A

MAJOR PROJECT REPORT

on

“Study and Analysis of Mobile based GeoAI framework for Plant Classification and Geotagging”

Submitted by

Nishikanta Parida
Regd. No.: 2124100015

Under the Guidance of

Mr. Dilip Kumar Dalei
Scientist 'F', CAIR, DRDO, Bengaluru, India

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATION
ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH
BHUBANESWAR, Techno Campus, PO - Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751003

2023

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751003)

CERTIFICATE

This is to certify that the work embodied in the project work entitled "**Study and Analysis of Mobile based GeoAI framework for Plant Classification and Geotagging**" being submitted by **Mr. NISHIKANTA PARIDA** in partial fulfilment for the award of the Degree of Master of Computer Science and Application to the Odisha University of Technology & Research is a record of bonafide work carried out by him under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Signature of Guide
Name: Mr. Dilip Kumar Dalei
Designation: Scientist 'F'

Signature of Head of the department
Name: Dr. Jibitesh Mishra
Designation: Associate Professor

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751003)

DECLARTION

I **Nishikanta Parida** bearing Regd. no.: **2124100015,** a bonafide student of Odisha University of Technology & Research, would like to declare that the Project work entitled "**Study and Analysis of Mobile based GeoAI framework for Plant Classification and Geotagging**", is a record of an original work done by me under the esteemed guidance of **Mr. Dilip Kumar Dalei**, Scientist 'F' at CAIR, DRDO, Bengaluru, India. This project work is submitted in the fulfilment of the requirements for Master's degree.

Nishikanta Parida

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751003)

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my advisor **Mr. Dilip Kumar Dalei**, Scientist 'F' at CAIR, DRDO, Bengaluru, India, whose knowledge and guidance has motivated me to achieve goals I never thought possible. He has consistently been a source of motivation, encouragement, and inspiration.

I would also like to convey my deep regards to all other faculty members of Department of Computer Science and Application, who have bestowed their great effort and guidance at appropriate times without which it would have been very difficult on my part to finish this work. Finally, I would also like to thank my friends for their suggestions and cooperation.

What I write or mention in this sheet will hardly be adequate in return for the amount of help and cooperation I have received from all the people who have contributed to make this project a reality. I am grateful to all for their constant support.

Date: **Nishikanta Parida**

Regd.No.: 2124100015

ABSTRACT

Plants are critical for sustaining life on earth by providing food and oxygen necessary for living things. Earth is abundant with different species of plants, which are being continuously studied and analyzed by biologists, researchers and nature lovers. From time immemorial, plants are source of medicines and food for human being. The medicinal values of plants are always been an area of research for discovery of new potential drugs. Moreover, the usage of leaf for food is always a promising idea for many critical civil and military situations where food availability is a matter of concern. Edible leaf is quite helpful in many situations like military operations for soldiers for longer survivability. The same is also applicable for civilian activities like mountain trekking, forest touring etc.

Plant Leaf is one key part that carries crucial information for characterizing and identification of a plant. Apart from leaf, other parts such as flower, fruits etc. are also being studied, but in a lesser extent. Leaf classification is essential for characterization plant and building a central database comprising a comprehensive plant related information. There are numerous efforts for collecting and building such plant database through various means. The modern approaches for plant identification comprises machine learning and deep learning-based methods to expediate plant data collection, identification and management. This necessitates to build and design a modular and usable system which can be easily used by a user for leaf-based plant identification. Mobile based Leaf classification system is a promising solution that capture leaf image from a mobile and processes these images using deep learning techniques in backend servers.

The current project looks into the area of building such a complete system where the user can capture an image of a leaf through a mobile device and wants to find out the characteristics of leaf on the device instantly. The paper made a thorough study and analysis of requirement of such a GeoAI framework starting from an android app to a server module. It makes a detailed investigation of establishing a real-time storage of leaf data and analysis of leaf data using deep learning methods. A prototype system has been built to understand the complete workflow of such GeoAI system. A detailed analysis of various CNN models for leaf data classification was also carried out in the project.

Contents

Figures

Tables

CHAPTER 1: INTRODUCTION

1.1 OVERVIEW

Plants are the essential resource for human well-being providing us with oxygen and food. So, the researchers along with breeding industry are making great efforts to continue agriculture for a long period without any interruptions. A good knowledge of plants is essential to help fully recognize new, different or uncommon plant species in order to support the ecosystem. The life of humans depends on plants as if the plants exist, humans also exist. The use of plants for humans may vary from the food to medicines. So, a robust method for plant identification and classification is very much needed.

