AROGYA - AN INTELLIGENT AYURVEDIC HERB MANAGEMENT PLATFORM

N.J. Pathiranage

(IT17129404)

BSc (Hons) in Information Technology
Specializing in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

September 2020

AROGYA - AN INTELLIGENT AYURVEDIC HERB MANAGEMENT PLATFORM

N.J. Pathiranage

(IT17129404)

Dissertation submitted in partial fulfilment of the requirement for the Bachelor of Science Special (Honors) In Information Technology Specializing in Software Engineering)

Department of Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

September 2020

DECLARATION OF THE CANDIDATE AND SUPERVISOR

I declare that this is my own work and this proposal does not incorporate without acknowledgment 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 acknowledgment is made in the text.

Name	Student ID	Signature
N.J. Pathiranage	IT17129404	

The supervisor/s should certify the proposal report with the following declaration.

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

…….. ……

Signature of the supervisor: Date

(Mrs. Lokesha Weerasinghe)

…….. ……

Signature of the co-supervisor: Date

(Ms. Ishara Weerathunga)

ABSTRACT

Ayurvedic medicines have a vital role in preserving physical and mental health of human beings. Especially in Sri Lanka we have our own set of rare Ayurvedic herbs which have been utilized by generations as medicinal treatments for a variety of diseases. Lack of experts in this field makes proper identification and classification of medicinal plants a tedious task, which is essential for better treatment. Hence, a fully automated system for medicinal plant classification is highly desirable. Automatic classification of trees from leaves is a popular computer vision/machine learning task and has important applications. With this approach, there are existing applications which can identify plants with low prediction accuracies. However, these applications are based on foreign plant data sets that do not include valuable herbs and shrubs with medicinal qualities. Hence this research proposes a centralized mobile application unique to medicinal plants, which can identify a group of ayurvedic herbs in Sri Lanka, which is able to be performed in both online and offline approach. Here, a new ayurvedic plant dataset prepared from scratch, and preliminary results for classification of 5 types of herbs, compared with several CNN models based on transfer learning are presented. The major objective of the study was to analyze the collected image set using deep neural network architectures based on transfer learning, choose the best architecture, and create a deep learning model that can be applied effectively for this application. The methodology included gathering data, and transfer learning based on deep Convolutional Neural Networks used on noisy image set for processing them using TensorFlow in a local computer. Images were retrained on the available neural network architectures, fine-tuned from pre-trained weights and then the best architecture was selected. The selected algorithm was fine-tuned using data augmentation techniques on the labeled dataset and hyper-parameter tuning. Experimental results indicate VGG-16 as the best CNN classification model which reached a promising testing accuracy of 99.53%. Conclusively, the outcome of this study will be able to be used by locals in identifying herbal plants worldwide accurately.

Keywords: Ayurvedic leaf classification, computer vision, convolutional neural network, deep learning, transfer learning

ACKNOWLEDGEMENT

Several people played important roles in accomplishing this research. First, I would like to express my deepest appreciation and full-hearted gratitude to my supervisor Mrs. Lokesha Weerasinghe and co-supervisor Ms. Ishara Weerathunga who guided us to success in this research and gave entire support to complete this project. Also, I would like to thank Sri Lanka Institute of Information Technology for encouraging us to carry out this study and the academic staff for the continuous guidance, support, and insightful comments, which motivated us to move further. Furthermore, I am grateful to all the resource people of Ayurveda Research Institute at Navinna for giving an ultimate support for collecting relevant information as well as photographs of some selected valuable ayurvedic plants in Sri Lanka. Finally, I wish to extend my warm appreciation to my family members for their encouragement and support to complete this task.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN-Artificial Neural Network

CNN-Convolutional Neural Network

MSF-CNN-Multi-Scale fusion Convolutional Neural Network

DNN-Deep Neural Network

WHO-World Health Organization

HOG- Histograms of Oriented Gradients

LBP- Local Binary Patterns

SVM- Support Vector Machines

ELM- Extreme Learning Machines

PCA- Extreme Learning Machines

KNN- K-Nearest Neighbor

PNN- Probabilistic Neural Network

CBIR- Content-Based Image Retrieval

1. INTRODUCTION

Ayurveda is an ancient medicinal system evolved in India around thousands of years ago, still followed by many people as it is purely natural and has no side effects [2]. It is very relevant from ancient to this most modern time because of its power to cure chronic diseases [2]. The parts like leaf, flower, root, bark, and fruit are mainly used in the preparation of medicines in Ayurveda [2]. In 2009, World Health Organization (WHO) has stated that 80% of the people worldwide still count on ayurvedic drugs and medicine. They have remarkable contributions towards human lives and play a predominant role for the health of the global inhabitants.

According to Ayurveda every plant on earth has some medicinal value, so it is important to protect the plant and identify its medicinal values [30]. Studies have proved that consuming so much of allopathic medicines may lead to side effects as it carries out many chemical reactions within the body [30]. A general fact about Allopathy is that once it is taken, it requires taking another medicine to cure the side effects which has happened due to the previous medicine [30]. In general, process of consuming medicines will not end. Allopathic treatments are meant to treat the Symptoms of a disease whereas Ayurveda treats the root of the disease [30].

One of the major advantages of Ayurveda is that it does not have any side effects as it is purely natural, that is relevant in this most modern time as new diseases evolve due to changed lifestyle and changed diet [4]. Also, majority prefer Ayurveda medicine over Western medicine due to some versatile factors such as Completely natural, Massage treatment, Positive influence on mental health, Fights inflammation, adds to weight loss, Improves heart health, Healthy detox, Individually prescribed methods, etc. [4]. So, it is important for every human being to return to Ayurveda. Almost all general diseases can be cured through Ayurveda using the shrubs and herbs that are around us. Ayurveda also brings lots of foreign money to the country since many foreign countries are inclining towards it. These days a potential class of users exists who favor ayurvedic medicines than modern medicines. Therefore, the knowledge

regarding medicinal plants carried by different age groups should be preserved and protected.

Computer vision, image processing technologies as well as pattern recognition give optimistic results for the identification and classification of herbs. Recognizing a herb with required biological and medicinal values is one of the crucial tasks [20] because it plays a prominent role in the preparation of Ayurveda related medicines and recipes. In addition, real classification of herbs is significant for almost all the persons who are involved in the preparation of herbal medicines [20]. But absence of expert taxonomists is one of the main issues in this research area. Although ayurvedic medicine has less side effects when compared with synthetic drugs, treatment using an incorrectly recognized herbal plant may claim the life of a patient.

Computer vision as well as image processing technics can bridge the gap between the absence of expert taxonomists and potential need in recognition and classification of the herbal plants. Features of leaf, fruit, flower, trunk, and branch normally are being used for classification purposes [33]. According to many research carried out regarding this problem [5-10], there are existing applications which can identify plants with low prediction accuracies. However, these applications are based on foreign plant data sets that do not include valuable herbs and shrubs with medicinal qualities. Additionally, in most of those research, shape, color, and textures have been considered as the significant morphological features to classify majority of plants [32]– [34]. Additionally, since the leaves are available on all seasons, most of the researchers have decided that it would the better way for classification purpose of plants. But the feature "color" is not a wise feature for classification because it differs on seasons and various stages of the same leaf with a different color. Taxonomists use the variations in leaf characteristics as a tool for the classification of medicinal plants. Many research works have been done in this area. But the high rate of inter-class similarity in shape, color and texture features make this problem still a challenging and unsolved one. Hence, a fully automated system to correctly classify the local medicinal plants is inevitable at this point of time.

As a solution for all the mentioned research problems, this research work proposes a mobile application which can identify Ayurvedic plants using Convolution Neural Network models (CNN) with Transfer learning. Several CNN architectures were applied in this study experimentally to obtain the best accurate model. A newly prepared and annotated Ayurvedic plant dataset from Sri Lanka was trained on CNN from scratch. However, training a model from scratch is too costly. To overcome this challenge transfer learning was applied in the Ayurvedic dataset. This model was then used to build a mobile application in the Android platform with TensorFlow-Lite which could identify Ayurvedic plants using the built-in camera module.

1.1 Background & Literature Survey

An overall idea about the importance of Ayurveda in Sri Lanka, current problems faced by people in this domain, apparent tasks that should be there in any automated identification and classification as well as the limitations of previous researches carried out regarding automated leaf classification worldwide have been given in the introduction section. This section brings several works and important attention in the focus of this research which is plant species identification and classification.

As mentioned in the introduction section, researchers have tried many methodologies to extract the features and identify the plant species automatically. According to previous work done, there are existing applications which can identify plants with low prediction accuracies. However, these applications are based on foreign plant data sets that do not include valuable herbs and shrubs with medicinal qualities. Additionally, most of these methods make use of the combination of many parameters like color, shape, texture features etc.

In 2019, research [14] has presented a new oak leaf dataset with preliminary results for classification of 8 types of oak trees. The new research findings include comparative analysis of a small set of hand-crafted geometric features as well as mostly used high-dimensional appearance features which includes Local Binary Patterns (LBP) and

Histograms of Oriented Gradients (HOG). It has been further compared with Support Vector Machines (SVM) classifier with Extreme Learning Machines (ELM). Results indicate an accuracy of 75% with a small set of geometric features, while an accuracy of 92% with high dimensional appearance features.

In 2019, in the study [15], an approach has been proposed which depicts a combination of deep architectures together. Deep features have been extracted from the leaves using the fc6 layer of AlexNet and VGG16 models. After that, dimension reduction of deep features using the Principal Component Analysis (PCA) method has been applied efficiently and the best differentiated features were acquired. Finally, performances against classifications have been calculated using the K-Nearest Neighbor (KNN) algorithm. Flavia and Swedish, which are two popular plant leaf datasets, have been used for the testing of the system proposed. According to the experimental results, accuracy scores 99.42% and 99.64% were achieved for Flavia and Swedish leaf datasets, respectively.

In 2019, in the study [20], a deep learning based Convolutional Neural Network (CNN) model has been proposed to a system named AyurLeaf, in order to classify herbs using leaf features which included shape, size, color, texture etc. In addition, a standard dataset for medicinal plants has been proposed by this research work, which are habitually visible in various regions of Kerala. This dataset contains samples of leaves from 40 medicinal plants. A deep neural network inspired by Alexnet is utilized for the efficient feature extraction from the dataset. Finally, the classification is performed using Softmax and SVM classifiers. A classification accuracy of 96.76% has been achieved upon 5-cross validation, by their proposed model on the AyurLeaf dataset.

In 2019, in the study [21], a model to address the fine-grained plant image classification task by using the wide and deep learning framework which combines a linear model and a deep learning model has been proposed. It sums the result of the wide and deep learning model using a logistic function so that discrete features can be considered simultaneously with continuous image content. Their works have used

metadata such as the date of flowering and locational information for the wide model. This shows that the proposed method gives better performance than a baseline method.

In 2018, the study [16], a multiscale fusion convolutional neural network (MSF-CNN) is proposed for plant leaf recognition at multiple scales. First, an input image is down-sampled into multiples low resolution im- ages with a list of bilinear interpolation operations. Then, these input images with different scales are step-by-step fed into the MSF-CNN architecture to learn discriminative features at different depths. At this stage, the feature fusion between two different scales is realized by a concatenation operation, which concatenates feature maps learned on different scale images from a channel view. Along with the depth of the MSF-CNN, multiscale images are progressively handled, and the corresponding features are fused. Third, the last layer of the MSF-CNN aggregates all discriminative information to obtain the final feature for predicting the plant species of the input image. Experiments show the proposed MSF-CNN method is superior to multiple state-of-the art plant leaf recognition methods on the MalayaKew Leaf dataset and the Leaf Snap Plant Leaf dataset.

In 2018, the [22] has proposed architecture that combined knowledge-based approach to improve the accuracy of plant recognition. Towards this, hybrid features are constructed by merging three types of knowledge-based features; morphological feature, texture feature and color feature with convolutional neural network extracted features. Their architecture consists of three main stages which are data pre-processing, feature extraction and classification. Before features are extracted, images will be resized and augmented in the pre-processing stage. To classify the species of leaf, they consider decision tree and artificial neural network as a classifier. They have experimented on two datasets: Flavia and Swedish dataset. The experimental result shows that the proposed architecture can predict unseen images correctly more than existing models.

In 2017, the study [23] has proposed a Convolutional Neural Network (CNN) architecture to classify the type of plants from the image sequences collected from smart agro-stations. First challenges introduced by illumination changes and

deblurring are eliminated with some preprocessing steps. Following the preprocessing step, Convolutional Neural Network architecture is employed to extract the features of images. The construction of the CNN architecture and the depth of CNN are crucial points that should be emphasized since they affect the recognition capability of the architecture of neural networks. In order to evaluate the performance of the approach proposed in this paper, the results obtained through CNN model are compared with those obtained by employing SVM classifier with different kernels, as well as feature descriptors such as LBP and GIST. The performance of the approach is tested on dataset collected through a government supported project, TARBIL, for which over 1200 agro-stations are placed throughout Turkey. The experimental results on TARBIL dataset confirm that the proposed method is quite effective.

In 2015, the study [24] has proposed a methodology for recognition of plant species by using a set of statistical features obtained from digital leaf images. As the features are sensitive to geometric transformations of the leaf image, a preprocessing step is initially performed to make the features invariant to transformations like translation, rotation and scaling. Images are classified to 32 pre-defined classes using a Neuro fuzzy classifier. Comparisons are also done with Neural Network and k-Nearest Neighbor classifiers. Recognizing the fact that leaves are fragile and prone to deformations due to various environmental and biological factors, the basic technique is subsequently extended to address recognition of leaves with small deformations. Experimentations using 640 leaf images varying in shape, size, orientations, and deformations demonstrate that the technique produces acceptable recognition rates.

In 2015, the [26] studies convolutional neural networks (CNN) to learn unsupervised feature representations for 44 different plant species, collected at the Royal Botanic Gardens, Kew, England. To gain intuition on the chosen features from the CNN model (opposed to a 'black box' solution), a visualization technique based on the deconvolutional networks (DN) is utilized. It is found that venations of different order have been chosen to uniquely represent each of the plant species. Experimental results using these CNN features with different classifiers show consistency and superiority compared to the state-of-the art solutions which rely on hand-crafted features.

In 2014, the [31], the performance of different features extraction methods are compared, different combinations of features and a number of classifiers applied for leaf identification process are also discussed.

In 2012, the [27] uses an efficient machine learning approach for the classification purpose. This proposed approach consists of three phases such as preprocessing, feature extraction and classification. The preprocessing phase involves a typical image processing steps such as transforming to gray scale and boundary enhancement. The feature extraction phase derives the common DMF from five fundamental features. The main contribution of this approach is the Support Vector Machine (SVM) classification for efficient leaf recognition. 12 leaf features which are extracted and orthogonalized into 5 principal variables are given as input vector to the SVM. Classifier tested with flavia dataset and a real dataset and compared with k-NN approach, the proposed approach produces very high accuracy and takes very less execution time.

In 2012, the study [32] has aimed at implementing a system using image processing with images of the plant leaves as a basis of classification. The software returns the closest match to the query. The proposed algorithm is implemented, and the efficiency of the system is found by testing it on 10 different plant species. The software is trained with 100 (10 number of each plant species) leaves and tested with different plant species) leaves. The efficiency of the implementation of the proposed algorithms is found to be 92%.

In 2010, the [28] a method of plant leaf classification is proposed based on the neighborhood rough set. They mainly show that this is applicable to plant leaf classification. Experimental results on plant leaf image database demonstrate that the proposed method is effective and feasible for leaf classification.

In 2008, the study [29] they employ Probabilistic Neural Network (PNN) with image and data processing techniques to implement a general-purpose automated leaf recognition for plant classification. 12 leaf features are extracted and orthogonalize into 5 principal variables which consist the input vector of the PNN. The PNN is

trained by 1800 leaves to classify 32 kinds of plants with an accuracy greater than 90%. Compared with other approaches, their algorithm is an accurate artificial intelligence approach which is fast in execution and easy in implementation.

1.2 Research gap

In computer vision, despite many efforts [8–13], plant identification is still considered a challenging and unsolved problem. It is very challenging since rich plant leaf morphological variations, such as sizes, textures, shapes, venation, and so on. Most existing plant leaf methods typically normalize all plant leaf images to the same size and recognize them at one scale, resulting in unsatisfactory performances [16].

In Sri Lanka we have our own set of rare Ayurvedic herbs. But most of us are unable to identify these plants due to lack of knowledge. Dissemination of knowledge regarding herbal plants is restricted mainly to very limited group of people and is passed down from generation to generation who practice traditional medicine.

So, recognizing these endemic herbal plants is a challenging problem in the fields of Ayurvedic medicine, computer vision, and machine learning, because although leaf classification apps have been implemented for leaves in other countries, such an automated app with a reliable identification and classification mechanism has not still been implemented for the purpose of locals. There are existing applications which can identify plants with low prediction accuracies. However, these applications are based on foreign plant data sets that do not include valuable herbs and shrubs with medicinal qualities. Therefore, it is the main research gap which can be identified through the existing research and implemented automated applications regarding plant identification and classification.

If the main research gap is summarized, it is the “Lack of technological research carried out on the recognition and classification of Ayurvedic plants in Sri Lanka using machine learning and computer vision, and unavailability of a full-automated mobile application for the particular purpose.” This is mostly important in the system of

medicine called Ayurveda in Sri Lanka because identification and classification of medicinal plants is considered an important activity in the preparation of herbal medicines. Because treatment using a wrongly identified medicinal plant may claim the life of a patient. Additionally, it is important for Ayurveda practitioners and also traditional botanists to know how to identify and classify the medicinal plants through computers [5]. They should apply their knowledge of Ayurveda with technical approach too.

Following table shows the comparison of several existing mobile applications with plant classification functionality with our system, in the means of functionalities, technologies and techniques used for implementation, devices in which can be used as well as accuracy, which is the most important criteria to be considered.

Table 1.2.1: Comparison with existing systems and identifying gaps

Reference	**Functionalities of the Application**	**Technology and techniques Used for implementation**	**Devices in which can be used**	**Accuracy (%)**
Pl@ntNet [40]	Image sharing and retrieval for the identification of plants, works with several parts of the plant including flowers, leaves, fruits and bark, allows integrating user's observations in the database to a collaborative workflow involving the members of a social network specialized on plants	Generalist CBIR method for the visual search engine	iPhone and iPad devices	Medium – The percentage of accuracy is not mentioned in the source

LeafSnap [41]	Identification of tree species from photographs of their leaves.	Automatic Visual Recognition, using computer vision components for discarding non-leaf images and for segmentation	iPhone and iPad devices	Medium - The percentage of accuracy is not mentioned in the source
MedLeaf [42]	Medical plant identification based on leaf image and document searching of medicinal plant	Computer vision, machine learning and deep learning, Local Binary Pattern (LBP) to extract leaf texture and Probabilistic Neural Network (PNN) for classification	Android mobile devices	Medium – 56.33%
ApLeafis [43]	Automatically identifies plant species by the photographs of the tree leaves (digital/captured) photograph	Based on CBIR technique	Android mobile devices	Medium – accuracy percentage is not mentioned
Arogya – Our System	Classification of a group of rare ayurvedic plants in Sri Lanka with the use of leaf/digested root/fruit of a particular plant, GIS for plant locations, showcase of recipes and prescriptions from them with accurate details	Computer vision and transfer learning based on deep Convolutional Neural Networks (CNN)	Android mobile devices	Higher accuracy was expected

1.3 Research problem

The issue regarding recognition and familiarization of herbs, only limited to the specified herbal plant leaves, along with the need for the development of herbal plant identification should be addressed by a study. Especially in Sri Lanka we have our own set of rare Ayurvedic herbs. But most of us are unable to identify these plants due to lack of knowledge. Although majority are aware of the living of various medicine plants and their familiar applications, most are still not able to identify which are these medicine plants from the huge diversity of plant sources in the environment, so as their noted and approved applications by health professionals.

There are several methods to recognize a plant. At present, plants are identified manually by taxonomist, which are prone to human errors. But manual process of classification and recognition needs prior knowledge and also it is lengthy. Leaf Identification by mechanical means often leads to wrong identification. Classification and recognition of herbs is essential for better accurate treatment. Absence of experts in this area makes valid recognition of herbal plants a tedious process. This is one of the major problems in this domain.

