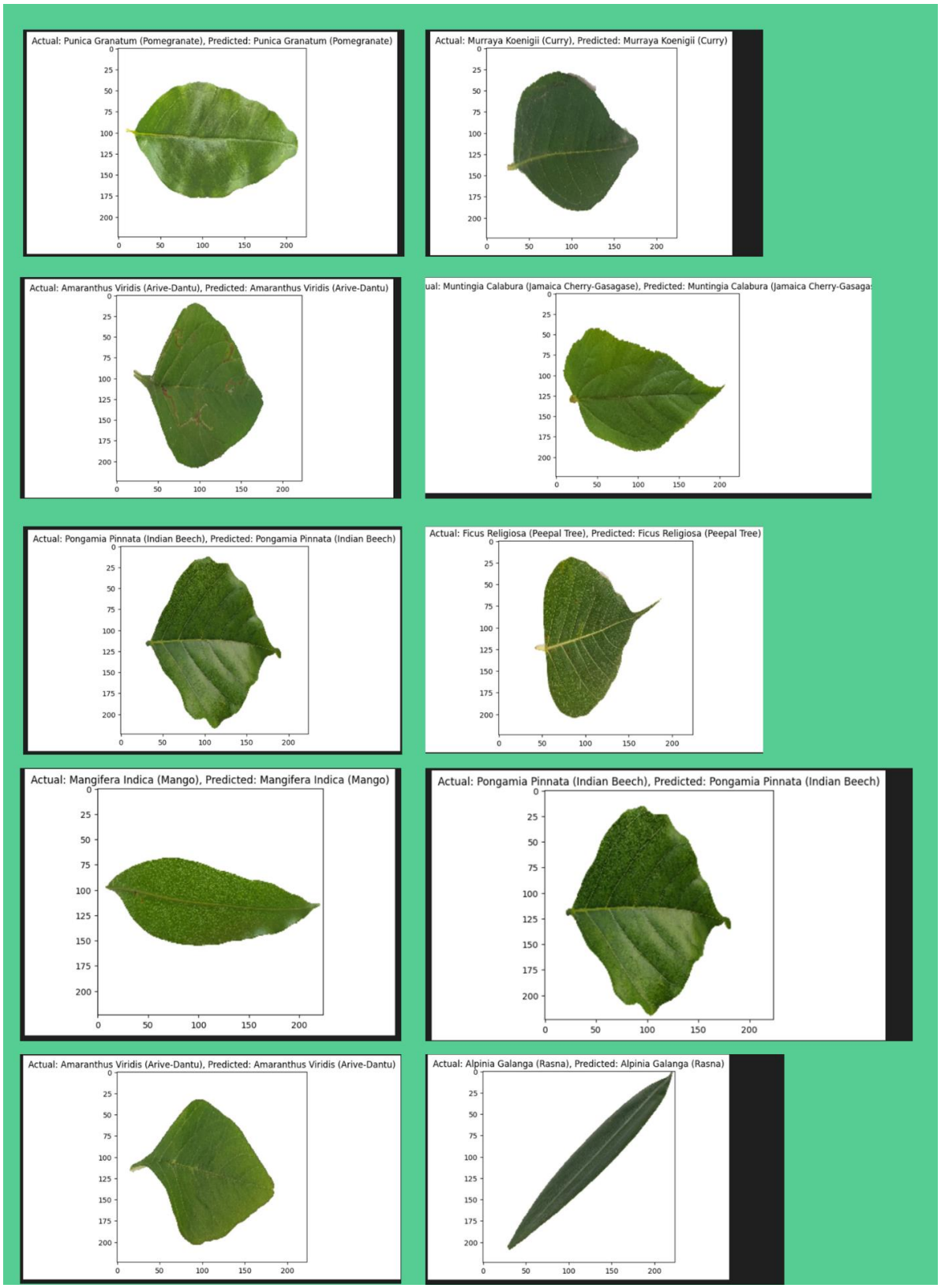

Figure 42: Leaf Image Datasets

5.1.3 FAST API

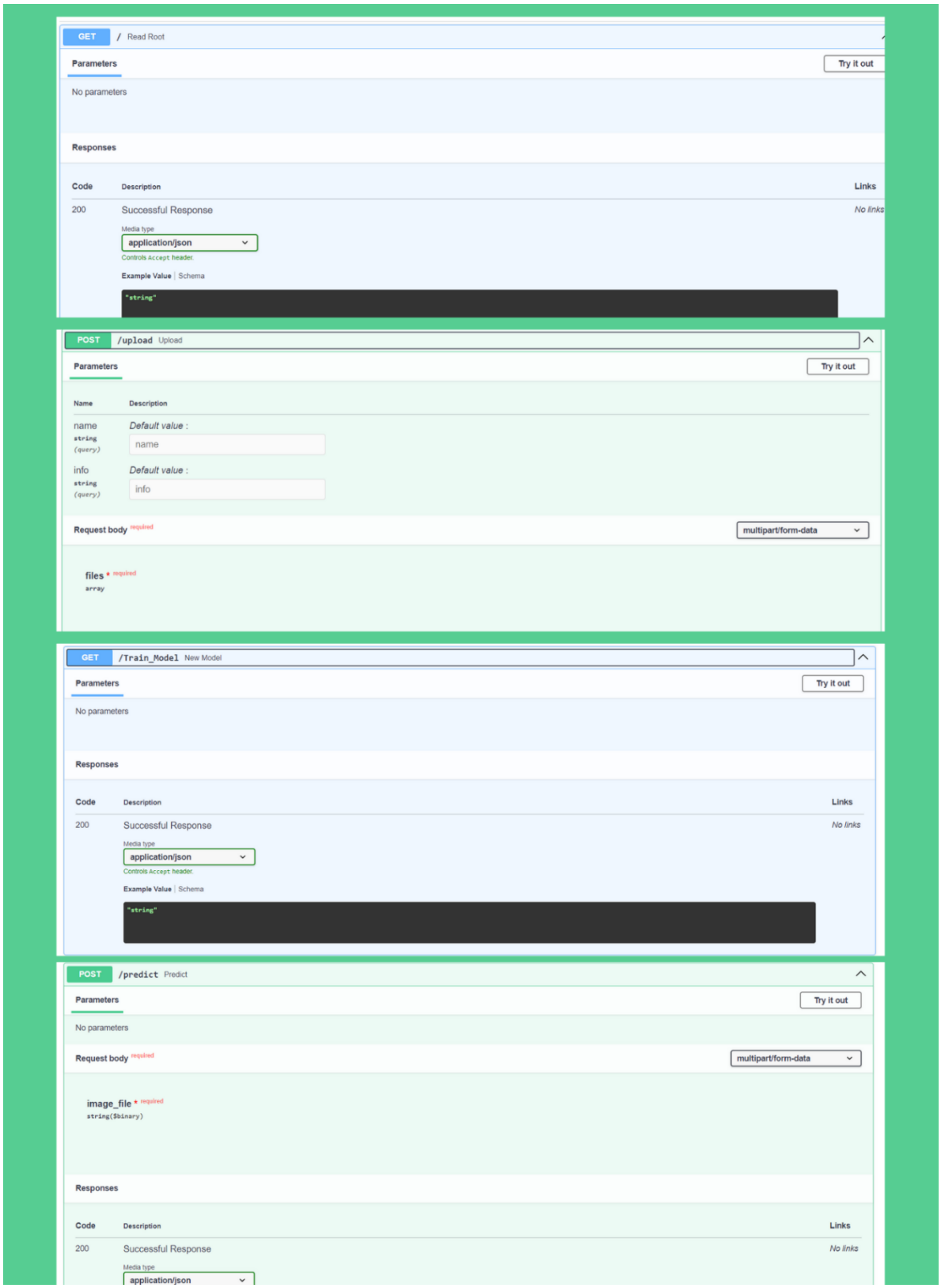

Figure 43: FAST API

5.2 Testing

5.2.1 Test Plan and Test Strategy

The testing strategy specifies the aspects to be tested, and the functions to be tested are selected based on the importance of the functions and the risks they pose to users. The test cases were then written under the available use cases, they were manually handled, and the results were recorded. Test planning is crucial for creating a baseline plan with tasks and accomplishments to track the project's progress. It also demonstrates the test's scope.

1) Employed test Strategy:
2) Define the testable items.
3) Choose the functions based on user risk and relevance.
4) design test cases in accordance with the use case description
5) execute
6) record results
7) find bugs.
8) fix bugs

and repeat the test case until the desired results are obtained.