Plant leaves are studied by many researchers and botanist to a greater detail. And also, by Environmental scientists to keep track of plant species in an area hence plant species with location data i.e. Geotagging is necessary.

Plant classification refers to identify and map the plant to a known plant spice. Different modern and advanced models have been proposed for an automatic plant identification and Geotagging is the process of appending geographic coordinates to media based on the location of a mobile device. But these techniques need to be accessible for a large dataset creation and plant species analysis.

1.2 PROBLEM STATEMENT

Plant Leaf is one key part that carries crucial information for characterizing and identification of a plant. There are already a lot of research regrading leaf classification using various techniques, but a mobile system with robust enough architecture to identify plant leaf with geotagging capabilities which can be easily accessible is needed for plant analysis.

This brings out the importance of an efficient and mobile based system to identify, classify and geotag of plant leaves.

1.3 OBJECTIVE

The objective of this work is to analyze various leaf classification system to find a suitable architecture for the “Mobile based GeoAI framework” system which is accurate, have geotagging capabilities and accessible. The study will also build a prototype GeoAI System for leaf classification using CNN models.

1.4 SCOPE OF WORK

- Study and Analysis of AI models for Plant Classification and Identification.
- Study and Analysis of framework and applications for mobile-based Geo AI operations.
- Design and development of an Android mobile app for Plant Leaf Image Management and Geotagging.

1.5 PROPOSED SOLUTION

- Our proposed solution is to implement a Deep Learning model to classify leaves.
- Another Deep Learning model to remove the background of the leaf for better accuracy.
- Use a mobile device to capture the image and location data of the leaf in order to do geo-tagging.
- A centralized database to store all results and captured data for building a geotagged dataset for future research.

CHAPTER 2: LITERATURE SURVEY

2.1 Neeraj Kumar et al. [18]

The authors designed and developed a complete mobile app, called "LeafSnap" for automatic plant species classification. This is considered as the first of its kind end-to-end visual recognition system for identifying plant species using computer vision techniques. The system identifies the plant species through leaf images captured on a mobile device. The system has following main steps:

- Classifying whether the image is of a valid leaf, to decide if it is worth processing further, using a binary classifier applied to gist features.
- Segmenting the image to obtain a binary image separating the leaf from the background by estimating foreground and background color distributions in the saturation-value space of the HSV color space.
- Extracting curvature features from the binarized image for compactly and discriminatively representing the shape of the leaf.
- Comparing the features to those from a labeled database of leaf images and returning the species with the closest matches using a simple nearest neighbor approach with histogram intersection as the distance metric.

The system is built for IOS platform only. The authors have also released the complete dataset in public for further research and analysis. One downside of this system that the leaf has to be on a clear white background, which is not an always feasible in real world scenario.

2.2 Sofiene Mouine et. al. [22]

The authors developed an android application for plant identification using selective leaf characters. The application provides option of the leaf character to guide the identification process. Two kinds of descriptors are proposed: a shape descriptor based on a multiscale triangular representation of the leaf margin and a descriptor of the salient points of the leaf. Th work has been evaluated on four public leaf datasets: Swedish, Flavia, ImageCLEF 2011 and 2012.

2.3 Sue Han Lee et. al [21]

The authors have carried a deep learning based system for automatic plant identification. The authors have employed pretrained CNN model based on AlexNet and fine-tuned for a leaf dataset. In this they employed deep learning in a bottom-up and top-down manner for plant identification. In the former, they choose to use a convolutional neural networks (CNN) model to learn the leaf features as a means to perform plant classification. In the latter, rather than using the CNN as a black box mechanism, they employed deconvolutional networks (DN) to visualize the learned features. This is in order to gain visual understanding on which features are important to identify a leaf from different classes, thus avoiding the necessity of designing hand-crafted features.

The CNN model used in this paper is based on the model proposed in with ILSVRC2012 dataset used for pre-training. Rather than training a new CNN architecture, they re-used the pre-trained network due to the fact that features extracted from the activation of a CNN trained in a fully supervised manner on large-scale object recognition works can be re-purposed to a

novel generic task.

They have also collected a new leaf dataset, named as Malaya Kew (MK) Leaf Dataset is also collected with full annotation.

2.4 Sue Han et al. [25]

The authors employed the well trained convolutional neural network model to perform plant identification. They proposed a methodology to recognize the learned features using deconvolution networks (DN), instead of using the CNN. This approach was used to obtain visual perception of the features needed to recognize a leaf from various classes, thereby evading the need to manually design the features. They worked on a new dataset called MalayaKew Leaf Dataset with only 44 classes. They created a new dataset (called as D2) by extending the given dataset(D1) by cropping manually and rotating the images. They selected 34672 leaf patches for training and 8800 for testing randomly, which resulted in 99.6% accuracy on the D2 dataset and 97.7% on the D1 dataset.