Additionally, researchers in the Ayurveda related fields, and each party who is involved in the preparation of ayurvedic medicines and others who are concerned with plant studies face with the application of considerable effort on identifying plants. Recognizing a herbal plant with needed medicinal and biological values is one of the major crucial tasks faced by them. It plays a major role in the preparation of ayurvedic medicines. But absence of expert taxonomists is a main issue in this area. Although herbal medicine has less side effects when compared with modern synthetic drugs, treatment using an incorrectly classified herbal plant might claim the life of a patient.

1.4 Research Objectives

1.4.1 Main Objective

The primary objective of the proposed solution was to develop an android mobile application that gives user the ability to identify and classify a group of selected important ayurvedic plant species in Sri Lanka, based on photographs of the plant's leaf as well as other parts such as digested root and fruit, taken with mobile phone. And it was attempted to identify herbal plants using deep learning analysis in order to assist more locals to identify them.

The identification and classification aspect were to be achieved through an android mobile application. The app should use a photograph of a plant leaf/digested root/fruit captured by the user, or taken from the gallery, to give an output of to which plant species it belongs to. Therefore, anyone without prior knowledge will be able to identify and classify the particular ayurvedic plant hopefully.

The development strategy and methodology used in this approach will be able to be used and extended to identify any ayurvedic herb in Sri Lanka furthermore.

1.4.2 Specific Objectives

1. A newly captured, prepared, and annotated Ayurvedic plant dataset from Sri Lanka was to be trained on Convolutional Neural Network (CNN) from scratch.

2. The obtained noisy image set of medical leaves/digested roots/fruits were to be analyzed using deep neural network architectures (Convolution Neural Network models) based on transfer learning.

• However, training a model from scratch is too costly. To overcome this challenge transfer learning was to be applied on the Ayurvedic dataset.

3. The best Convolution Neural Network architecture, which would produce the highest accuracy was to be used to produce the final deep learning prediction model, that could be applied effectively for the relevant application.

• According to the above methodology, several CNN architectures were to be applied in this study experimentally to obtain the best accurate model, by comparing the accuracies with each.

4. This model was to be used to build a mobile application in the Android platform with TensorFlow-Lite which could identify Ayurvedic plants using the built-in camera module.

• On a mobile device, the Ayurveda plant detection has to be done with time, battery life critical manner, especially when it has to be done in a forest area. In the proposed system the whole identification process would take place on mobile devices where internet connection is not essential. Therefore, this would be a great solution to identify Ayurveda plants in deep forest areas, where mobile network coverage is not available.

2. METHODOLOGY

A methodology is a set of guidelines or principles that can be applied to a specific situation. Within the project duration, below guidelines and approaches were followed in each research component. A methodology could also be a specific template, approach, forms, and testing used over the project life cycle. A formal project methodology should lead the work of all team members throughout the life cycle of a project.

In methodology part, it shows the way how the research was managed and how it was developed. This section exaggerates the project scope, project features, and final outcome of project within time duration of one year of period. To fulfill the above outcomes, Agile methodology was used throughout the software development. Agile software development methodology refers to a group of software development methodologies. It is based on iterative model.

2.1 Understanding the key pillars of the research domain

The system mainly relies on the key pillars of Deep Learning, Transfer Learning based on deep CNN and Data Augmentation.

2.1.1 Deep Learning

According to many previous research works done [10-30], recognition of herbs has been carried out for the last few decades using classical image processing techniques. They have used shape, texture, and color-based features for the performance purpose. Compactness, aspect ratio, skewness, energy, eccentricity, kurtosis, correlation, sum variance and entropy are some of these features [20]. Excess computation time acquired for handcrafted feature extraction is one of the major issue associated with these classical ways. At present, all the classical methods are replaced by machine learning techniques.

Deep learning can be defined as a class of machine learning in which a computational model learns to perform classification tasks directly from images. It is a machine learning technique that teaches computers to do what comes naturally to humans: learn

by example [37]. Ability to handle huge volumes of image data, higher accuracy, inbuilt ability to use GPUs for parallel computation as well as availability of inbuilt pre-trained convolutional neural networks contribute towards the majority use of deep learning. Since Arogya mobile app performs classification on huge volume of image data, this approach will be the best choice.

2.1.2 Convolutional Neural Network (CNN)

A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data [39]. A CNN uses a system much like a multilayer perceptron that has been designed for reduced processing requirements. The layers of a CNN consist of an input layer, an output layer and a hidden layer that includes multiple convolutional layers, pooling layers, fully connected layers and normalization layers. The removal of limitations and increase in efficiency for image processing results in a system that is far more effective, simpler to trains limited for image processing and natural language processing [39].

CNN has been recognized as a competent method for image recognition in past few years, on mobile devices in an offline environment. CNNs work somewhat similar to neurons in brain. It also performs image transformation with a certain degree of rotation and distortion [35]. Since this network is avoiding the complex preprocessing of the image, we will be able to input the original image of the suspected plant directly.

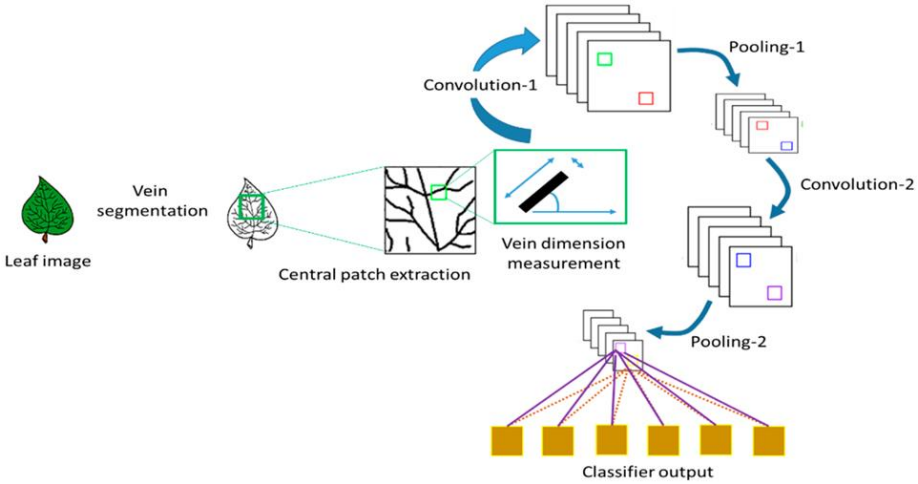

Figure 2.1.2.1: Review on techniques for plant leaves CNN:

Source: https://www.google.com/https://www.mdpi.com

2.1.3 Transfer Learning based on deep CNN

An interesting benefit of deep learning neural networks is that they can be reused on related problems. Transfer learning refers to a technique for predictive modeling on a different but somehow similar problem that can then be reused partly or wholly to accelerate the training and improve the performance of a model on the problem of interest [38]. In deep learning, this means reusing the weights in one or more layers from a pre-trained network model in a new model and either keeping the weights fixed, fine tuning them, or adapting the weights entirely when training the model [38].

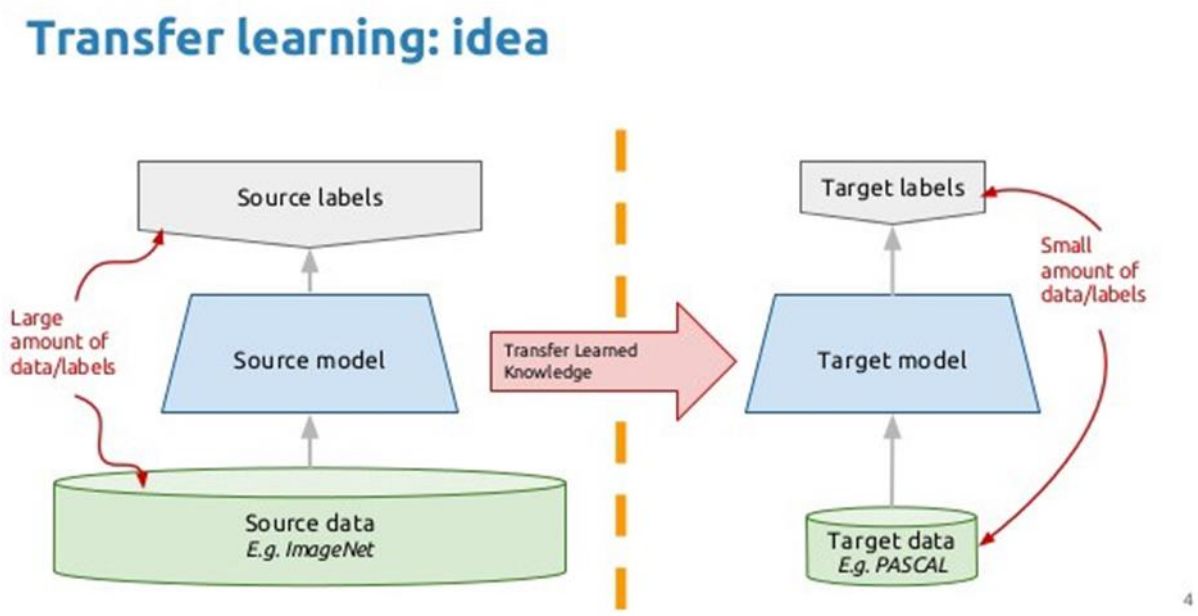

Figure 2.1.3.1: Transfer Learning with Convolutional Neural Networks in PyTorch:

Source: https://www.google.com/towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch

Following is the general outline for transfer learning for object recognition [18]:

1. Load in a pre-trained CNN model trained on a large dataset
2. Freeze parameters (weights) in model's lower convolutional layers

3. Add custom classifier with several layers of trainable parameters to model

4. Train classifier layers on training data available for task

5. Fine-tune hyperparameters and unfreeze more layers as needed

2.1.4 Data Augmentation

As mentioned in many research in this domain, among majority of DNN structures, CNN are used currently as the main tool for the image analysis and classification purposes [17]. Even though great achievements and perspectives have been populated, deep neural networks and accompanied learning algorithms have some relevant challenges to be tackled [17]. The most frequently mentioned issue in the area of machine learning and deep learning, is the lack of sufficient amount of training data or uneven class balance within the datasets [17]. One of the ways of dealing with this problem is so called data augmentation [17]. For training the convolutional neural network that we use for extracting features, there is an idea for training images to be augmented. As a CNN requires a large number of images to train it, various data augmenting methods is planned to be applied to the collected dataset to increase the size of the dataset.

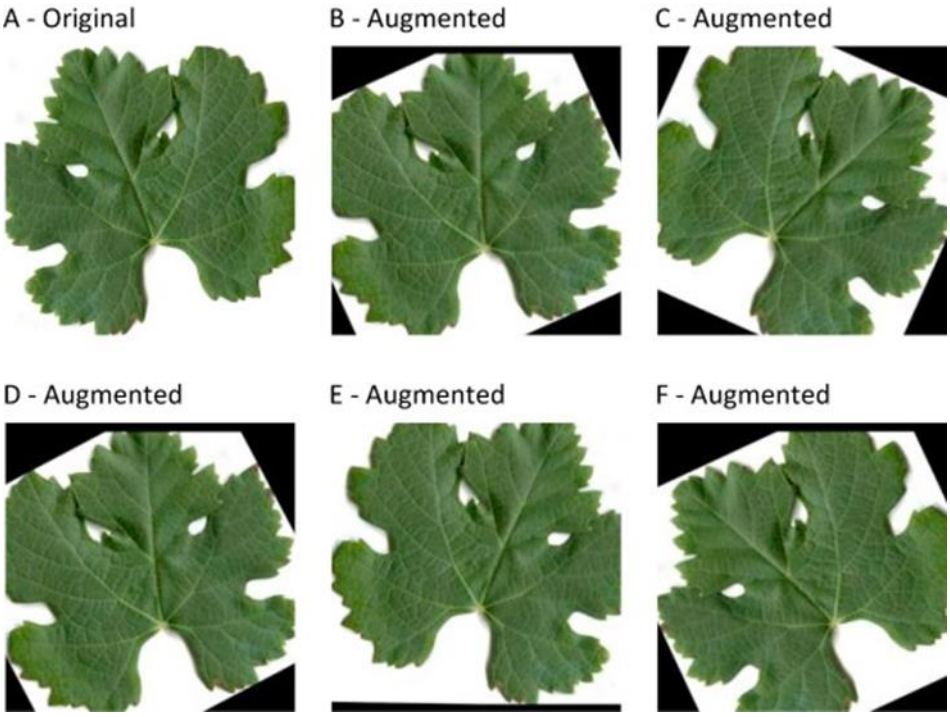

Figure 2.1.4.1: Data Augmentation with leaves:

Source: https://www.google.com/https/www.sciencedirect.com/science/article/pii

2.2 Classification of Ayurvedic plants using transfer learning based on deep CNN using a mobile application in an offline approach

The following diagram illustrates the methodology followed in order to implement the classification of the selected ayurvedic herbal plants in Sri Lanka using a mobile application in an offline environment.

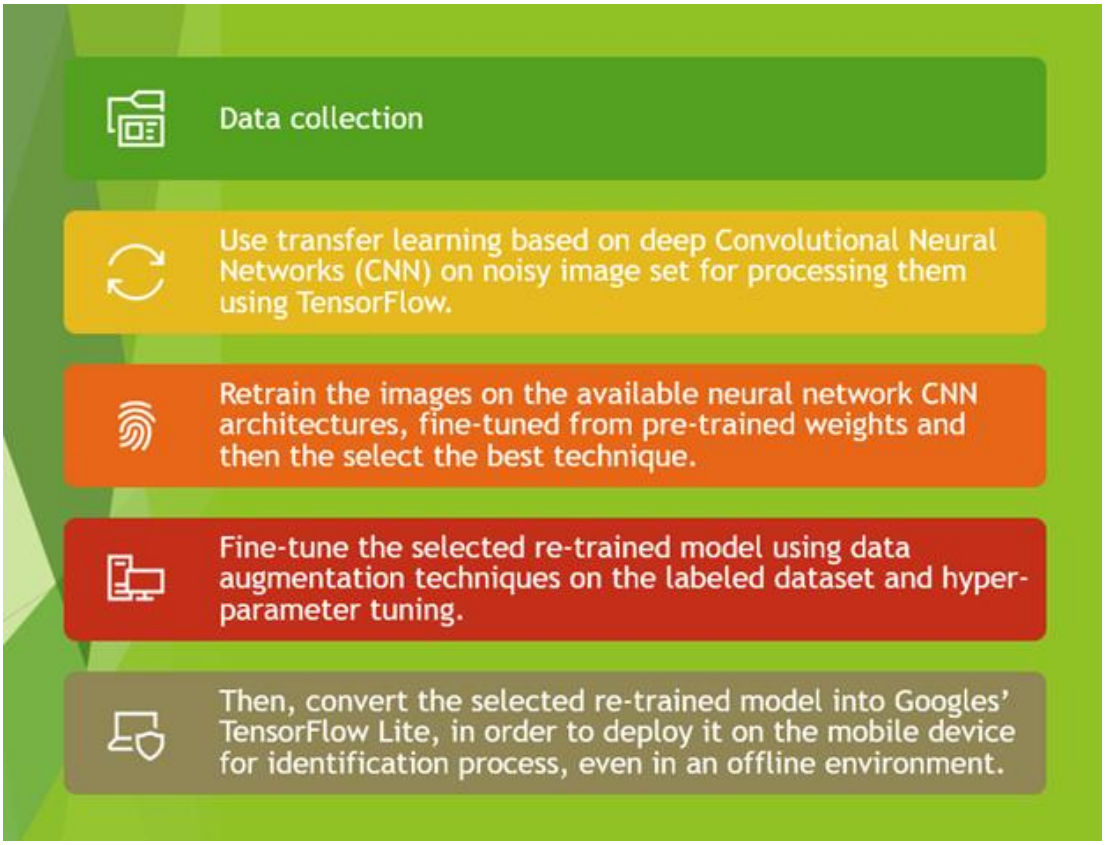

Figure 2.2.1: Methodology of the approach

2.2.1 Data Collection

In this research, the images of the leaf/digested root/fruit that was to be scanned using the application acted as the input for this phase. Among many herbal plants, 5 of the most important ayurvedic plants in Sri Lanka were chosen to analyze further in detail

and the images of the plants were acquired from plant nursery of Navinna, Ayurveda Medical Hospital, social media, Institute of Ayurveda, alternative medicine website and blogs related to Sri Lankan herbal plants creating a noisy web data set.

The 5 types of plants chosen to be classified were:

1. Akkapana (leaf)
2. Cinnamon (leaf)
3. Katupila (leaf and fruit)
4. Kohomba (leaf)
5. Turmeric (leaf and digested root)

The following are some camera-captured images of the above mentioned ayurvedic herbs, acquired from the Research Institute at Navinna.

Figure 2.2.1.1: Akkapana leaf

Figure 2.2.1.2: Cinnamon leaf

Figure 2.2.1.3: Katupila leaf

Figure 2.2.1.4: Katupila leaf

Figure 2.2.1.5: Turmeric leaf

Figure 2.2.1.6: Turmeric digested root

Figure 2.2.1.7: Kohomba leaf

A set of 1348 camera captured as well as lab images were collected which included 214 Akkapana leaves, 230 Cinnamon leaves, 289 Katupila leaves including fruit, 249 Kohomba leaves, 158 Turmeric leaves and 208 turmeric digested root. These were not the augmented images, but original images.

2.2.2 Methodology

The main target was to analyze several deep CNN models with the acquired training and testing data from scratch, get the final testing accuracy from each in order to compare and achieve the model with the highest accuracy, and then to use it as the finalized model in the herbal plant classification purpose in Arogya, based on images as the input from the mobile camera module.

The proposed identification method was based on running CNN, which has been recognized as a competent method for image recognition in the past few years, on mobile devices in an offline environment. CNNs work somewhat similar to neurons in the brain. It also performs image transformation with a certain degree of rotation and distortion [3]. Since this network is avoiding the complex preprocessing of the image, we were able to input the original image of the selected Ayurveda plant directly.

After acquiring the images of leaves/fruits/digested roots of the selected Ayurveda plants, they were labelled and annotated using "VGG Image Annotator" tool for multi-class classification, by forming 6 classes for the specific plant. After the dataset was prepared, transfer learning based on deep Convolutional Neural Networks (CNN) was used on the prepared image set for processing them using TensorFlow, in the local computer. When taken as overall, a sample of 1348 images were prepared, where 48%, 26% and 26% of that were used as training, testing, and validation splits, without data augmentation.

With the requirements of this functionality, we identified that it is easy to use transfer learning than building and training a model from scratch. However, training a model

from scratch is too costly. To overcome this challenge transfer learning was applied in the Ayurvedic dataset. In transfer learning it is able to re-train an already implemented model with a custom dataset. A colab's [12] free GPU which is offered for 12 hours free, was used to re-train the deep learning CNN model.