5.2.2 Test Case Design

Test Scenario	The user uploads an image of a herb for identification.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User uploads an image of a herb.
Expected Output	The image processing component processes the image, identifies the herb, and provides detailed information about the herb, including its name, uses, and properties.
Actual Result	The image processing component successfully identifies the herb in the uploaded image and provides comprehensive details about it.
Status (Pass/Fail)	Pass
Test Scenario	The user captures an image of a herb using the device's camera and requests herb details.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User captures an image of a herb using the device's camera and requests herb details.
Expected Output	The image processing component processes the captured image, identifies the herb, and provides detailed information about the herb, including its name, uses, and properties.

Actual Result	The image processing component successfully identifies the herb in the captured image and provides comprehensive details about it.
Status (Pass/Fail)	Pass

Table 3: Test Case 1

Test Scenario	The user uploads an image containing multiple herbs for identification.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User uploads an image containing multiple herbs.
Expected Output	The image processing component processes the image, identifies all the herbs present, and provides detailed information about each herb, including their names, uses, and properties.
Actual Result	The image processing component successfully identifies all the herbs in the uploaded image and provides comprehensive details about each one.
Status (Pass/Fail)	Pass

Table 4: Test Case 2

Test Scenario	The user uploads an image that does not contain any recognizable herbs.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User uploads an image that does not contain any recognizable herbs.
Expected Output	The image processing component gracefully handles the query, stating that it could not identify any herbs in the image and suggests trying with a different image.
Actual Result	The image processing component responds by saying, "I couldn't identify any herbs in the image. Please try with a clearer picture of a herb."
Status (Pass/Fail)	Pass

Table 5: : Test Case 3

Test Scenario	The user attempts to upload an image, but the upload process encounters an error.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User attempts to upload an image, but the upload process encounters an error.
Expected Output	The image processing component gracefully handles the error, provides a user-friendly error message, and suggests alternative ways to proceed.
Actual Result	The image processing component responds with an error message, such as, "Sorry, there was an issue with the image upload. Please try again or use the camera to capture a herb image."
Status (Pass/Fail)	Pass

Table 6: : Test Case 4

Test Scenario	The user uploads a herb image, but the herb identification process takes too long to complete.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User uploads a herb image.
Expected Output	The image processing component should process the image and identify the herb in a reasonable time frame. If it takes longer than expected, it should provide an error message indicating a timeout and suggest trying again.
Actual Result	The image processing component responds with an error message, such as, "Sorry, the identification process took too long. Please try uploading the image again."
Status (Pass/Fail)	Pass

Table 7: Test Case 5

Test Scenario	The user attempts to upload a low-quality or blurry herb image.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User uploads a low-quality or blurry image of a herb.
Expected Output	The image processing component should check the image quality and, if it determines that the image is of insufficient quality for identification, it should prompt the user to upload a clearer image.
Actual Result	The image processing component detects the low image quality and asks the user to try with a clearer picture of the herb.

Status (Pass/Fail)	Pass

Table 8: Test Case 6

Test Scenario	The user attempts to upload an image that does not contain any herb or contains unrelated objects.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User uploads an image that does not contain any herb or contains unrelated objects.
Expected Output	The image processing component should reject the image as it does not contain a herb for identification. It should provide an error message and suggest trying with an image of a herb.
Actual Result	The image processing component recognizes the absence of a herb in the image and responds with an error message, such as, "No herb detected in the image. Please upload an image of a herb for identification."
Status (Pass/Fail)	Pass

Table 9: Test Case 7

Test Scenario	The user uploads an image of a herb, but the identification is incorrect.
Precondition	The image processing component is operational, and the VGG19 model for herb identification is trained.
Input	User uploads an image of a herb.
Expected Output	The image processing component should accurately identify the herb in the image. If it provides an incorrect identification, it should offer a correction and detailed herb information for the correct herb.
Actual Result	The image processing component incorrectly identifies the herb as "Basil" but corrects itself and provides details for "Mint."
Status (Pass/Fail)	Pass

Table 10: Test Case 8

6. RESULTS AND DISCUSSIONS

6.1 Discussion

Discussion: Enhancing Ayurvedic Healthcare Through Image Identification

The research outlined an ambitious objective: to harness the wisdom of Ayurveda and modern technology to promote a healthier lifestyle. This objective is underpinned by the identification of Ayurvedic medicinal herbs through image processing and the creation of a user-friendly interface for herb recognition and exploration. Let's delve into the implications and significance of this research.

Fulfilling a Healthier Lifestyle with Ayurveda:

The primary goal of this research is to empower individuals to embrace a healthier lifestyle rooted in Ayurvedic principles. Ayurveda offers a holistic approach to well-being, and leveraging modern technology can make its benefits more accessible.