2.2.2.1 Transfer learning vs. Non-transfer learning

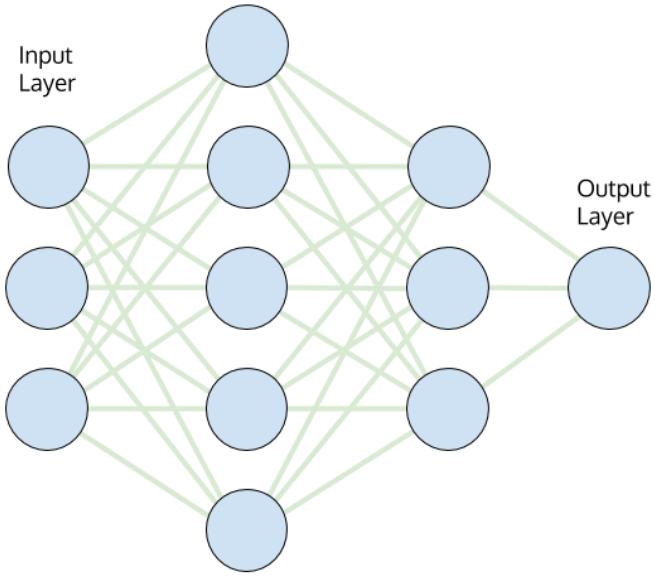

Figure 2.2.2.1.1: Transfer-learning neural network model, green color indicating the training of just the final layer's weights and biases and red color indicating fixed weights and biases
Source: https://towardsdatascience.com/how-to-train-your-model-dramatically-faster-9ad063f0f718

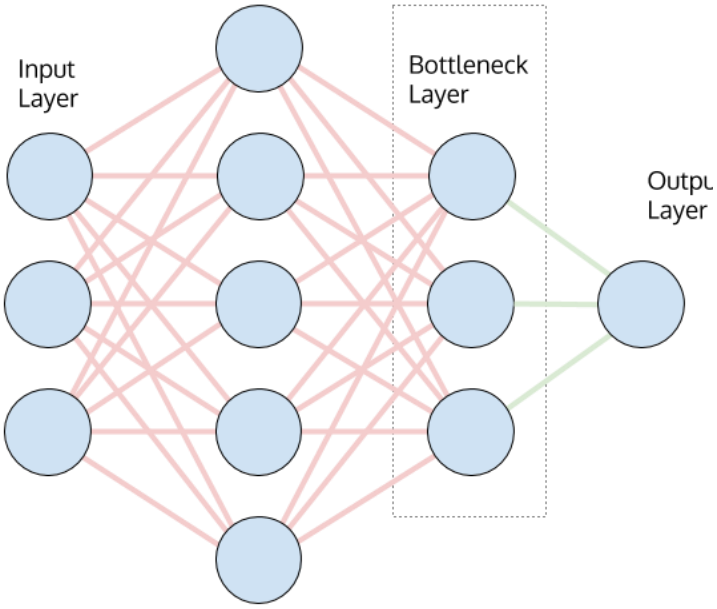

Figure 2.2.2.1.2: Transfer-learning neural network model, green color indicating the training of just the final layer's weights and biases and red color indicating fixed weights and biases
Source: https://towardsdatascience.com/how-to-train-your-model-dramatically-faster-9ad063f0f718

Following are the main benefits of using transfer learning in this problem,

1. Training on a new dataset is faster than implementing from scratch.

2. We can solve the problem with lesser amount of training data than the amount of data that we need if we start from scratch.

After that, the dataset was uploaded to google drive and were retrained with colab on the available neural network CNN architectures, then was fine-tuned from pre-trained weights and then the best technique with the highest accuracy was selected. While the training dataset was augmented, testing and validation datasets were not augmented for higher accuracy purposes.

The variants of deep Convolutional Neural Networks used to compare the initial training and testing accuracies included:

- InceptionV3
- MobileNetV2
- InceptionResNetV2
- Xception
- DenseNet121
- ResNet50
- VGG16

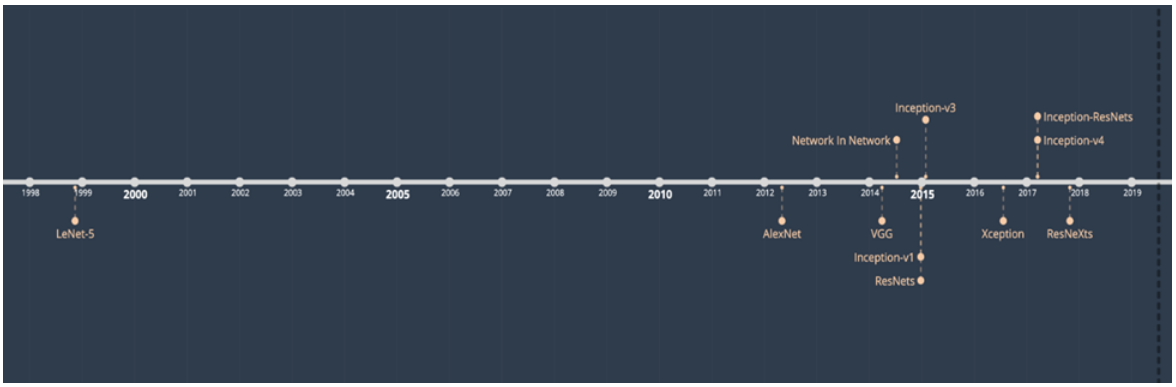

Figure 2.2.2.1: The 10 architectures and the year their papers were published. Source: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614

Then, the selected re-trained model with the best accuracy was fine-tuned using data augmentation techniques on the labeled dataset and hyper-parameter tuning. While the training dataset was augmented, testing and validation datasets were not augmented for higher accuracy purposes. It was re-trained with the dataset using Keras which is an open source neural network library, written in python. It is user friendly, modular, and extensible since it has been designed for fast experimentations with deep neural networks.

The next step was to deploy the re-trained model on mobile devices. Therefore, the re-trained model was converted for the deployment on mobile devices with Googles' TensorFlow Lite. TensorFlow Lite is a TensorFlow's lightweight solution for mobile devices which provides on-device machine learning inference with a small binary size

and low latency. TensorFlow provides an interface for expressing machine learning algorithms and an application for executing these algorithms [36].

Following diagram shows the architecture of TensorFlow lite.

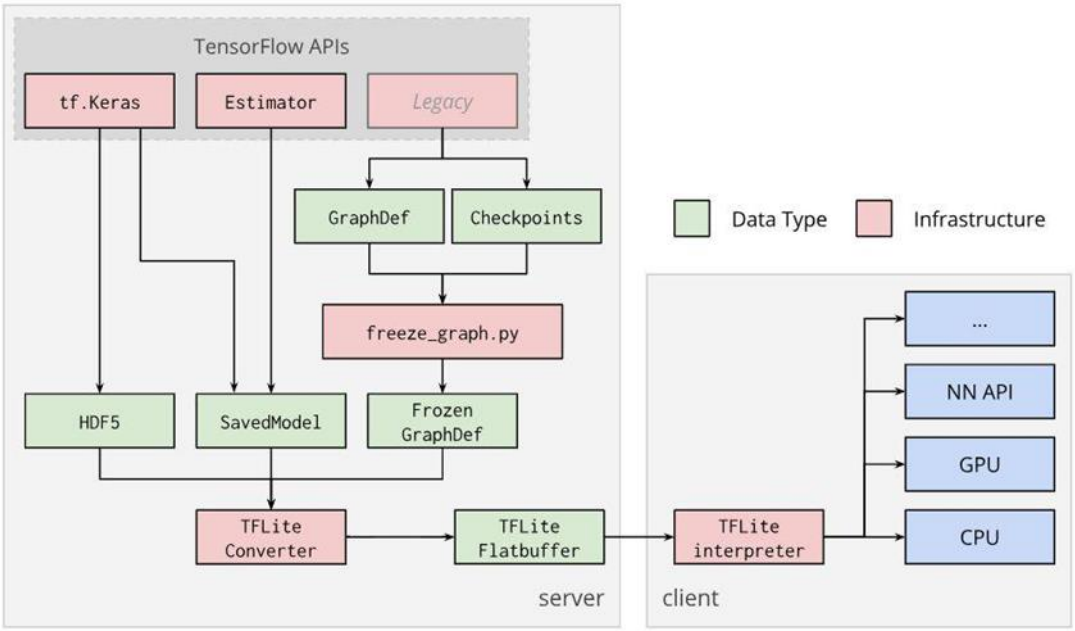

Figure 2.2.2.2: TensorFlow Lite Converter
Source: https://www.tensorflow.org/lite/convert

TensorFlow lite converter has been used to convert the re-trained model to TensorFlow Lite.

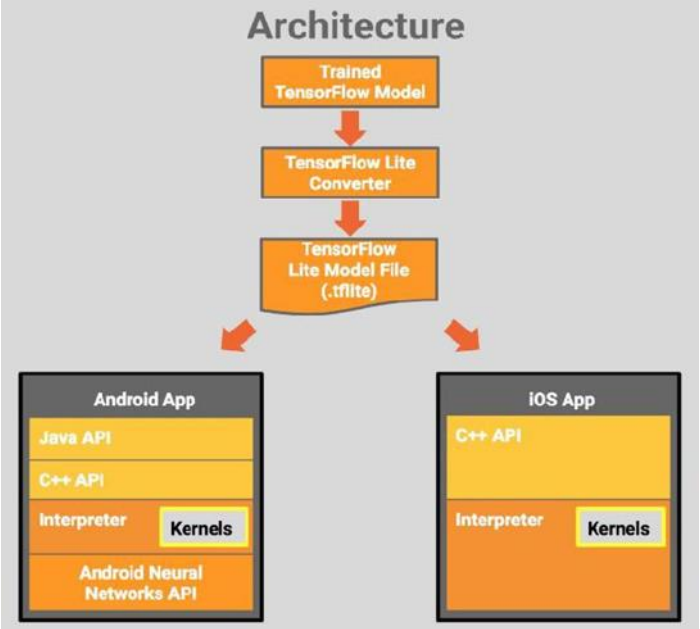

Figure 2.2.2.3: TensorFlow Lite Architecture
Source: https://www.tensorflow.org/lite/guide

One of the main challenges in detection of Ayurveda plants are similarity between different plant species in different phases of their life cycles and complexity of the background since the end user will be using this application in deep forest environment. Therefore, we cannot just rely on a single feature, such as color, texture, or shape to distinguish them. The same species of leaves will be different because of the shades of colors, shape, scale, viewpoint etc. [35].

On a mobile device the Ayurveda plant detection has to be done with time, battery life critical manner, especially when it has to be done in a forest area .In the proposed system the whole identification process will take place on mobile devices and it doesn't require internet .Therefore this will be a great solution to identify invasive plants in deep forest areas, where mobile network coverage is not available .The mobile application will be built using android.

2.3 Summary of methodology

The following table summarizes the methodology to be used in the functionality of Ayurvedic leaf classification precisely.

Table 2.3.1: Summary of methodology

Research	**Research Work**	**Methods and Algorithms to be used**	**Expected Accuracy**	**Remarks**
Arogya – An Intelligent Ayurvedic Herb management Platform -> Classification of a group of rare ayurvedic leaves in Sri Lanka	Use transfer learning based on deep Convolutional Neural Networks (CNN) on collected image set for processing them using TensorFlow (The selected re-trained model will be finetuned using data augmentation techniques on the labeled dataset and hyper-parameter tuning.)	CNN Architectures (Such as Inception v3, ResNet50, VGG-16, MobileNet v2, etc.) will be used to choose the best technique	Higher Accuracy will be expected	Several existing algorithms will be analyzed, and accuracy will be tested in order to select the suitable algorithms to classify the selected 5 plants accurately.

2.4 High-Level Architecture Diagram

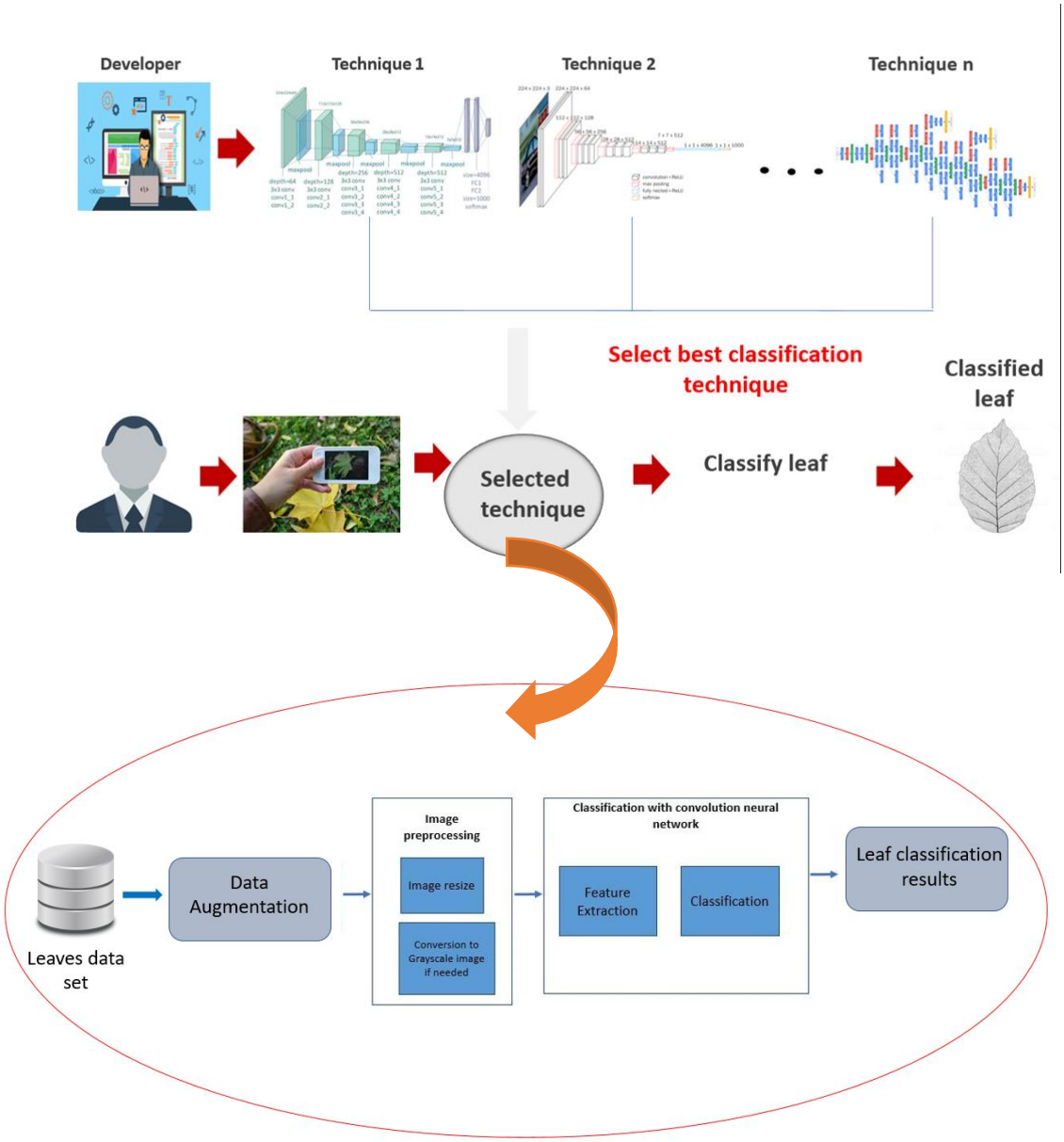

Figure 2.4.1: Software Architectural diagram

2.5 Project Requirements that have been achieved

2.5.1 Functional Requirements

1. Classification of a set of selected Ayurveda herbs in an offline environment has been achieved.

2. Selecting the best classification technique with comparing several transfer learning models based on deep CNN has been achieved.

3. Comparison of the accuracies of these models with justification has been achieved.

4. Request to re-train the existing model with new datasets by premium uses can be done.

5. Showing the classification history on the mobile device has been implemented.

6. Management of Ayurveda herb details in mobile device has been implemented.

2.5.2 Non-Functional Requirements

1. The application is able to give the results **as fast as possible**.

2. Accuracy is high, and results are efficient. The **reliability** of this research function can guarantee 95% in average. (for Akkapana – 100%, Cinnamon – 90%, Katupila – 90%, Kohomba – 100%, Turmeric root – 100%, Turmeric leaf – 95%)
3. **Feasible** in order to understand the functionalities of the app.

4. **Usability**- Usability means how easy to use this mobile application, also it includes how easy it is to learn and how easily user can use this application.

This mobile application has user guidelines which provide all the information about how to use this mobile application properly.

5. **User friendly** interfaces have been provided. User interfaces of mobile app are looking pleasant and even attractive, often promoting confidence in its use. The way to use the mobile application is understandable by the image icons in the interfaces.
6. **User Experiences** are properly managed in order to achieve the specific functionality.
7. **Safety**- This application is not causing harm, injury, or damage to the user or his/her mobile.
8. **Security**- Security of this app is high because only a registered user can use this app and they have to log in before they use this application. Also, user has to provide suitable username and password to access this mobile app which was given at the registration.
9. **Modifiability**- The system has the ability to add new features and new data. Therefore, whole application is built with object oriented and module concepts.

2.5.3 Other Requirements

To run this mobile app, user needs an android mobile which has the capability to store 30MB storage space. No internet connection is needed to use this mobile application, can be used in an offline approach.

2.6 Consideration of the aspects of the system

Standards: We followed coding standards when doing our individual coding parts. Coding standards tell developers how they should write their codes. All the group members used object-oriented concepts to maintain coding standards. Inside the code, we commented important things. When writing reports and referencing, we followed IEEE format.

2.6.1 Social aspects

This mobile application can be used by any person who is not aware about but keen to experience Ayurveda medication worldwide. For example, people who use and wish to use ayurvedic medicines and treatment, researchers in the field of botany, medicine, chemical structure analysis, agriculture, ayurvedic medicinal practitioners, taxonomists, forest department officials, those who are involved in the preparation of ayurvedic medicines and others who are concerned with plant studies, as well as doctors, students, locals, foreigners, Ayurvedic plant sellers and many more can use this application wisely. Therefore, a great appreciation as well as a demand can be expected from the society for this mobile application. Especially, there is no age limitations for the users, no need of advanced computer literacy, as well as no need of advanced knowledge in Ayurveda field to use this app. Only a simple knowledge of how to use a smart mobile phone will be sufficient for this. In addition, the name of the identified plant is given in native as well as in the scientific name of the plant, which would support any person including foreigners. So, this would be very beneficial for the whole society which would do a great service in uplifting our ancient Ayurveda, which is anyway better than modern medicine due to its naturality.

2.6.2 Security aspects

Security of this mobile application is high, because only a registered user can use this app and they have to log in before they use this application. Also, user has to provide suitable username and password (correct credentials) to access this mobile app which were given at the registration process. Credentials of all the registered users are stored in the firebase database with high security rules, so there will not be any unauthorized access for them because it is pretty similar to the security rules followed when creating the Facebook application. (this mobile application is created as a centralized social media platform) Server-side security is also very high because all the services are hosted in Azure with IAM (Identity Access Management) user credentials.

2.6.3 Ethical aspects

This application is not causing any harm, injury, or damage to the user or his/her mobile. No unethical behaviors, rules or principles have been followed in the implementation of this mobile application. This will be capable of identifying majority of ayurvedic plants through any part of it like leaves, root, fruit, flower, etc. if retrained with more datasets by professionals. However, this does not replace the role of a plant taxonomist or an Ayurveda doctor, but it supports any person without any background knowledge about Ayurveda to classify a particular Ayurveda plant and to know about the medicinal value of it hopefully. This would more be a learning resource for almost all persons who need to experience Ayurveda.

2.6.4 Limitations

- This application is currently built only in English. Further, this can be applied in native languages such as Sinhala, Tamil as well as in any of the other languages preferred.
- Since the system is designed only as a mobile application, later can be improvised to a web application with the same functionality and content.

- Currently this mobile application is developed only for Android users, so have to consider about other mobile platforms and operating systems as well.
- The development strategy in this approach is only to identify 5 categories of plants, but the same strategy and methodology can be used and extended to identify any ayurvedic herb worldwide.
- If the user ends up with doubts and clarifications regarding this procedure, this application can be facilitated with consultation help from Ayurveda doctors.

2.7 Commercialization aspects of the product

2.7.1 Target Audience

- People who use ayurvedic treatment
- Researchers in the field of botany, medicine, chemical structure analysis, agriculture, ayurvedic medicinal practitioners, forest department officials, those who are involved in the preparation of ayurvedic medicines and others who are concerned with plant studies
- Doctors, Students, locals, and foreigners
- Ayurvedic plant sellers

2.7.2 Market Space

- No age limitations for the users
- No need of advance computer literacy
- No need of advance knowledge in Ayurveda field

2.7.3 Revenue Earning

- Through subscription fee
- Revenue via additional services

2.8 Testing and Implementation

2.8.1 Implementation

The client side runs as a mobile application which was built on the top of Android. The user login and registration along with profile management, herb classification and providing biological details about the classified herb run on the mobile client itself.

The finalized deep CNN model achieved with the highest accuracy of 99.53% related to herb classification (VGG-16) was trained on Jupiter notebook, and retrained on Google Colab itself with data augmentation techniques. The data augmentation techniques included position (crop, scale, zoom, rotate, padding) and color (brightness, contrast, hue) techniques. The implementation of the model was based on Python environment, Keras, Tensorflow and OpenCV. After the best model was obtained, it was converted to Tensorflow Lite in order to be deployed in the Android Operating System.

Server-side API s related to herb classification, giving the result related to the correctly classified herb and giving the biological details of the specific herb were hosted and deployed on Azure as cloud services. The activation of those deployed services depends on the starting and stopping of particular services deployed in Azure. The best model achieved was deployed as a service on a virtual machine and was integrated with plant detection mechanism.

2.8.1.1 Frontend Implementation

This component comes with the frontend of mobile application.

Environment: Android IDE with Java.

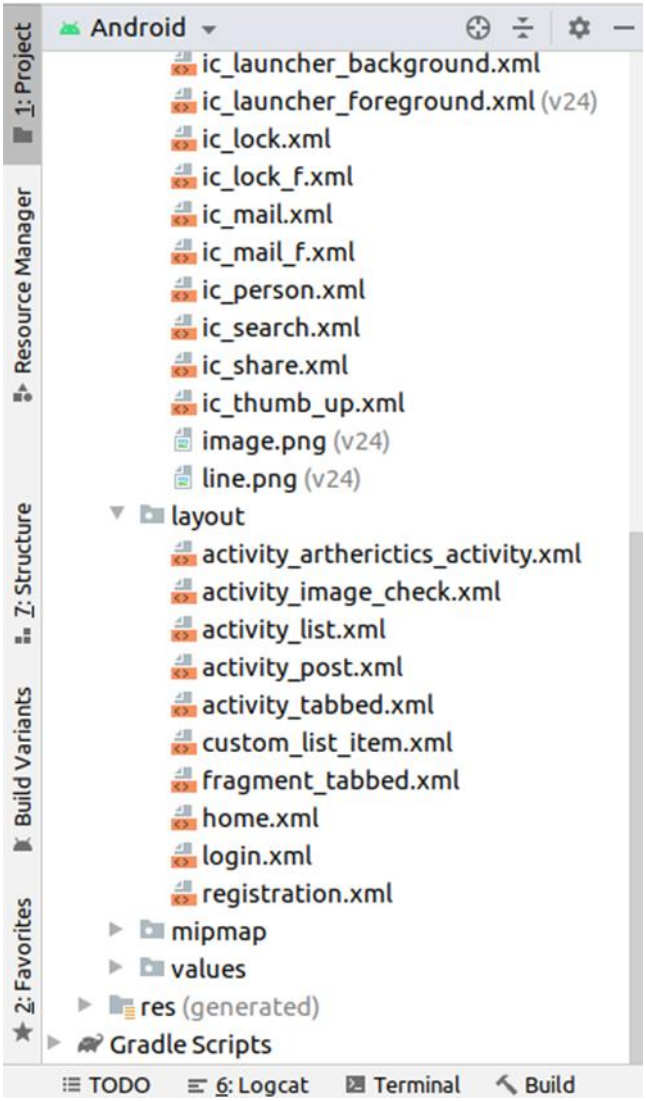

Figure 2.8.1.1.1: Frontend Mobile Folder Structure with layout files

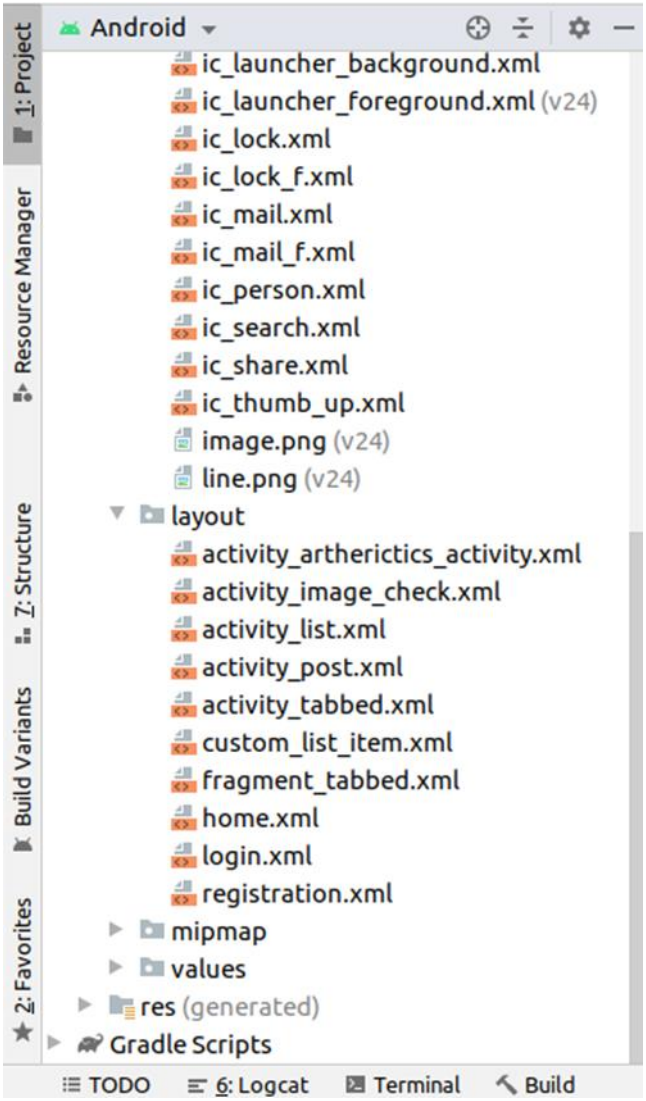

Figure 2.8.1.1.2: Frontend Mobile Folder Structure with java files

2.8.1.2 Backend Implementation

Database Structure

Firebase has been used as the database to store user authentication details (login and registration with account creation) and classified herbal plants details.

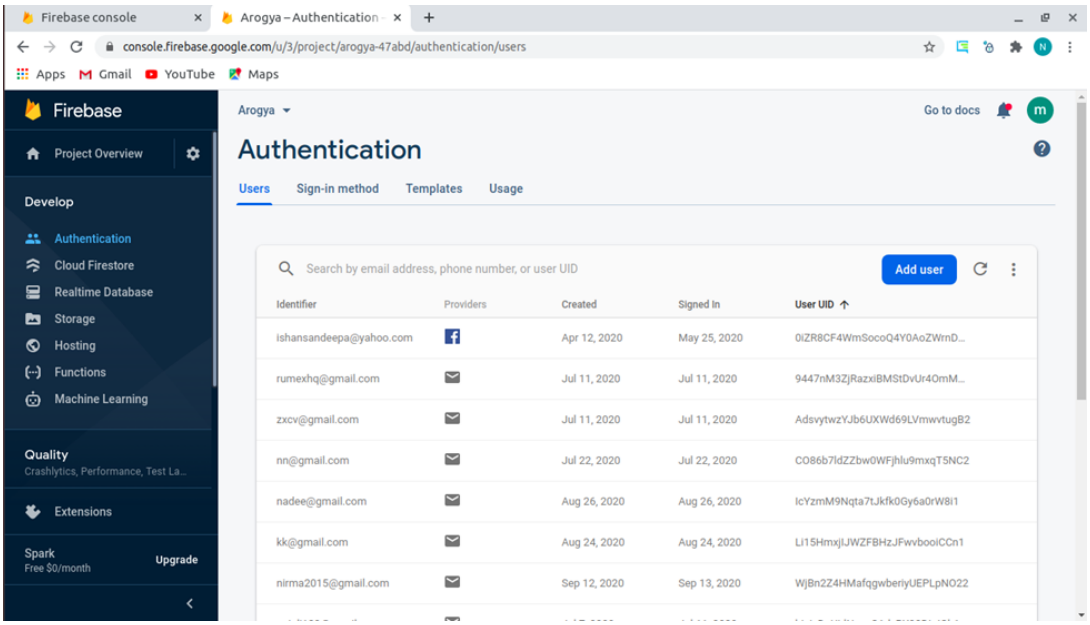

Figure 2.8.1.2.1: Firebase database for user authentication and account creation

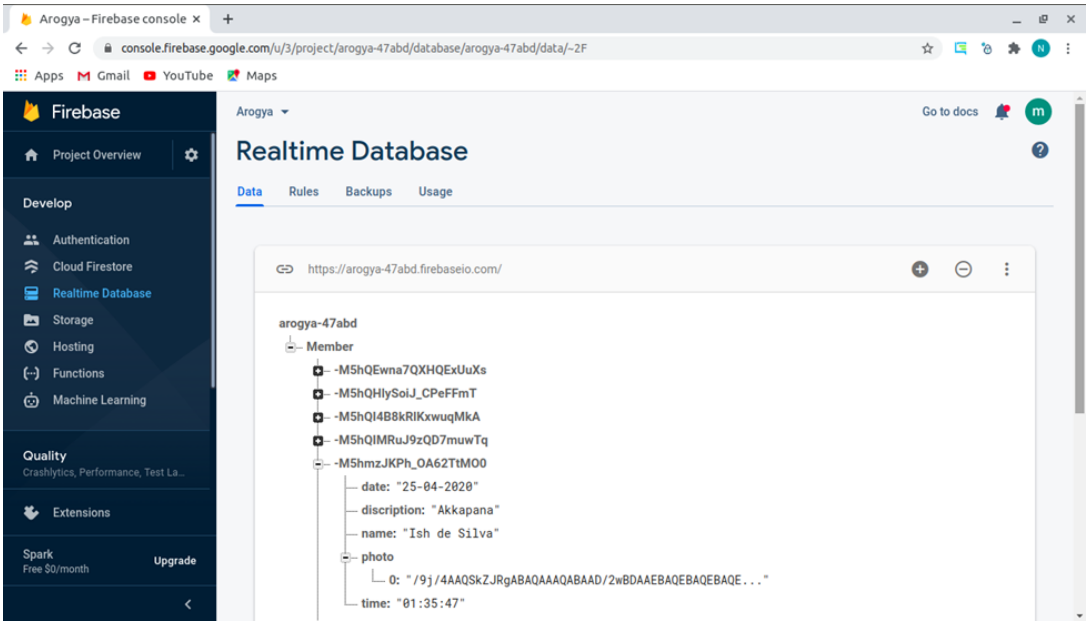

Figure 2.8.1.2.2: Firebase database for uploading posts related to ayurvedic leaves

Cloud Services Structure

Azure has been used as the cloud service for the hosting and deployment of services related with herbal plant classification.

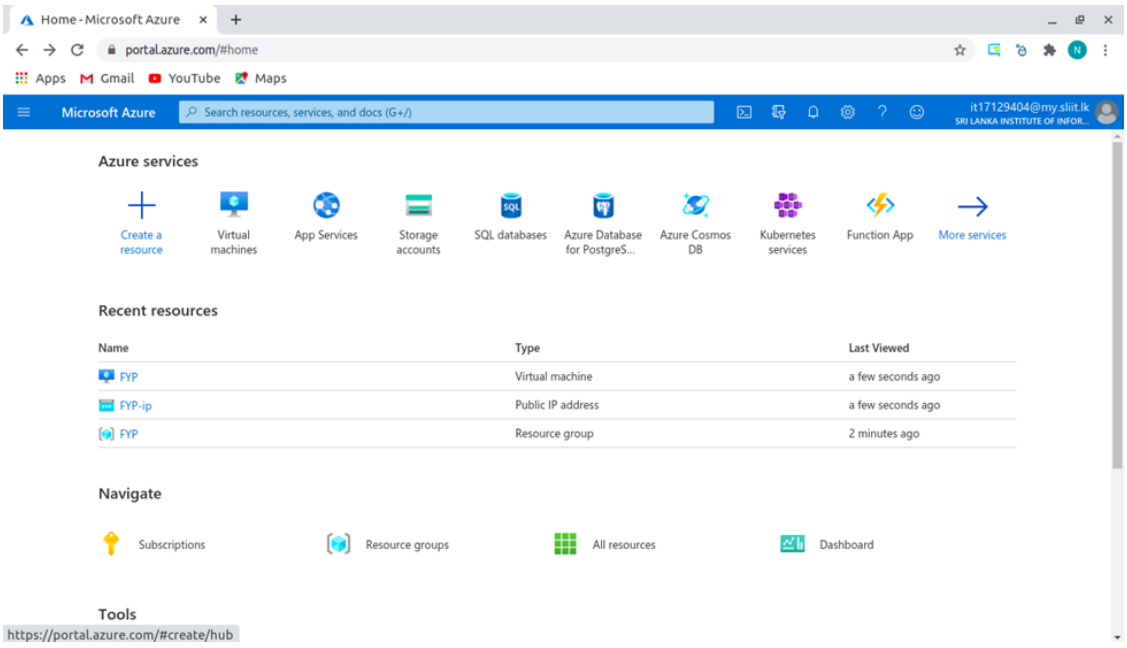

Figure 2.8.1.2.3: Azure cloud resources allocated for services

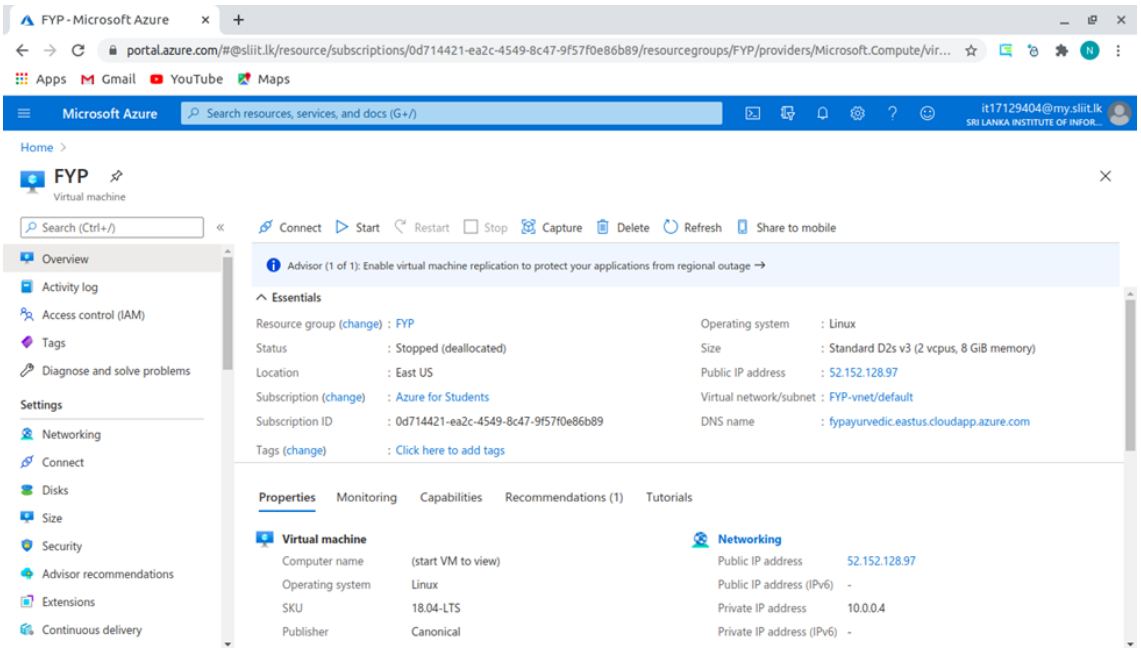

Figure 2.8.1.2.4: Azure virtual machine created for deployment of plant detection and classification

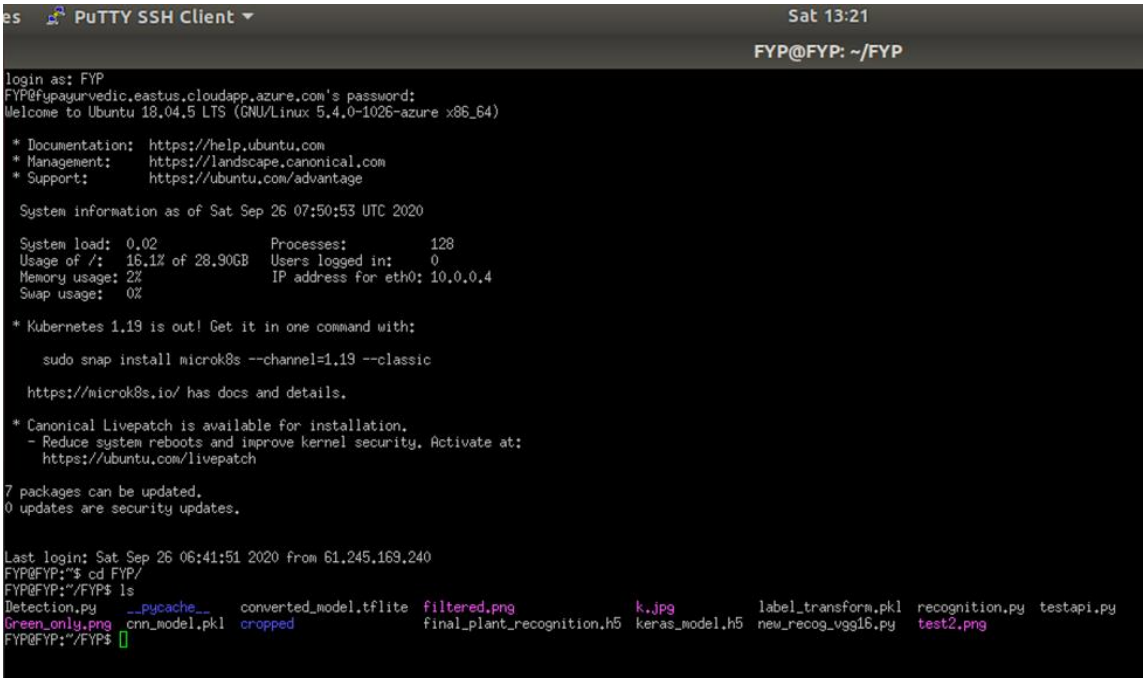

Figure 2.8.1.2.5: Model with highest accuracy hosted and deployed in the virtual machine