Herb Identification for Disease Management:

One of the pivotal components is the identification of Ayurvedic herbs through image processing. This innovation can assist in addressing common health symptoms by recommending appropriate herbs. This aligns with the ancient Ayurvedic practice of using herbs to manage and alleviate various health conditions.

Addressing Limitations in Plant Identification:

The research acknowledges the limitation of focusing primarily on leaf identification for medicinal plants. While leaves are essential, some herbs are recognized by their roots, stems, and flowers. Expanding the scope of plant identification can enhance its practicality in real-world scenarios.

Balancing Traditional and Modern Medicine:

Recognizing that not all diseases can be treated with Ayurvedic remedies, the research emphasizes the importance of balance. Conventional medical treatments remain crucial, particularly in critical health situations. This balanced approach promotes informed decision-making in healthcare.

Navigating Healthcare Social Networks:

The integration of social networks in healthcare raises concerns about the reliability of shared content. The research highlights the need for caution when consuming health-related information on such platforms. While they facilitate knowledge-sharing, misinformation and unverified claims are pitfalls to watch out for.

Practical Implementation:

The research goes beyond theoretical concepts and focuses on practical implementation. It introduces a user-friendly interface that bridges the gap between Ayurvedic recommendations and user exploration. This interface enables users to understand and identify Ayurvedic herbs with ease.

Technology Integration:

The integration of technology components such as Fast API and Firebase strengthens the system's functionality. Fast API facilitates image uploads and herb predictions, while Firebase ensures seamless storage and retrieval of herb information. This harmonious blend of technology enhances the user experience.

Continuous Learning and User Satisfaction:

The research's emphasis on continual learning and fine-tuning of the herb identification model ensures accuracy. User satisfaction is a critical aspect, and the research aims to measure it to enhance the feature's usability.

Commercialization Potential:

The research envisions the commercialization of its findings through an application named "AYUR MANA." Targeting a wide audience, including patients, Ayurvedic practitioners, and the general public, the application aims to fill a critical gap in Ayurvedic healthcare.

Conclusion:

In conclusion, this research represents a significant step towards making Ayurvedic healthcare more accessible and user-friendly. By integrating ancient wisdom with cutting-edge technology, it bridges the gap between traditional and modern healthcare approaches. The emphasis on balanced healthcare decisions, practical implementation, and user satisfaction underscores its potential to reshape the future of healthcare, promoting well-being through Ayurveda.

6.1.1 Results of CNN based approach.

Model	Accuracy (%)
CNN	95.79
VGG16	97.8
VGG19	97.6

Table 11: CNN comparisons

A comparison between the VGG16 and VGG19 pretrained CNN models and standard CNN was shown. Different plant species' leaves were employed in this experiment. In this case, 80% of the data were used for training and 20% for testing. 10% is applied to validate. 25 epochs have been set as the number. The batch size is 16 pieces. Test accuracy is finally recorded. We'll start by creating a fundamental convolutional neural network from scratch, train it using the training image dataset, and then assess the model, as was previously mentioned. Three convolutional layers with a 3x3 kernel size and ReLU as the activation function make up CNN's design. This layer combined with a 2x2 2D max pool. Three of the initial convolutional layers are used to extract features. The dense fully connected layer receives the layer's output and processes it before making the final prediction. Finally, in order to extract features and categorize photos, we will use the pre-trained models VGG16 and VGG19, which have already been trained on a sizable dataset with a wide variety of categories. The trial outcomes may be shown in TABLE , where it is revealed that VGG16 had the highest classification accuracy (97.8). The accuracy of basic CNN was 95.79, whereas VGG19 was 97.6 accurate. According to the results, VGG16 performed better than VGG19 and standard CNN.