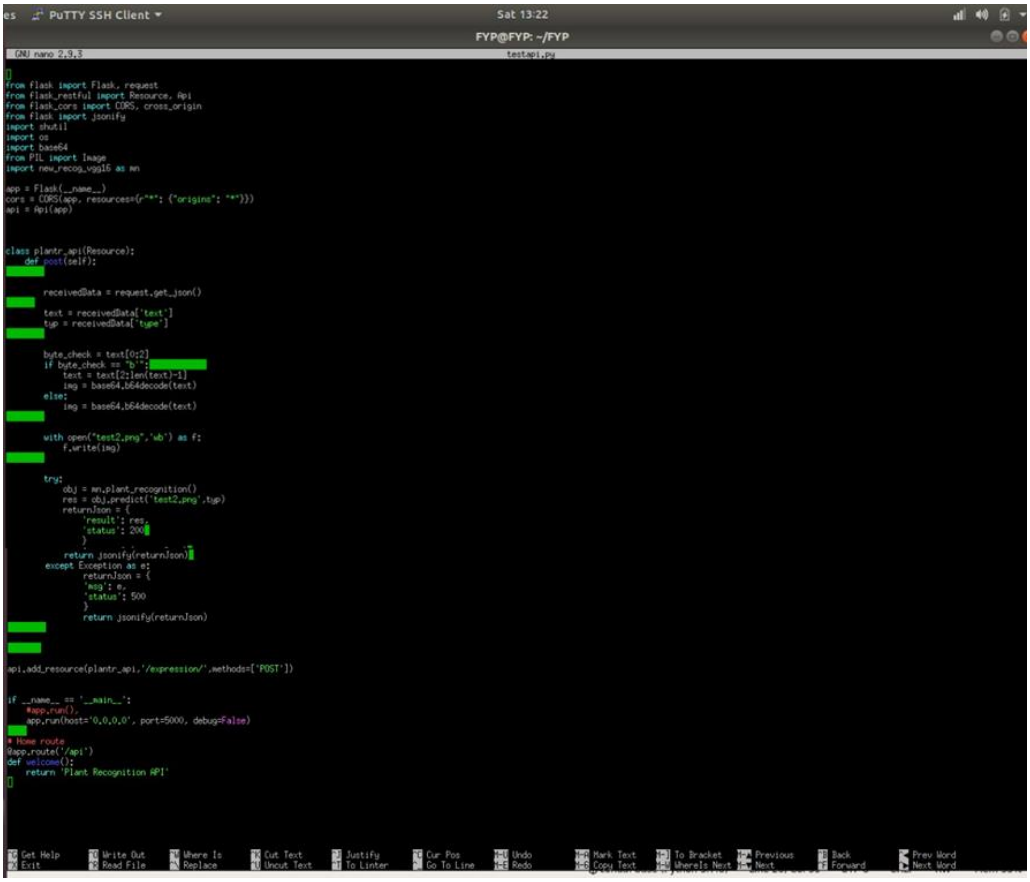

Figure 2.8.1.2.6: Code snippet with integrated models in the virtual machine

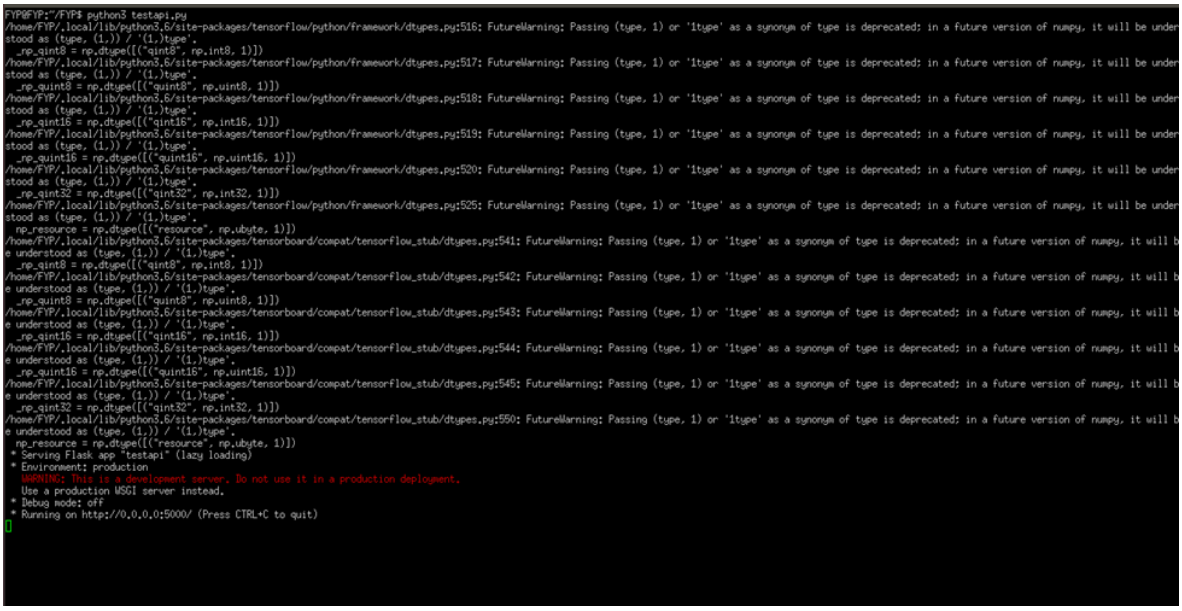

Figure 2.8.1.2.7: Backend running as a service in the virtual machine

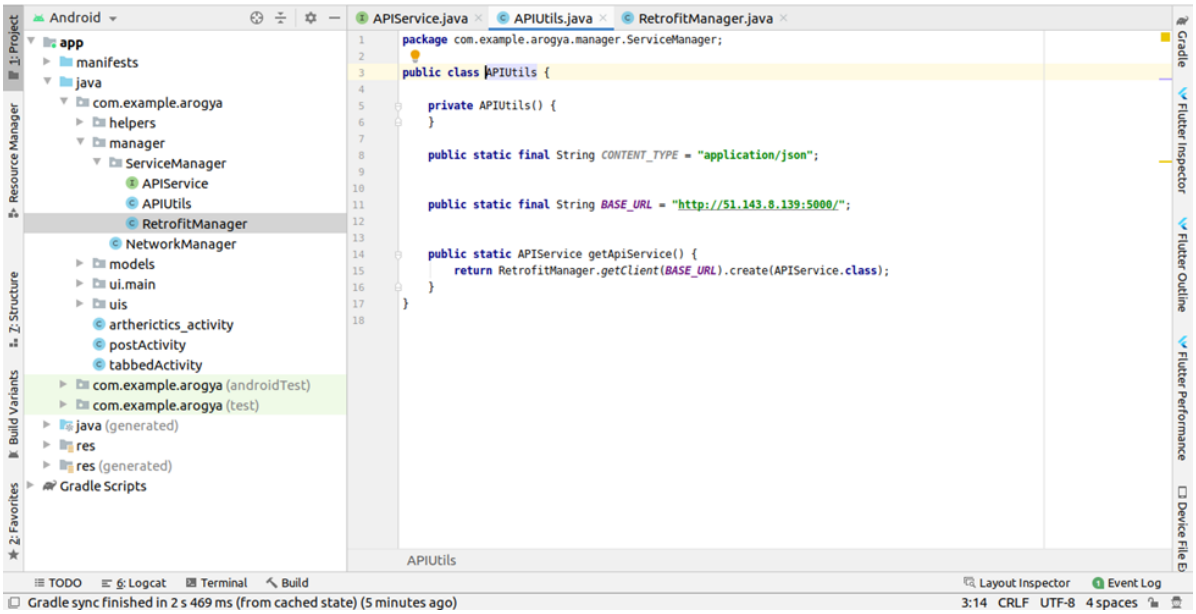

Figure 2.8.1.2.8: Code snippet to connect frontend mobile client with backend API in cloud

2.8.1.3 Code Implementations for the research part

Used Technologies

- Python, Keras, Tensorflow, TensorFlow-Backend installed Anaconda Navigator
- **IDE:** Jupiter Notebook and Google Colab

The code implementation part of the specific research component starts with the application of transfer learning based on deep Convolutional Neural Networks (CNN), to the collected and annotated dataset from scratch, for processing them using TensorFlow.

Throughout this research, the variants of deep Convolutional Neural Networks used to compare the initial training and testing accuracies included:

- InceptionV3
- MobileNetV2
- InceptionResNetV2
- Xception
- DenseNet121
- ResNet50
- VGG16

I. InceptionV3

1. Created the base model from the InceptionV3 model, which is pretrained on the ImageNet dataset.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3

conv_base=InceptionV3(weights='imagenet',include_top=False,input_shape=(100,100,3))
```

Figure 2.8.1.3.1: Code snippet to get InceptionV3 base model

2. Extracted the features through freezing the convolutional base.

```
conv_base.trainable = False
```

Figure 2.8.1.3.2: Code snippet to freeze InceptionV3 base model

3. Added a global_average_pooling2d layer, which reduced the tendency of overfitting.
4. Added a dropout layer, with a rate of 0.5 for regularization purpose.
5. Added a dense layer for 6-class classification, with the activation function as 'softmax'.

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D, BatchNormalization, Glob
alAveragePooling2D, Concatenate

from tensorflow.python.keras import models
from tensorflow.python.keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(6, activation='softmax'))
```

Figure 2.8.1.3.3: Code snippet to add extra layers in InceptionV3 base

6. Compiled the model before training it - here, 'categorical_crossentropy' was used as the loss function, 'SGD' as the optimizer, (1e-4) as the learning rate, 0.9 as the momentum and 'categorical_accuracy' as metrics.

```
from keras import optimizers, losses, activations, models

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),
              metrics=['categorical_accuracy'])
```

Figure 2.8.1.3.4: Code snippet to compile InceptionV3 newly created model

7. Training and testing datasets were loaded, where training dataset was augmented according to different augmentation techniques, while testing dataset was not augmented due to valid accuracy purposes

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.
apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2
fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train_dir='/content/drive/My Drive/plant/train'
```

```
validation_dir='/content/drive/My Drive/plant/val'
```

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import preprocess_input

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32)

validation_generator = test_datagen.flow_from_directory(
        # This is the target directory
        validation_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32,
        shuffle=False)
```

Figure 2.8.1.3.5: Code snippet to load and generate train and test datasets in InceptionV3

8. Finally, the dataset was trained for 10 epochs, with using the number of steps per epoch training as 50, and the number of steps per epoch testing as 25, including a callback.

```
from tensorflow.python.keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping, ReduceLROnPlateau, TensorBo
ard

checkpoint = ModelCheckpoint(filepath = 'best_InceptionV3.hdf5', monitor='val_loss', save_best_only=True, mode='auto')

early = EarlyStopping(monitor="val_loss", mode="auto", patience=3)

callbacks_list = [checkpoint, early] #early

fit_history = model.fit_generator(
      train_generator,
      steps_per_epoch=50,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=25,
      callbacks=callbacks_list)
```

Figure 2.8.1.3.6: Code snippet to train InceptionV3

II. MobileNetV2

1. Created the base model from the MobileNetV2 model, which is pretrained on the ImageNet dataset.

```
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2

conv_base=MobileNetV2(weights='imagenet',include_top=False,input_shape=(224,224,3))
```

Figure 2.8.1.3.7: Code snippet to get MobileNetV2 base model

2. Extracted the features through freezing the convolutional base.

```
conv_base.trainable = False
```

Figure 2.8.1.3.8: Code snippet to freeze MobileNetV2 base model

3. Added a conv2d layer, with filters=32, kernel_size= (3,3) and activation function as 'relu'.
4. Added a dropout layer, with a rate of 0.2 for regularization purpose.
5. Added a global_average_pooling2d layer, which reduced the tendency of overfitting.
6. Added a dense layer for 6-class classification, with the activation function as 'softmax'.

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D, BatchNormalization, Glob
alAveragePooling2D, Concatenate

from tensorflow.python.keras import models
from tensorflow.python.keras import layers

model = models.Sequential()
model.add(conv_base)

model.add(layers.Conv2D(32, 3, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.GlobalAveragePooling2D())

model.add(layers.Dense(6, activation='softmax'))
```

Figure 2.8.1.3.9: Code snippet to add extra layers in MobileNetV2 base

7. Compiled the model before training it - here, 'categorical_crossentropy' was used as the loss function, (1e-5) as the learning rate, 'Adam' as the optimizer and 'categorical_accuracy' as metrics.

```
from tensorflow.keras import optimizers

Adam=optimizers.Adam(1e-5)

model.compile(loss='categorical_crossentropy',
              optimizer=Adam,
              metrics=['categorical_accuracy'])
```

Figure 2.8.1.3.10: Code snippet to compile MobileNetV2 newly created model

8. Training and testing datasets were loaded, where training dataset was augmented according to different augmentation techniques, while testing dataset was not augmented due to valid accuracy purposes.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.
apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2
fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train_dir='/content/drive/My Drive/plant/train'
```

```
validation_dir='/content/drive/My Drive/plant/val'
```

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import preprocess_input

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32)

validation_generator = test_datagen.flow_from_directory(
        # This is the target directory
        validation_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32,
        shuffle=False)
```

Figure 2.8.1.3.11: Code snippet to load and generate train and test datasets in MobileNetV2

9. Finally, the dataset was trained for 10 epochs, with using the number of steps per epoch training as 50, and the number of steps per epoch testing as 25.

```
fit_history = model.fit_generator(
      train_generator,
      steps_per_epoch=50,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=25)
```

Figure 2.8.1.3.12: Code snippet to train MobileNetV2

III. InceptionResNetV2

1. Created the base model from the InceptionResNetV2 model, which is pretrained on the ImageNet dataset.

```
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2

conv_base=InceptionResNetV2(weights='imagenet',include_top=False,input_shape=(100,100,3))
```

Figure 2.8.1.3.13: Code snippet to get InceptionResNetV2 base model

2. Extracted the features through freezing the convolutional base.

```
conv_base.trainable = False
```

Figure 2.8.1.3.14: Code snippet to freeze InceptionResNetV2 base model

3. Added a flatten layer, which did not affect the batch size.
4. Added a dropout layer, with a rate of 0.5 for regularization purpose.
5. Added a dense layer, with 1x512, with the activation function as ‘relu’.
6. Added a dropout layer, with a rate of 0.5 for regularization purpose.
7. Added a dense layer, with 1x256, with the activation function as ‘relu’ (adding of dense layers was decided by considering the number of classes)
8. Added a dense layer for 6-class classification, with the activation function as ‘softmax’.

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D, BatchNormalization, Glob
alAveragePooling2D, Concatenate

from tensorflow.python.keras import models
from tensorflow.python.keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(6, activation='softmax'))
```

Figure 2.8.1.3.15: Code snippet to add extra layers in InceptionResNetV2

9. Compiled the model before training it - here, ‘categorical_crossentropy’ was used as the loss function, ‘Adam’ as the optimizer, (1e-3) as the learning rate and ‘categorical_accuracy’ as metrics.

```
from keras import optimizers, losses, activations, models

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(1e-3),
              metrics=['categorical_accuracy'])
```

Figure 2.8.1.3.16: Code snippet to compile InceptionResNetV2 newly created model

10. Training and testing datasets were loaded, where training dataset was augmented according to different augmentation techniques, while testing dataset was not augmented due to valid accuracy purposes.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.
apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2
fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train_dir='/content/drive/My Drive/plant/train'
```

```
validation_dir='/content/drive/My Drive/plant/val'
```

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import preprocess_input

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32)

validation_generator = test_datagen.flow_from_directory(
        # This is the target directory
        validation_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32,
        shuffle=False)
```

Figure 2.8.1.3.17: Code snippet to load and generate train and test datasets in InceptionResNetV2

11. Finally, the dataset was trained for 10 epochs, with using the number of steps per epoch training as 50, and the number of steps per epoch testing as 25, including a callback.

```
from tensorflow.python.keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping, ReduceLROnPlateau, TensorBo
ard

checkpoint = ModelCheckpoint(filepath = 'best_InceptionResNetV2.hdf5', monitor='val_loss', save_best_only=True, mode='auto')

early = EarlyStopping(monitor="val_loss", mode="auto", patience=3)

callbacks_list = [checkpoint, early] #early

fit_history = model.fit_generator(
      train_generator,
      steps_per_epoch=50,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=25,
      callbacks=callbacks_list)
```

Figure 2.8.1.3.18: Code snippet to train InceptionResNetV2

IV. Xception

1. Created the base model from the Xception model, which is pretrained on the ImageNet dataset.

```
from tensorflow.keras.applications.xception import Xception

conv_base=Xception(weights='imagenet',include_top=False,input_shape=(100,100,3))
```

Figure 2.8.1.3.19: Code snippet to get Xception base model

2. Extracted the features through freezing the convolutional base.

```
conv_base.trainable = False
```

Figure 2.8.1.3.20: Code snippet to freeze Xception base model

3. Added a global_average_pooling2d layer, which reduced the tendency of overfitting.
4. Added a dense layer, with 1x512, with the activation function as ‘relu’ (adding of dense layers was decided by considering the number of classes)
5. Added a dense layer for 6-class classification, with the activation function as ‘softmax’.

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D, BatchNormalization, Glob
alAveragePooling2D, Concatenate

from tensorflow.python.keras import models
from tensorflow.python.keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(6, activation='softmax'))
```

Figure 2.8.1.3.21: Code snippet to add extra layers in Xception base model

6. Compiled the model before training it - here, ‘categorical_crossentropy’ was used as the loss function, ‘Nadam’ as the optimizer and ‘categorical_accuracy’ as metrics.

```
from keras import optimizers, losses, activations, models

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Nadam(),
              metrics=['categorical_accuracy'])
```

Figure 2.8.1.3.22: Code snippet to compile Xception newly created model

7. Training and testing datasets were loaded, where training dataset was augmented according to different augmentation techniques, while testing dataset was not augmented due to valid accuracy purposes.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.
apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2
fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train_dir='/content/drive/My Drive/plant/train'
```

```
validation_dir='/content/drive/My Drive/plant/val'
```

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import preprocess_input

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32)

validation_generator = test_datagen.flow_from_directory(
        # This is the target directory
        validation_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32,
        shuffle=False)
```

Figure 2.8.1.3.23: Code snippet to load and generate train and test datasets in Xception

8. Finally, the dataset was trained for 10 epochs, with using the number of steps per epoch training as 10, and the number of steps per epoch testing as 25, including a callback.

```
from tensorflow.python.keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping, ReduceLROnPlateau, TensorBo
ard

checkpoint = ModelCheckpoint(filepath = 'best_Xception.hdf5', monitor='val_loss', save_best_only=True, mode='auto')

early = EarlyStopping(monitor="val_loss", mode="auto", patience=5)

callbacks_list = [checkpoint, early] #early

fit_history = model.fit_generator(
      train_generator,
      steps_per_epoch=50,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=25,
      callbacks=callbacks_list)
```

Figure 2.8.1.3.24: Code snippet to train Xception

V. DenseNet121

1. Created the base model from the DenseNet121 model, which is pretrained on the ImageNet dataset.

```
from tensorflow.python.keras.applications.densenet import DenseNet121

conv_base=DenseNet121(weights='imagenet',include_top=False,input_shape=(100,100,3))
```

Figure 2.8.1.3.25: Code snippet to get DenseNet121 base model

2. Extracted the features through freezing the convolutional base.

```
conv_base.trainable = False
```

Figure 2.8.1.3.26: Code snippet to freeze DenseNet121 base model

3. Added a global_average_pooling2d layer, which reduced the tendency of overfitting.
4. Added a dense layer for 6-class classification, with the activation function as ‘softmax’.

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D, BatchNormalization, Glob
alAveragePooling2D, Concatenate

from tensorflow.python.keras import models
from tensorflow.python.keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(6, activation='softmax'))
```

Figure 2.8.1.3.27: Code snippet to add extra layers in DenseNet121 base model

5. Compiled the model before training it - here, ‘categorical_crossentropy’ was used as the loss function, ‘Adam’ as the optimizer and ‘categorical_accuracy’ as metrics.

```
from tensorflow.keras import optimizers

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['categorical_accuracy'])
```

Figure 2.8.1.3.28: Code snippet to compile DenseNet121 newly created model

6. Training and testing datasets were loaded, where training dataset was augmented according to different augmentation techniques, while testing dataset was not augmented due to valid accuracy purposes.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.
apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2
fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train_dir='/content/drive/My Drive/plant/train'
```

```
validation_dir='/content/drive/My Drive/plant/val'
```

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import preprocess_input

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32)

validation_generator = test_datagen.flow_from_directory(
        # This is the target directory
        validation_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32,
        shuffle=False)
```

Figure 2.8.1.3.29: Code snippet to load and generate train and test datasets in DenseNet121

7. Finally, the dataset was trained for 10 epochs, with using the number of steps per epoch training as 50, and the number of steps per epoch testing as 25.

```
from tensorflow.python.keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping, ReduceLROnPlateau, TensorBo
ard

fit_history = model.fit_generator(
      train_generator,
      steps_per_epoch=50,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=25)
```

Figure 2.8.1.3.30: Code snippet to train DenseNet121

VI. ResNet50

1. Created the base model from the ResNet50 model, which is pretrained on the ImageNet dataset.

```
from tensorflow.keras.applications import ResNet50

# Define transfer learning network model

conv_base = ResNet50(include_top = False, pooling = RESNET50_POOLING_AVERAGE, weights='imagenet', input_shape=(100,100,3))
```

Figure 2.8.1.3.31: Code snippet to get ResNet50 base model

2. Extracted the features through freezing the convolutional base, while making conv5_block1_1_conv layer unfreezed.

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'conv5_block1_1_conv':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Figure 2.8.1.3.32: Code snippet to freeze and unfreeze ResNet50 base model

3. Added a dense layer for 6-class classification, with the activation function as 'softmax'.

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras import models
from tensorflow.python.keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Dense(NUM_CLASSES, activation = DENSE_LAYER_ACTIVATION))
```

Figure 2.8.1.3.33: Code snippet to add extra layers in ResNet50 base model

4. Compiled the model before training it - here, ‘accuracy’ was used as the loss function, 0.01 as the learning rate, (1e-6) as the decay, 0.9 as the momentum, ‘SGD’ as the optimizer and ‘categorical_crossentropy’ as metrics.

```
# Compile transfer learning model

from tensorflow.keras import optimizers

sgd = optimizers.SGD(lr = 0.01, decay = 1e-6, momentum = 0.9, nesterov = True)
model.compile(optimizer = sgd, loss = OBJECTIVE_FUNCTION, metrics = LOSS_METRICS)
```

Figure 2.8.1.3.34: Code snippet to compile ResNet50 newly created model

5. Training and testing datasets were loaded, where training dataset was augmented according to different augmentation techniques, while testing dataset was not augmented due to valid accuracy purposes.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.
apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2
fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train_dir='/content/drive/My Drive/plant/train'
```

```
validation_dir='/content/drive/My Drive/plant/val'
```

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import preprocess_input

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32)

validation_generator = test_datagen.flow_from_directory(
        # This is the target directory
        validation_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32,
        shuffle=False)
```

Figure 2.8.1.3.35: Code snippet to load and generate train and test datasets in ResNet50

6. Finally, the dataset was trained for 10 epochs, with using the number of steps per epoch training as 50, and the number of steps per epoch testing as 25.