6.1.2 Product Deployment

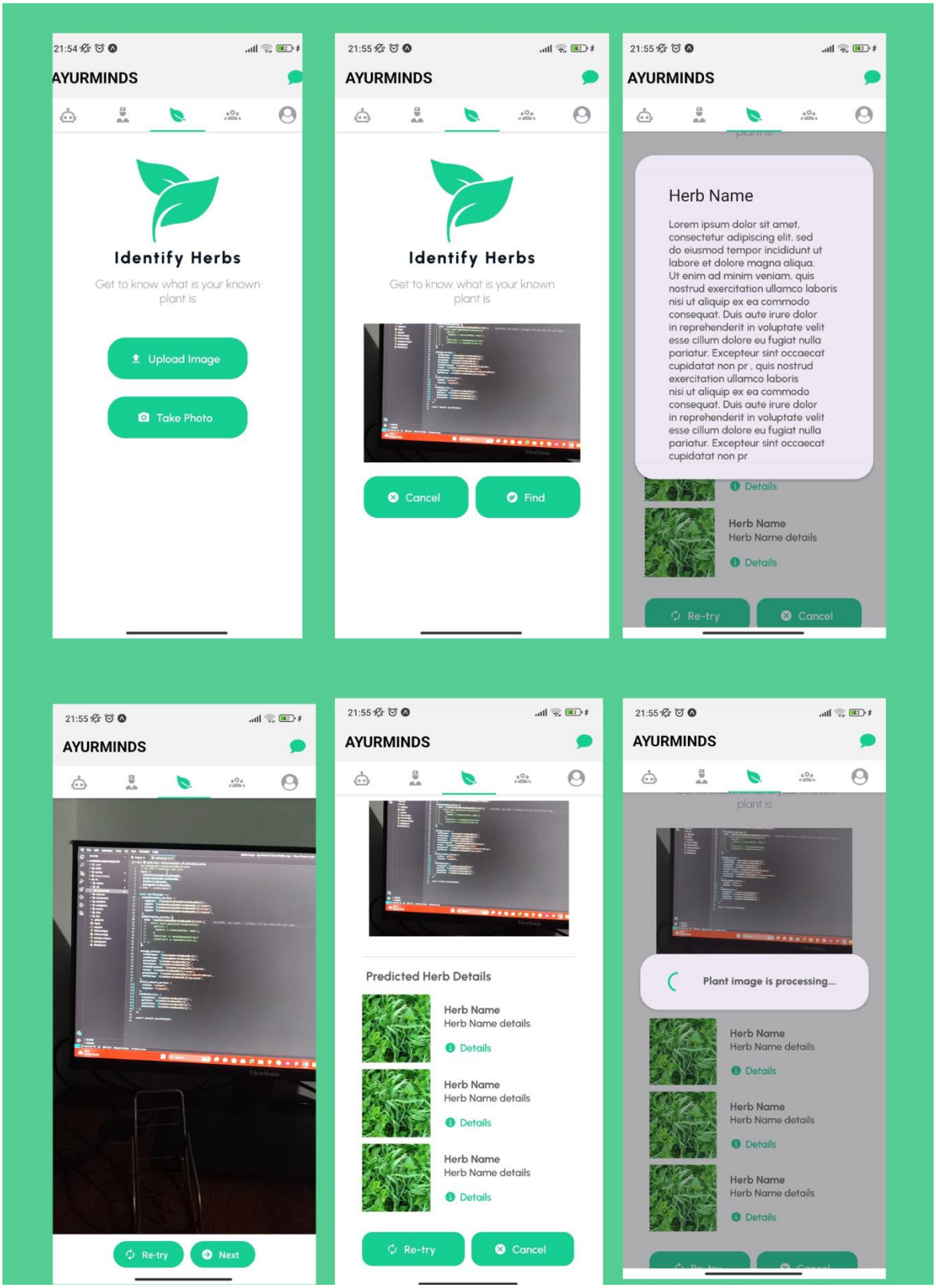

Figure 44: Mobile UI

6.2 Research Findings

7. CONCLUSION

In conclusion, this research embarks on a transformative journey that marries the rich tradition of Ayurveda with the possibilities of modern technology. By doing so, it envisions a future where individuals can seamlessly integrate Ayurvedic principles into their daily lives to foster well-being. The research recognizes the power of Ayurvedic herbs in addressing common health concerns and strives to make this wisdom accessible through innovative image identification techniques. It also acknowledges the importance of balance, advocating for the coexistence of Ayurvedic remedies and conventional medicine. Moreover, in a world increasingly connected through healthcare social networks, it raises a cautionary flag about the reliability of information. Through practical implementation and user-centric design, this research aspires to empower individuals to explore, understand, and embrace Ayurveda. With technology as an ally and user satisfaction as the compass, this endeavor holds the promise of reshaping the healthcare landscape and promoting holistic well-being. As we journey forward, the fusion of ancient wisdom and contemporary innovation beckons a future where Ayurveda takes its rightful place in the realm of healthcare alternatives, nurturing health and harmony.

8. REFERENCES

8. APPENDIX

8.1 Turnitin Report