```
from tensorflow.python.keras.callbacks import EarlyStopping, ModelCheckpoint

cb_early_stopper = EarlyStopping(monitor = 'val_loss', patience = EARLY_STOP_PATIENCE)
cb_checkpointer = ModelCheckpoint(filepath = 'best_Res50.hdf5', monitor = 'val_loss', save_best_only = True, mode = 'auto')
```

```
fit_history = model.fit_generator(
        train_generator, steps_per_epoch=STEPS_PER_EPOCH_TRAINING, epochs = NUM_EPOCHS,
        validation_data=validation_generator, validation_steps=STEPS_PER_EPOCH_VALIDATION,
        callbacks=[cb_checkpointer, cb_early_stopper])
```

Figure 2.8.1.3.36: Code snippet to train ResNet50

VII. VGG16

1. Created the base model from the VGG16 model, which is pretrained on the ImageNet dataset.

```
from tensorflow.python.keras.applications import VGG16

conv_base=VGG16(weights='imagenet',include_top=False,input_shape=(100,100,3))
```

Figure 2.8.1.3.37: Code snippet to get VGG16 base model

2. Extracted the features through freezing the convolutional base.

```
conv_base.trainable = False
```

Figure 2.8.1.3.38: Code snippet to freeze VGG16 base model

3. Added a flatten layer, which did not affect the batch size.
4. Added a dense layer, with 1x256, with the activation function as ‘relu’ (adding of dense layers was decided by considering the number of classes)
5. Added a dense layer for 6-class classification, with the activation function as ‘softmax’.

```
from tensorflow.python.keras import models
from tensorflow.python.keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(6, activation='softmax'))
```

Figure 2.8.1.3.39: Code snippet to add extra layers in VGG16 base model

6. Compiled the model before training it - here, ‘categorical_crossentropy’ was used as the loss function, ‘RMSprop’ as the optimizer, (1e-4) as the learning rate and ‘categorical_accuracy’ as metrics.

```
from tensorflow.python.keras import optimizers

model.compile(loss='categorical_crossentropy',optimizer=optimizers.RMSpropfruit_image_classification_using vgg16_transfe(lr=1e-
4),metrics=['categorical_accuracy'])
```

Figure 2.8.1.3.40: Code snippet to compile VGG16 newly created model

7. Training and testing datasets were loaded, where training dataset was augmented according to different augmentation techniques, while testing dataset was not augmented due to valid accuracy purposes.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.
apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2
fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train_dir='/content/drive/My Drive/plant/train'
```

```
validation_dir='/content/drive/My Drive/plant/val'
```

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import preprocess_input

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32)

validation_generator = test_datagen.flow_from_directory(
        # This is the target directory
        validation_dir,
        # All images will be resized to 100x100
        target_size=(100, 100),
        batch_size=32,
        shuffle=False)
```

Figure 2.8.1.3.41: Code snippet to load and generate train and test datasets in VGG16

8. Finally, the dataset was trained for 100 epochs, with using the number of steps per epoch training as 100, and the number of steps per epoch testing as 50.

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

Figure 2.8.1.3.42: Code snippet to train VGG16

After training for different number of epochs, number of steps per epoch training and number of steps per epoch testing; the testing accuracy was calculated for each of the above-mentioned CNN variant using the below equation:

```
steps_test=212/32
```

```
print(steps_test)
```

```
6.625
```

```
result = model.evaluate_generator(validation_generator, steps=steps_test)
print("Test-set classification accuracy: {0:.2%}".format(result[1]))
```

Figure 2.8.1.3.43: Code snippet to calculate testing accuracy

In addition, for collecting the history returned from training each CNN model and for easy reference of the performance from each, a plot of accuracy on the training and validation datasets over training epochs, as well as a plot of loss on the training and validation datasets over training epochs have been visualized and plotted with the following code.

```
import matplotlib.pyplot as plt
```

```
acc=history.history['categorical_accuracy']
val_acc=history.history['val_categorical_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(1,len(acc)+1)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.plot(epochs, acc,  label='Training acc')
plt.plot(epochs, val_acc, label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, label='Training loss')
plt.plot(epochs, val_loss, label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Figure 2.8.1.3.44: Code snippet to generate accuracy and loss graphs against epochs trained

2.8.2 Testing

Testing plays a major role in this research. The research is related to the health sector, due to the interconnectivity with Ayurveda. So, the implemented functions must be accurate since the software deals with the lives of people and affects the ingredients of medical recipes.

The initial process of the mobile application was detecting the constitution based on a photo that incorporates image processing. The image was uploaded for distinguishing the plant part (leaf/fruit/root) needed to be classified. There is a possibility where analysis can lead to inaccurate information if the conditions that affect a photo changes. Therefore, the testing of image processing part should be tested under different conditions of lighting, the angle of capture, brightness, zoom, rotation and the distance

between the object and capture device (mobile camera). All these different conditions which varies from each other were obtained through data augmentation techniques. In summary, testing phase is set to check the efficiency, effectiveness and reliability of the system and the final outputs, whether it satisfies the project requirements.

Unit Testing

The purpose of executing unit tests is to make sure, all the independent components work as expected according to the working plan. The system was divided into small components such as sign up (account creation), sign in (log in) and plant classification, and tested in order to get the results. They were compared to see whether the actual results were as similar as the expected results. The system was divided into modules accordingly. Finally, a verification process was run to check whether the system met its specifications as discussed by the team at the beginning.

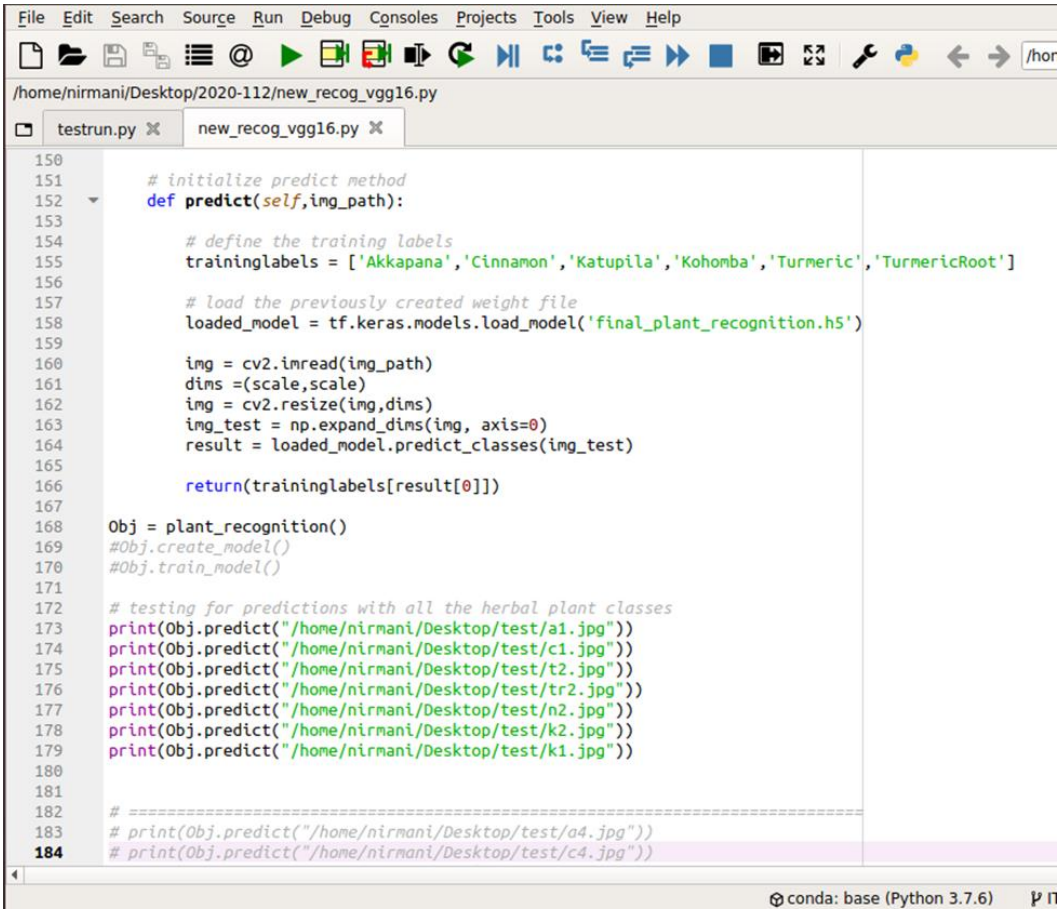

Figure 2.8.2.1: Code snippet of the method for unit testing of plant classification functionality

Integration Testing

Individual components such as system authentication, object detection and plant classification functionalities were combined and tested into a single module to check the quality of the overall product and make sure, there was no other internal conflicts on the combination of whole product.

Object detection component was followed by the plant classification component. They were integrated as:

- ✓ First either a plant image captured through the mobile camera or uploaded from gallery was detected successfully whether it was a leaf/root/fruit from the object detection functionality.
- ✓ Then those detected objects were saved to a separate folder as images.

- ✓ After that, the particular folder path was given for the plant classification functionality, so that the input for that functionality was the image of detected object, which was the output from the object detection functionality.
- ✓ Likewise, those 2 main components were integrated and tested successfully.

Finally, the information summarizer came into picture as it produced the detailed summary report on the classified herbal plant. Integration testing was done individually by the group members according to their component scope.

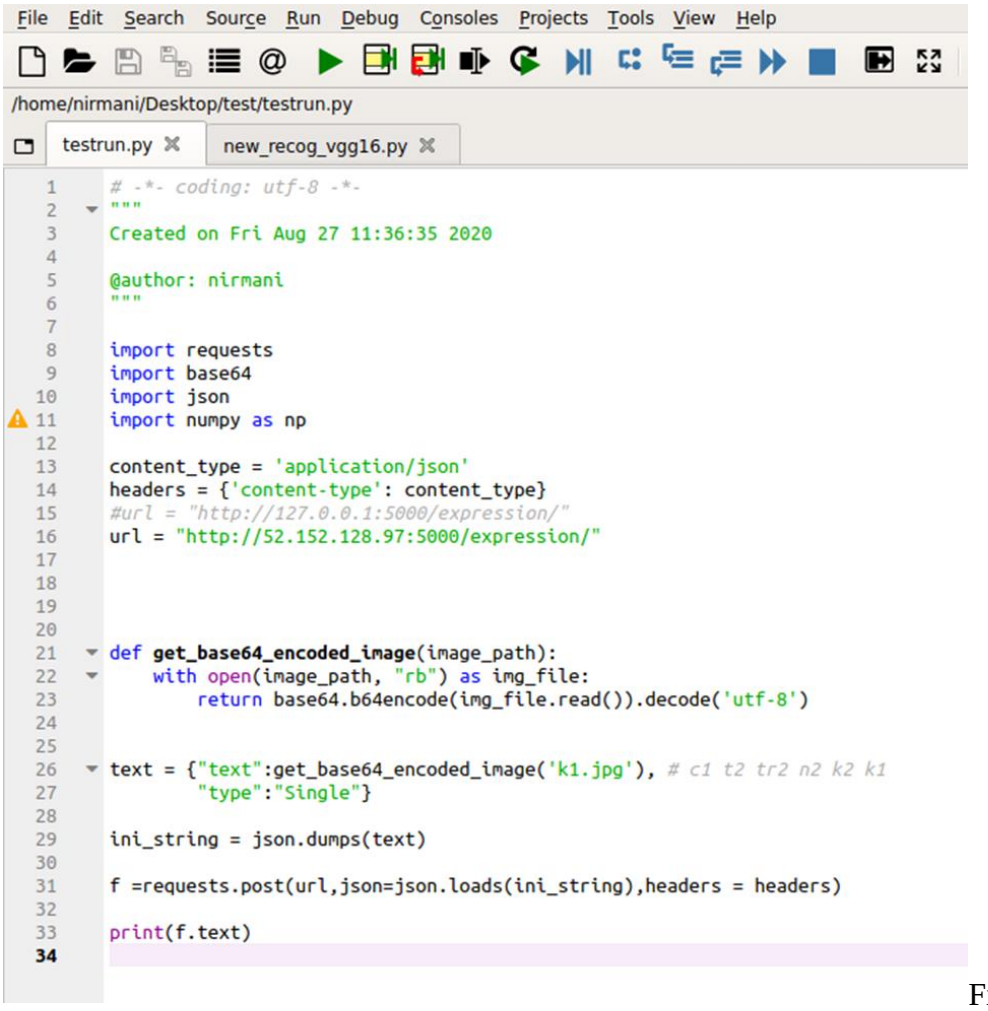

Figure 2.8.2.2: Code snippet of the class for integration testing of plant detection and classification functionalities with the backend API

System Testing

Whole system was tested according to the specification, is said to be a System Integration. This makes sure, the system to be compatible with all the other modules and work as the execution plan without arising any issues. This is more like, black box testing type of testing. If there is any bugs or issues, the overall team has to take the responsibility and relaunch the application with the needed modifications. System testing was done successfully for the whole product by integrating all the API s and bugs were resolved accordingly.

User Acceptance Testing

This is the phase, where the customer interacts with the developed trial product. The product will be tested by the customer to make sure whether it reaches the customer functional requirements. They will be given a demo version of the product, so that, they can test it. If the user accepts the products without making any issues and satisfied, that is the end of the testing life cycle. Otherwise, the team has to do some other modifications accordingly, and the cycle goes on until the user gets satisfied.

Test cases for Arogya mobile application

a. **Account creation**

Table 2.8.2.1: Test case for account creation with valid data

Test Case ID	TU001
Test Scenario	Check user account creation with valid data
Precondition	User should not have a previously created user account with the same email address.

Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Tap on “Register Here” 3. View the user registration page. 4. Enter a valid email address. 5. Enter a valid password. 6. Tap of the button “Register”.
Test Data	Email Address: nirma2015@gmail.com Password: KFD4ert1@#$
Expected Results	User should be registered to the system by creating a new account with the given credentials.
Actual Results	User was successfully registered to the system and given a successful message as “User Created Successfully”
Pass/Fail	**Pass**

Table 2.8.2.2: Test case for account creation with invalid data

Test Case ID	TU002
Test Scenario	Check user account creation with invalid data
Precondition	User should not have a previously created user account with the same email address.
Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Tap on “Register Here” 3. View the user registration page. 4. Enter an invalid email address. 5. Enter a valid password. 6. Tap of the button “Register”.
Test Data	Email Address: nirma2015123gmail.com Password: KFD4ert1@#$
Expected Results	User should be given an error message with real time validation as “enter a valid

	email address", and registration should be unsuccessful.
Actual Results	User was given an error message with real time validation as "enter a valid email address", and registration was unsuccessful.
Pass/Fail	**Pass**

b. Login to the application

Table 2.8.2.3: Test case for user login with valid data

Test Case ID	TU003
Test Scenario	Check user login with valid data
Precondition	User should have an account.
Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Enter the correct email address. 3. Enter the correct password. 4. Tap of the button "Login".
Test Data	Email Address: nirma2015@gmail.com Password: KFD4ert1@#$
Expected Results	User should be able to login to the system successfully and redirected to the home page.
Actual Results	User was given a successful message as "Login Successful" and was redirected to the home page
Pass/Fail	**Pass**

Table 2.8.2.4: Test case for user login with invalid data

Test Case ID	TU004
Test Scenario	Check user login with invalid data
Precondition	User should have an account.

Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Enter the correct email address. 3. Enter an incorrect password. 4. Tap of the button “Login”.
Test Data	Email Address: nirma2015@gmail.com Password: KFD4ert1$
Expected Results	User should be given an error message as “incorrect email address or password”, and login should be unsuccessful.
Actual Results	User was given an error message as “incorrect email address or password”, and login was unsuccessful.
Pass/Fail	**Pass**

c. Plant classification

Table 2.8.2.5: Test case for accurate plant classification with a camera-captured image

Test Case ID	TU005
Test Scenario	Check the plant classification function by a camera captured image
Precondition	User should have an account.
Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Enter the correct email address. 3. Enter the correct password. 4. Tap of the button “Login”. 5. View the home page 6. Tap on the “camera” icon in the toolbar

	7. Capture an image of an Akkapana leaf 8. Tap on the button “Classify the herb”.
Test Data	Email Address: nirma2015@gmail.com Password: KFD4ert1@#$ Input image: a camera captured Akkapana leaf
Expected Results	The name of the classified plant should be displayed as “Akkapana”, and a detailed description of the particular species should be visualized to the user.
Actual Results	The name of the classified plant was displayed as “Akkapana”, and a detailed description of the particular species was visualized to the user.
Pass/Fail	**Pass**

Table 2.8.2.6: Test case for inaccurate plant classification with a camera-captured image

Test Case ID	TU006
Test Scenario	Check the plant classification function by a camera captured image
Precondition	User should have an account.
Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Enter the correct email address. 3. Enter the correct password. 4. Tap of the button “Login”. 5. View the home page 6. Tap on the “camera” icon in the toolbar 7. Capture an image of a Katupila leaf

	8. Tap on the button “Classify the herb”.
Test Data	Email Address: nirma2015@gmail.com Password: KFD4ert1@#$ Input image: a camera captured Katupila leaf
Expected Results	The name of the classified plant should be displayed as “Katupila”, and a detailed description of the particular species should be visualized to the user.
Actual Results	The name of the classified plant was displayed as “Cinnamon”, and a detailed description of the particular species was visualized to the user.
Pass/Fail	**Fail**

Table 2.8.2.7: Test case for accurate plant classification with an image uploaded from gallery

Test Case ID	TU007
Test Scenario	Check the plant classification function by an uploaded image from phone gallery.
Precondition	User should have an account.
Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Enter the correct email address. 3. Enter the correct password. 4. Tap of the button “Login”. 5. View the home page 6. Tap on the “from gallery” icon in the toolbar 7. Upload an image of a turmeric leaf 8. Tap on the button “Classify the herb”.
Test Data	Email Address: nirma2015@gmail.com Password: KFD4ert1@#$ Input image: a camera captured turmeric digested root

Expected Results	The name of the classified plant should be displayed as “Turmeric Root”, and a detailed description of the particular species should be visualized to the user.
Actual Results	The name of the classified plant was displayed as “Turmeric Root”, and a detailed description of the particular species was visualized to the user.
Pass/Fail	**Pass**

Table 2.8.2.8: Test case for inaccurate plant classification with an image uploaded from gallery

Test Case ID	TU008
Test Scenario	Check the plant classification function by an uploaded image from phone gallery.
Precondition	User should have an account.
Assumption	User has installed the app in his android mobile device and has internet connectivity
Test Steps	1. Tap on the Arogya icon and open the mobile app to view the login page. 2. Enter the correct email address. 3. Enter the correct password. 4. Tap of the button “Login”. 5. View the home page 6. Tap on the “from gallery” icon in the toolbar 7. Upload an image of a turmeric leaf 8. Tap on the button “Classify the herb”.
Test Data	Email Address: nirma2015@gmail.com Password: KFD4ert1@#$ Input image: a camera captured turmeric leaf
Expected Results	The name of the classified plant should be displayed as “Turmeric”, and a detailed description of the particular species should be visualized to the user.
Actual Results	The name of the classified plant was displayed as “Cinnamon”, and a detailed

	description of the particular species was visualized to the user.
Pass/Fail	**Fail**

2.9 Tools and Technologies

1. Python
2. Keras
3. TensorFlow
4. TensorFlow Lite
5. Jupiter Notebook
6. Google Colab
7. Android
8. Android Studio
9. Firebase – for the database
10. Azure – for backend API deployment

Research Area – Deep Learning Algorithms (deep neural network architectures based on transfer learning) and Image Processing Techniques

3. RESULTS AND DISCUSSION

3.1 Results

Dataset

When taken as overall, a set of 1348 leaf images were retained, where 883, 235 and 230 images were used for training, validation, and testing purposes, respectively. These were the original images of the dataset those were not augmented. The class distribution of the samples over training, validation and testing folds are shown in the following table.

Table 3.1.1: Class distribution of samples over training, testing and validation folds

Herbal plant class	**Train**	**Validation**	**Test**	**Total**
Akkapana leaf	133	41	40	**214**
Cinnamon leaf	148	42	40	**230**
Katupila leaf with fruit	167	62	60	**289**
Kohomba	169	40	40	**249**
Turmeric leaf	78	40	40	**158**
Turmeric digested root	188	10	10	**208**
Total	**883**	**235**	**230**	**1348**

While retraining the dataset in order to obtain a higher accuracy from the selected best accurate CNN model, only the training dataset was augmented, because augmenting validation and testing datasets is not valid in order to obtain the most reliable value of accuracy.

Experimental Outcomes

The primary results obtained from transfer learning based on the trained deep CNN architectures can be illustrated as follows. Those results indicate the model summary, final testing accuracy, model performance accuracy against training epochs and model performance loss against training epochs, for each model which have been trained and tested throughout this research.

I. InceptionV3

- Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Model)         (None, 1, 1, 2048)        21802784
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 12294
=================================================================
Total params: 21,815,078
Trainable params: 12,294
Non-trainable params: 21,802,784
```

Figure 3.1.1: InceptionV3 model summary

- Finalized testing accuracy: **56.76%**

- Model performance accuracy graph

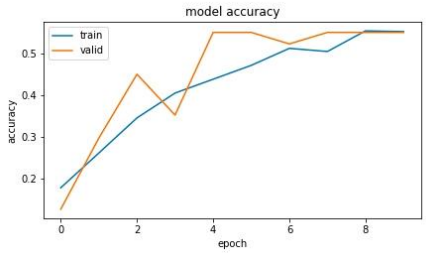

Figure 3.1.2: InceptionV3 accuracy performance

- Model performance loss graph

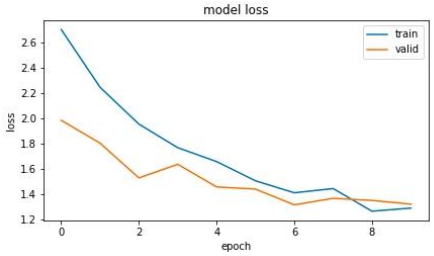

Figure 3.1.3: InceptionV3 loss performance

II. MobileNetV2

- Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Model) (None, 7, 7, 1280)        2257984
_________________________________________________________________
conv2d (Conv2D)              (None, 5, 5, 32)          368672
_________________________________________________________________
dropout (Dropout)            (None, 5, 5, 32)          0
_________________________________________________________________
global_average_pooling2d (Gl (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 6)                 198
=================================================================
Total params: 2,626,854
Trainable params: 368,870
Non-trainable params: 2,257,984
```

Figure 3.1.4: MobileNetV2 model summary

- Finalized testing accuracy: **63.58%**

- Model performance accuracy graph

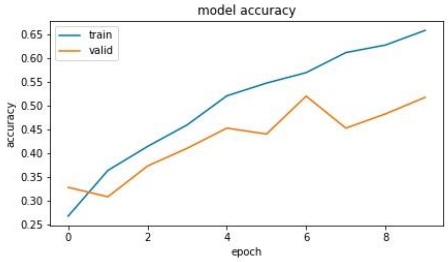

Figure 3.1.5: MobileNetV2 accuracy performance

- Model performance loss graph

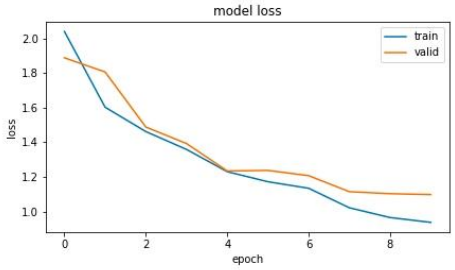

Figure 3.1.6: MobileNetV2 loss performance

III. InceptionResNetV2

- Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_resnet_v2 (Model)  (None, 1, 1, 1536)        54336736
_________________________________________________________________
flatten (Flatten)            (None, 1536)              0
_________________________________________________________________
dropout (Dropout)            (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 512)               786944
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 1542
=================================================================
Total params: 55,256,550
Trainable params: 919,814
Non-trainable params: 54,336,736
```

Figure 3.1.7: InceptionResNetV2 model summary

- Finalized testing accuracy: **82.77%**
- Model performance accuracy graph

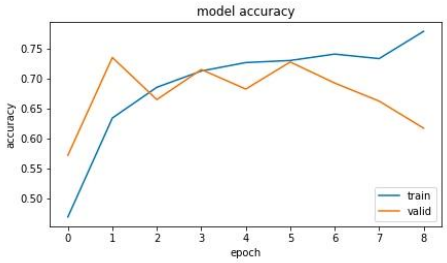

Figure 3.1.8: InceptionResNetV2 accuracy performance

- Model performance loss graph

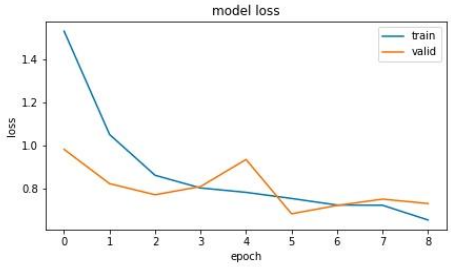

Figure 3.1.9: InceptionResNetV2 loss performance

IV. Xception

- Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Model)             (None, 3, 3, 2048)        20861480
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 3078
=================================================================
Total params: 21,913,646
Trainable params: 1,052,166
Non-trainable params: 20,861,480
```

Figure 3.1.10: Xception model summary

- Finalized testing accuracy: **86.01%**
- Model performance accuracy graph

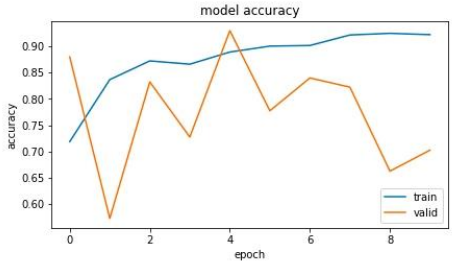

Figure 3.1.11: Xception accuracy performance

- Model performance loss graph

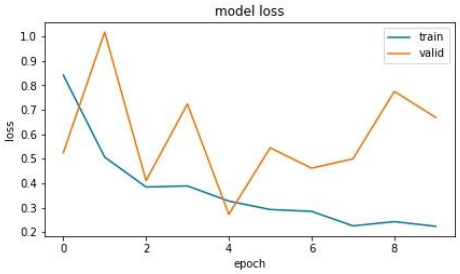

Figure 3.1.12: Xception loss performance

V. DenseNet121

- Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Model)          (None, 3, 3, 1024)        7037504
_________________________________________________________________
global_average_pooling2d (Gl (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 6150
=================================================================
Total params: 7,043,654
Trainable params: 6,150
Non-trainable params: 7,037,504
_________________________________________________________________
```

Figure 3.1.13: DenseNet121 model summary

- Finalized testing accuracy: **89.12%**
- Model performance accuracy graph

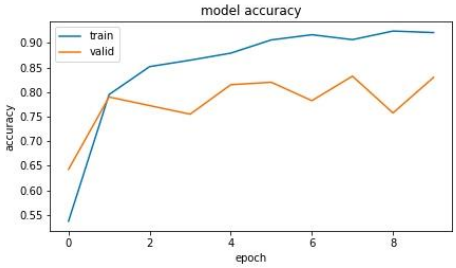

Figure 3.1.14: DenseNet121 accuracy performance

- Model performance loss graph

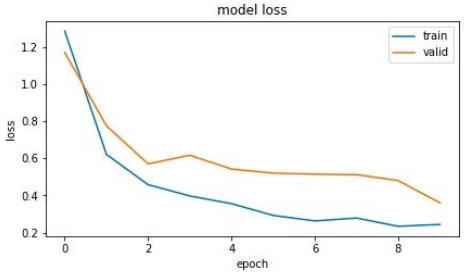

Figure 3.1.15: DenseNet121 loss performance

VI. ResNet50

- Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 2048)              23587712
_________________________________________________________________
dense (Dense)                (None, 6)                 12294
=================================================================
Total params: 23,600,006
Trainable params: 14,988,294
Non-trainable params: 8,611,712
```

Figure 3.1.16: ResNet50 model summary

- Finalized testing accuracy: **98.92%**

- Model performance accuracy graph

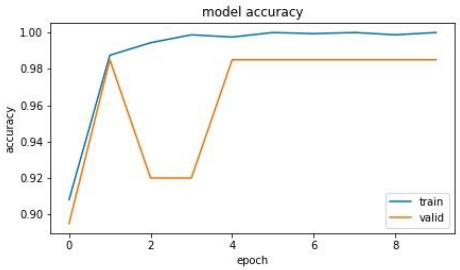

Figure 3.1.17: ResNet50 accuracy performance

- Model performance loss graph

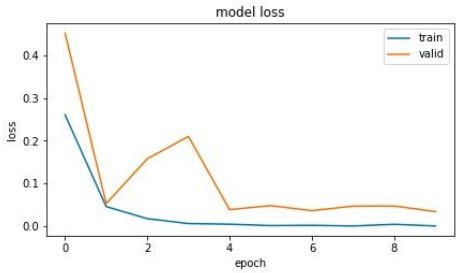

Figure 3.1.18: ResNet50 loss performance

VII. VGG16

- Model Summary

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 3, 3, 512)	14714688
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 256)	1179904
dense_1 (Dense)	(None, 6)	1542

Total params: 15,896,134
Trainable params: 15,896,134
Non-trainable params: 0

Figure 3.1.19: VGG16 model summary

- Finalized testing accuracy: **99.53%**

- Model performance accuracy graph

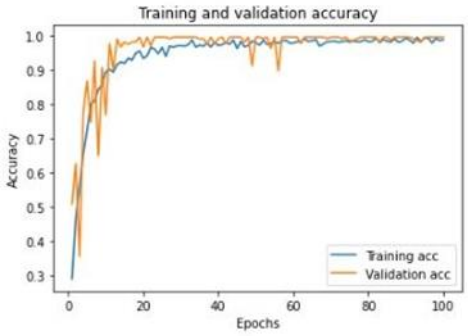

Figure 3.1.20: VGG16 accuracy performance

- Model performance loss graph

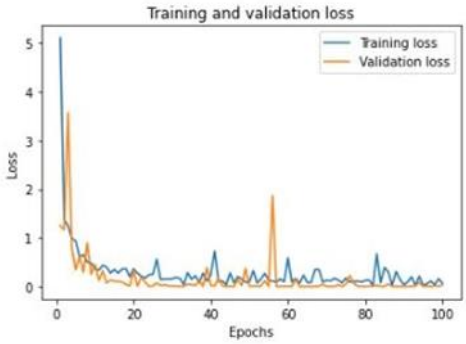

Figure 3.1.21: VGG16 loss performance

Classification results obtained from the finalized model

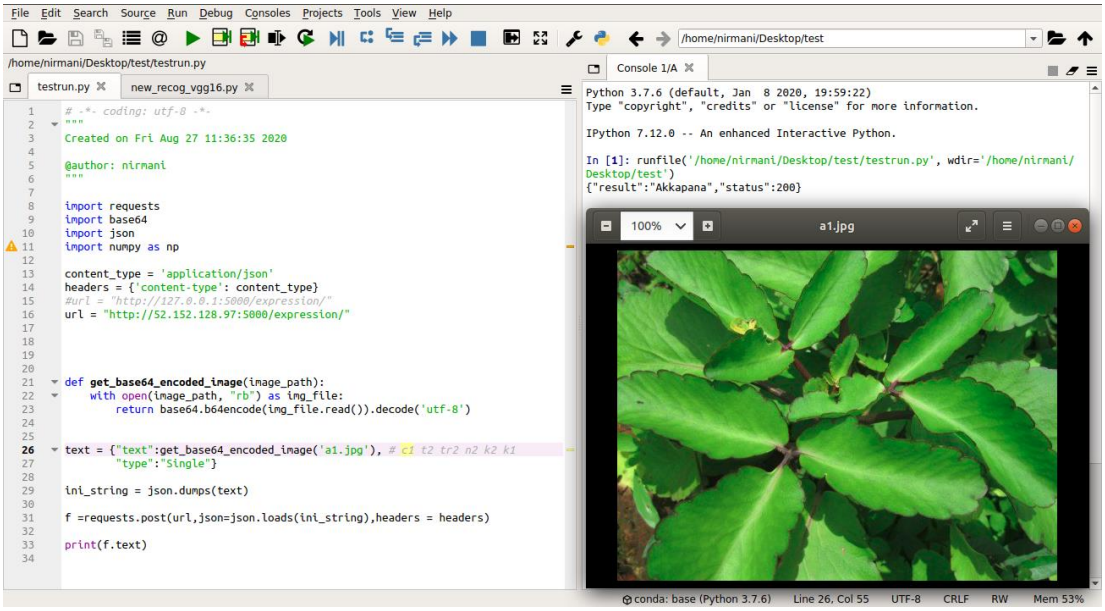

Figure 3.1.22: Classification of Akkapana

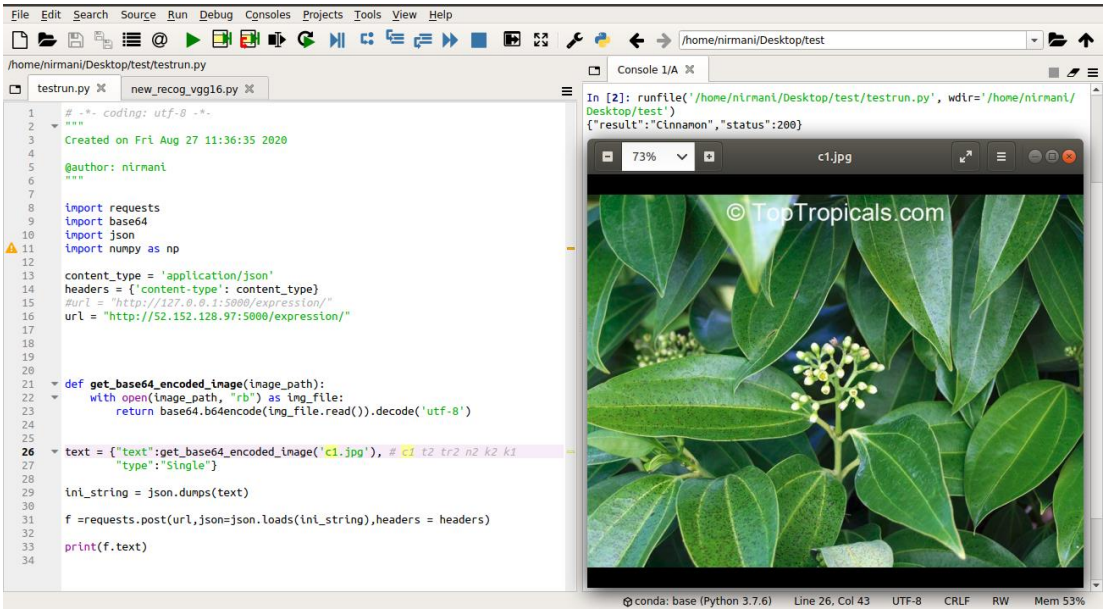

Figure 3.1.23: Classification of Cinnamon

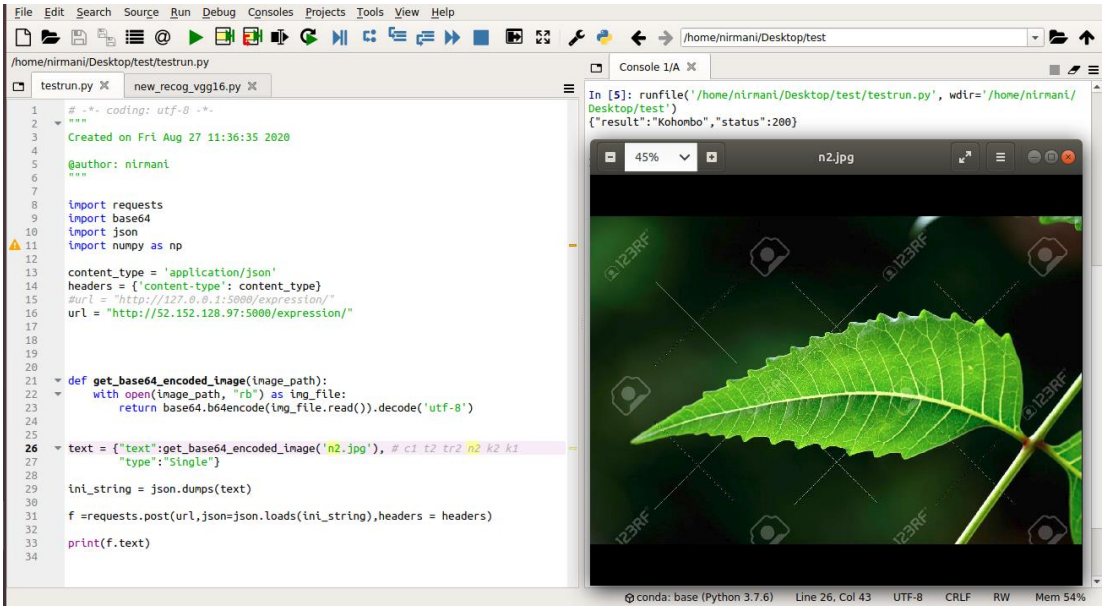

Figure 3.1.24: Classification of Kohomba

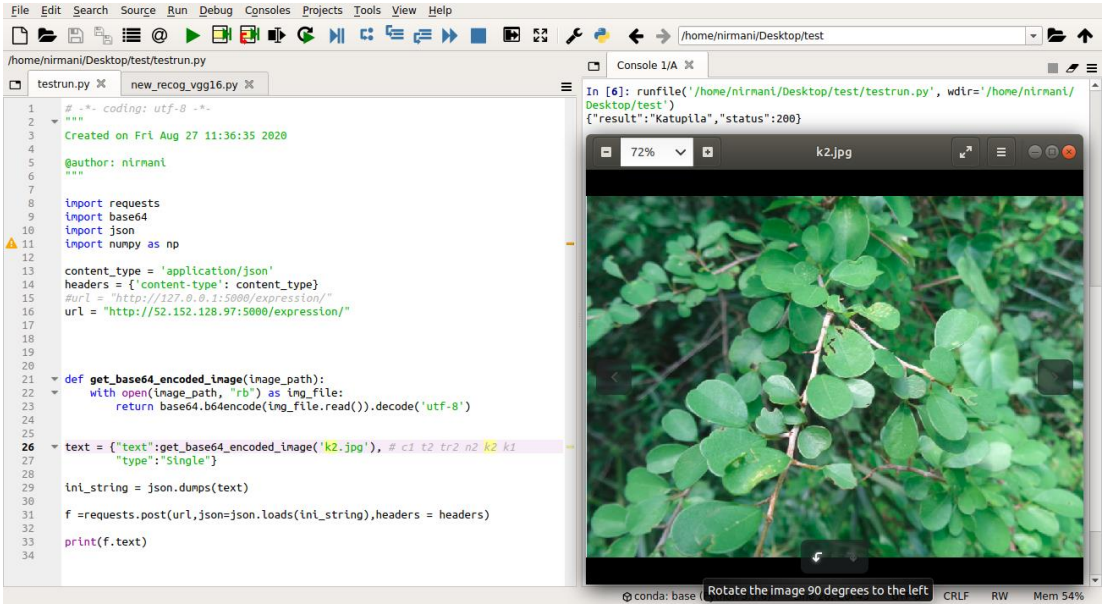

Figure 3.1.25: Classification of Katupila leaves

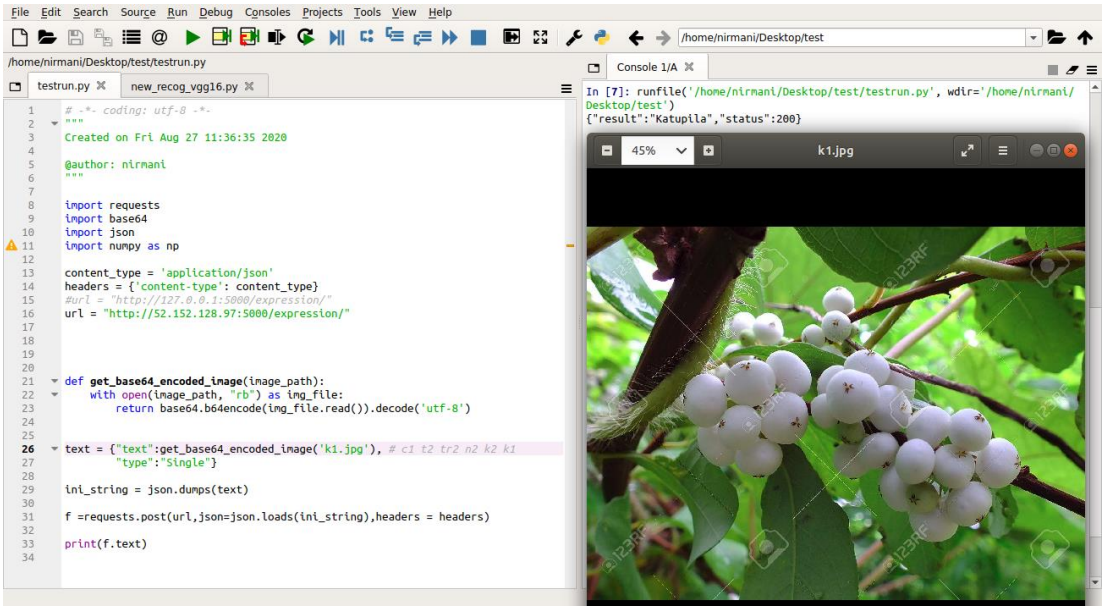

Figure 3.1.26: Classification of Katupila fruit

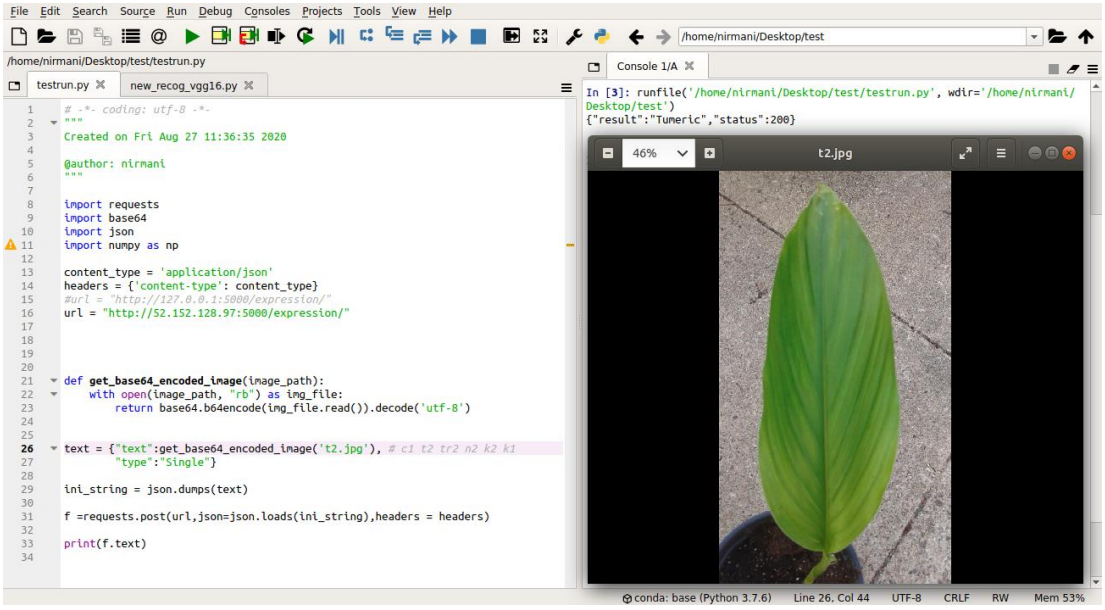

Figure 3.1.22: Classification of Turmeric leaf

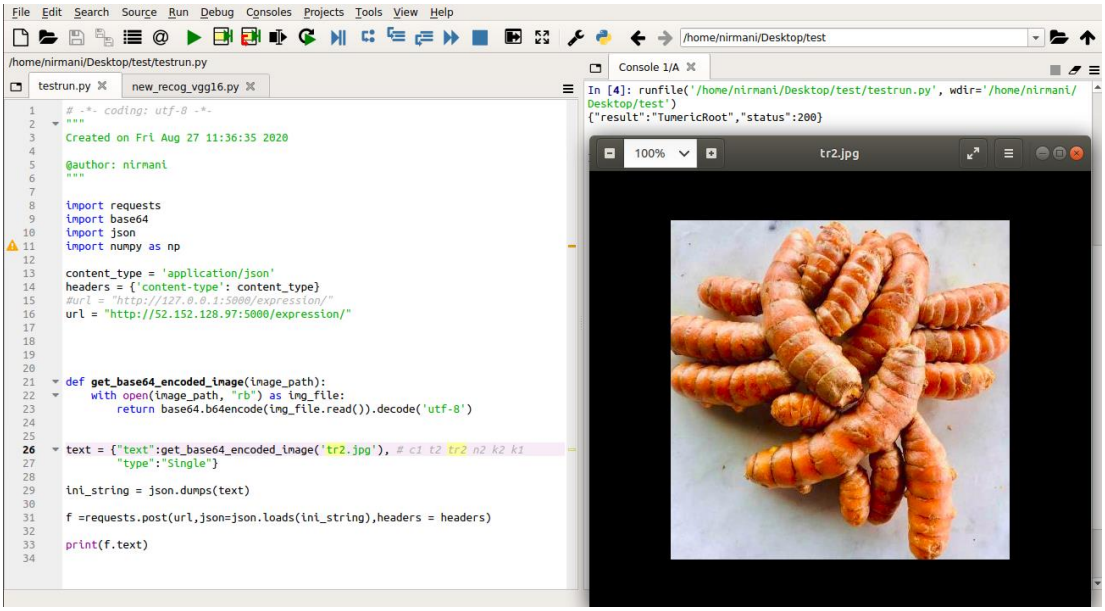

Figure 3.1.22: Classification of Turmeric root

3.2 Research Findings and Discussion

The main objective of this individual research component was to analyze several deep CNN models with the acquired training and testing data from scratch, get the final testing accuracy from each in order to compare and achieve the model with the highest accuracy, and then to use it as the finalized model in the herbal plant classification purpose in Arogya, based on images as the input from the mobile camera module.

Table 3.2.1: Comparison of accuracies for the selected CNN models in herbal plant classification

CNN Architecture	**Finalized testing accuracy**
InceptionV3	56.76%
MobileNetV2	63.58%
InceptionResNetV2	82..77%
Xception	86.01%
DenseNet121	89.12%
ResNet50	98.92%
VGG16	**99.53%**

Hence, according to the comparison, the pre-trained model VGG16 with a testing accuracy of 99.53% was chosen as the best model with the highest accuracy. The results were highly promising, reaching over 99% accuracy using the VGG16 model. So, this was the finalized model to be deployed in Arogya mobile application in order to classify the selected Ayurveda leaves/roots/fruits by the user.

The following table illustrates the experimental classification accuracies obtained for each plant class wise when the selected VGG16 model was applied for classification.

Table 3.2.2: Experimental classification accuracy class wise

Plant Class	Classified amount	Misclassified amount	Accuracy
Akkapana leaf	40	0	100%
Cinnamon leaf	36	4	90%
Katupila leaf with fruit	36	4	90%
Turmeric leaf	38	2	95%
Turmeric digested root	40	0	100%
Kohomba	40	0	100%
Average			**95.83%**

So, it is obvious with the experimental results that, the reliability of this research function can guarantee **95% in average.**

There is some research as well as mobile applications already available right now which classify plant leaves and give low prediction accuracies. However, these applications are based on foreign leaves, but not valuable medicinal plants or shrubs. So, Arogya mobile application would be a huge resource for almost everybody who keen to experience Ayurveda in their lives. Not only leaves, but also fruits, flowers and roots associated with Ayurveda will also be able to be classified with this application with a promising accuracy. This would be a great benefit for all the users since in Ayurveda not only leaves are used for preparation of medicine, but also the other parts of the plant are also used often for that purpose. In addition, the development strategy and methodology used in this approach will be able to be used and extended to identify any ayurvedic herb worldwide furthermore.

4. CONCLUSION

The purpose of this research is to develop a centralized social media application which is mobile-based and unique to herbal plants. This solution would be a great chance for those who are keen to learn and use Ayurveda medicine and plants, but who do not have prior knowledge about the specific domain. The application mainly creates awareness among common people about Ayurveda plants, their medicinal usage and value, and about their growth and availability throughout the country.

This proposed system has been tested in various situations and it is capable of providing the most reliable and accurate output to the user. According to the main research components focused, it has been experimentally proved in this research that the most accurate CNN pre-trained model used for classification purpose is VGG-16, which achieved a testing accuracy of 99.53%. The results are highly promising, reaching over 99% accuracy using the VGG-16 model.

This application is currently built only in English. Further, this can be applied in native languages such as Sinhala and Tamil. Since the system is designed only as a mobile application, later can be improvised to a web application with the same functionality and content. Additionally, the development strategy and methodology used in this approach will be able to be used and extended to identify any ayurvedic herb worldwide furthermore, and if the user ends up with doubts and clarifications regarding this procedure, this application can be facilitated with consultation help from Ayurveda doctors.

5. REFERENCES

[1] Pathirage Kamal Perera, "Current scenario of herbal medicine in Sri Lanka", ASSOCHAM, 4th annual Herbal International Summit cum Exhibition on Medicinal & Aromatic Products, Spices and finished products(hi-MAPS), pp. 1-3, April 2012.

[2] Lakpura LLC, Ayurveda. Accessed on: February 15,2020. [Online] Available: https://lanka.com/about/interests/ayurveda/

[3] Y. R. Azeez, R. A. C. P. Rajapakse, "An application of image processing techniques in identifying herbal plants in Sri Lanka", Proceedings of the 3rd. International Research Symposium on Pure and Applied Sciences, pp.1, 26th October 2018.

[4] Ayurveda Bansko, Blog, Benefits of Ayurveda and the difference to the traditional western medicine, 2015. Accessed on: February 15,2020. [Online] Available: https://www.ayurvedabansko.com/benefits-ayurveda-and-difference-to-western-traditional-medicine/.

[5] Basavaraj S. Anami, Suvarna S. Nandyal, A. Govardhan, "A Combined Color, Texture and Edge Features Based Approach for Identification and Classification of Indian Medicinal Plants", International Journal of Computer Applications (0975 – 8887), Volume 6– No.12, pp.1-3, September 2010.

[6] Robert G. de Luna, Renann G. Baldovino, Ezekiel A. Cotoco, Anton Louise P. de Ocampo, Ira C. Valenzuela, Alvin B. Culaba, Elmer P. Dadios, "Identification of Philippine Herbal Medicine Plant Leaf Using Artificial Neural Network", IEEE, 978-1-5386-0912-5/17/$31.00,PP.1-2, 2017.

[7] Rademaker C. A. 2000, "The classification of plants in the United States Patent Classification system", World Patent Information 22: 301–307.

[8] Neeraj Kumar, Peter N Belhumeur, Arijit Biswas, David W Jacobs, W John Kress, Ida C Lopez, João VB Soares, "Leafsnap: A computer vision system for automatic plant species identification," ECCV, pp. 502–516. Springer, 2012.

[9] Cem Kalyoncu, Önsen Toygar, "Geometric leaf classification," Computer Vision and Image Understanding", vol. 133, pp. 102–109, 2014.

[10] Abdul Kadir, Lukito Edi Nugroho, Adhi Susanto, Paulus Insap Santosa, "Leaf classification using shape, color, and texture features," arXiv preprint arXiv:1401.4447, 2013.

[11] Thibaut Beghin, James S Cope, Paolo Remagnino, SarahBarman, "Shape and texture based plant leaf classification," Advanced Concepts for Intelligent Vision Systems, 2010, pp. 345–353.

[12] James Charters, Zhiyong Wang, Zheru Chi, Ah Chung Tsoi, David Dagan Feng, “Eagle: A novel descriptor for identifying plant species using leaf lamina vascular features,” ICME-Workshop, 2014, pp. 1–6.

[13] Heysem KAYA, Ilhan KEKLIK, Tolga ENSARI, Fatih ALKAN, Yagmur BIRICIK, “Oak Leaf Classification: An Analysis of Features and Classifiers”, 978-1-7281-1013-4/19/$31.00, pp.1-4, 2019.

[14] James S Cope, Paolo Remagnino, Sarah Barman, Paul Wilkin, “The extraction of venation from leaf images by evolved vein classifiers and ant colony algorithms,” Advanced Concepts for Intelligent Vision Systems, pp. 135–144, 2010.

[15] Muammer TORKOGLU, Davut HANBAY, “Combination of Deep Features and KNN Algorithm for Classification of Leaf-Based Plant Species”.

[16] Jing Hu, Zhibo Chen, Meng Yang, Rongguo Zhang, Yaji Cui, “A Multiscale Fusion Convolutional Neural Network for Plant Leaf Recognition”, 1070-9908, IEEE SIGNAL PROCESSING LETTERS, VOL. 25, NO. 6, JUNE 2018.

[17] Agnieszka Mikołajczyk, Michał Grochowski, “Data augmentation for improving deep learning in image classification problem”, 17859931, ISBN: 978-1-5386-6143-7, May 2018.

[18] Will Koehrsen, Medium, towards data science, Transfer Learning with Convolutional Neural Networks in PyTorch, November 27, 2018. Accessed on: February 15,2020. [Online]
Available: https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce.

[19] Inside.Techlabs, Leaf Classification Project, October 23, 2018, Accessed on: February 15,2020. [Online]
Available: https://medium.com/@inside.techlabs/leaf-classification-project-89bb5c7577f3.

[20] Dileep M.R, Pournami P.N, “AyurLeaf: A Deep Learning Approach for Classification of Medicinal Plants”, IEEE, 978-1-7281-1895-6/19/$31.00, pp.1-4, 2019.

[21] Jun Woo Lee, Yeo Chan Yoon, "Fine-Grained Plant Identification using wide and deep learning model", Institute for Information & communications Technology Promotion (IITP), No. 2016-0-00010-003, Digital Content In-House R&D.

[22] Benjaphan Sommana, Thanaruk Theeramunkong, “Improving Plant Recognition using

Hybrid features from Connectionist and Knowledge-Based Approaches", IEEE, 978-1-7281-0161-3/18/$31.00, 2018.

[23] Hulya Yalcin, Salar Razavi, "Plant Classification using Convolutional Neural Networks", Visual Intelligence Laboratory Electronics & Telecommunication Engineering Department Istanbul Technical University, 2017.

[24] Jyotismita Chaki, Ranjan Parekh, Samar Bhattacharya, "Recognition of Whole and Deformed Plant Leaves using Statistical Shape Features and Neuro-Fuzzy Classifier", IEEE 2nd International Conference on Recent Trends in Information Systems (ReTIS), 978-1-4799-8349-0/15/$31.00, 2015.

[25] Chaoyun Zhang, Pan Zhou, Chenghua Li, Lijun Liu, "A Convolutional Neural Network for Leaves Recognition Using Data Augmentation", IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, DOI 10.1109/CIT/IUCC/DASC/PICOM.2015.318, 978-1-5090-0154-5/15 $31.00, 2015.

[26] Sue Han Lee, Paul Wilkin, Chee Seng Chan, Paolo Remagnino, "DEEP-PLANT: PLANT IDENTIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS", IEEE, 978-1-4799-8339-1/15/$31.00, 2015.

[27] ArunPriya C, Balasaravanan T, Antony Selvadoss Thanamani, "An Efficient Leaf Recognition Algorithm for Plant Classification Using Support Vector Machine", Proceedings of the International Conference on Pattern Recognition, Informatics and Medical Engineering, 978-1-4673-1039-0/12/$31.00, 2012, March 21-23, 2012.

[28] Shanwen Zhang, YouQian Feng, "Plant Leaf Classification Using Plant Leaves Based on Rough Set", 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), 978-1-4244-7237-6/10/$26.00, 2010.

[29] Stephen Gang Wu, Forrest Sheng Bao, Eric You Xu, Yu-Xuan Wang, Yi-Fan Chang, Qiao-Liang Xiang, "A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network", IEEE International Symposium on Signal Processing and Information Technology, 978-1 -4244-1 835-0/07/$25.00 , 2007.

[30] Pushpa BR, Anand C, Mithun Nambiar P, "Ayurvedic Plant Species Recognition using Statistical Parameters on Leaf Images", International Journal of Applied Engineering Research, ISSN 0973-4562, Volume 11, Number 7 (2016) pp 5142-5147.

[31] Sethulekshmi A V, Sreekumar K, "Ayurvedic leaf recognition for Plant Classification", Sethulekshmi A V et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (6), ISSN:0975-9646, 2014, 8061-8066.

[32] A Gopal, S. Prudhveeswar Reddy, V. Gayatri, “Classification of Selected Medicinal Plants Leaf Using Image Processing”, IEEE, 978-1-4673-2322-2112/$31.00, 2012.

[33] J. W. Tan, S. Chang, S. Binti Abdul Kareem, H. J. Yap, K. Yong, “Deep learning for plant species classification using leaf vein morphometric”, pages 1–1, 2018.

[34] R. Janani, A. Gopal, “Identification of selected medicinal plant leaves using image features and ann”, 2013 International Conference on Advanced Electronic Systems (ICAES), pages 238–242, Sept 2013.

[35] Gavai, N. R., Jakhade, Y. A., Tribhuvan, S. A., & Bhattad, R. (2017, December). MobileNets for flower classification using TensorFlow. In 2017 International Conference on Big Data, IoT and Data Science (BID) (pp. 154-158). IEEE.

[36] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 265-283).

[37] MathWorks, Deep Learning. Accessed on: February 19,2020. [Online] Available: https://www.mathworks.com/discovery/deep-learning.html.

[38] Machine Learning Mastery, How to Improve Performance With Transfer Learning for Deep Learning Neural Networks. Accessed on: February 19,2020. [Online]
Available: https://machinelearningmastery.com/how-to-improve-performance-with-transfer-learning-for-deep-learning-neural-networks/.

[39] TechTarget, Convolutional Neural Network. Accessed on: February 19, 2020. [Online]
Available: https://searchenterpriseai.techtarget.com/definition/convolutional-neural-network

[40] Hervé Goëau, Pierre Bonnet, Alexis Joly, Julien Barbe, Souheil Selmi, Vera Baki´c, Jennifer Carré, Daniel Barthelemy, Nozha Boujemaa, “Pl@ntNet Mobile App”, Proceedings of the 21st ACM international conference on Multimedia, DOI: 10.1145/2502081.2502251, ppt. 1-2, October 2013.

[41] Neeraj kumar Kumar, Peter N. Belhumeur, Arijit Biswas, João V. B. Soares, “Leafsnap: A Computer Vision System for Automatic Plant Species Identification”, European Conference on Computer Vision, DOI:10.1007/978-3-642-33709-3_36, ppt. 1, October 2012.

[42] Desta Sandya, Yeni Herdiyeni, “MedLeaf: Mobile Application For Medicinal Plant Identification Based on Leaf Image”, International Journal on Advanced Science, Engineering and Information Technology, Vol. 3 (2013) No. 2 ISSN: 2088-5334, ppt 1-3, May 2013.

[43] Lin-Hai Ma, Zhong-Qiu Zhao, Jing Wang, “ApLeafis: An Android-Based Plant Leaf Identification System”, International Conference on Intelligent Computing, LNCS 7995, pp. 106–111, 2013.

[44] PredictiveAnalyticsWorld.com, 3 Ways to Test the Accuracy of Your Predictive Models. Accessed on: February 22, 2020. [Online]
Available:http://https//www.predictiveanalyticsworld.com/machinelearningtimes/3-ways-test-accuracy-%20predictive-models/3295/

[45] Machine Learning Mastery, What is the Difference between Test and Validation Datasets?
Accessed on: February 22, 2020. [Online]
Available: https://machinelearningmastery.com/difference-test-validation-datasets/

[46] ResearchGate, What is training and testing data in machine learning?
Accessed on: February 22, 2020. [Online]
Available:https://www.researchgate.net/post/What_is_training_and_testing_data_in_machine_learning