2.5 Jing Hu et al. [26]

For the purpose of leaf detection at various scales of plants, the authors proposed a MSF-CNN (MultiScale Fusion Convolutional Neural Network). They down-sampled an input image with a list of bilinear interpolation operations into multiple low-resolution images. The images were then fed into the MSF- CNN architecture to learn different characteristics in various layers, step by step. The final feature for anticipating the input image plant species is obtained by aggregating all last layer information. They re-trained the DeepPlant on D1 dataset and predicted classes with an accuracy of 98.1% using Support Vector Machine(SVM) model and 97.7% accuracy using Multi-Layer Perceptron approach(MLP). They observed that some of the classes were misclassified and concluded that identifying the shape of plants is not a good choice to recognize plants. Then they trained their model on the D2 dataset and achieved 99.6%, which is higher than the D1 dataset. They concluded that the D2 has better performance than D1, is because venation of distinct orders is a more robust characteristic for plant recognition.

2.6 Mohamed Abba et al. [27]

The authors have modified a trained model to visually identify the leaves in images. They showed how a model can be used on a small training dataset that is already trained on a large dataset. Its result was that the traditional machine learning methods were outperformed with the use of local binary patterns (LBPs). They didn't train their model from scratch and instead, took a CNN model trained on ImageNet. They worked on an ImageClef2013 dataset which includes images of clean as well as the cluttered background. Due to the shortage of the training data, it led to the problem of overfitting and high variability. They, therefore, applied transfer learning to avoid overfitting and made the AlexNet fine-tuning with the help of Caffe framework. They compared AlexNet from scratch with the help of random initialization with the fine-tuning versions which resulted in an accuracy of 71.17% on validation dataset and 70.0% on the testing dataset

2.7 Zhong-Qiu Zhao et. al. [30]

The authors have developed an Android based app called "AppLeaf" for automatic identification of plant species using tree leaves. They have assumed a controlled condition where the leaf image should be in light and untextured background without much clutter. Theier process involves three steps: leaf segmentation, feature extraction and species identification. The work is validated on ImageCLEF20212 Plant database having 126 tree species.

2.8 Ali Beikmohammadi et. al. [29]

The authors have devised a system comprising three integrated CNN based models for automated leaf classification. The system is built on the model of a botanist's behavioral approach for leaf identification. The paper has designed three CNN models - SLeafNET, W-LeafNET and P-LeafNET. The three models were utilized successively based on uncertainty of classification information in previous model. The complete work has been validated on two open datasets - MalayaKew (MK) and Flavia. The paper reported 99.81% and 99.67% of accuracy in the classification results.

2.9 Pierre Barré et. al. [20]

The authors have developed an automated plant identification system using leaf features. The system uses CNN based techniques for leaf classification. The work is tested on three public datasets - LeafSnap, Flavia and Foliage.

They trained the LeafNet network on the LeafSnap dataset in 200,000 iterations and employed a momentum $\alpha = 0.9$, mini-batches of 10 training examples and started with a learning rate $\eta = 0.001$.And after 100,000 iterations the learning rate is decreased by a gamma value $\gamma = 0.1$ resulting in a new value of the learning rate $\eta = 0.0001$ to foster convergence.

2.10 Vinit Bodhwani et. al. [28]

The authors have proposed a 50-layer deep learning model using residual networks which is designed to overcome the problem of vanishing gradient descent in case of large number of hidden layers.The model is mainly designed with the help of 5 stages. Initially, zero-padding pads the input image of dimension 64x64x3 with a pad of size (3,3). The resulted image is fed into a 7×7 convolution layer followed by batch normalization applied to the channel's axis of the input and a 3×3 max pooling layers. In stage 2, the convolutional block uses three sets of filters of size (64×64×256) and stacking 2 identity blocks together using three sets of filters of size (64×64×256). The dimension of images increases suddenly and in the last stage, the convolutional block uses three sets of filters of size (512×512×2048) followed by stacking 2 identity blocks which use three sets of filters of size (256×256×2048). The network finally ends with an average pooling and the output is flattened by adding a fully connected layer that reduces the number of classes with softmax activation.

This is evaluated on LeafSnap dataset which consists of 185 different tree species. The proposed model achieves an accuracy rate of 93.09 percent with 0.24 percent error.

CHAPTER 3: DATASET

3.1 OVERVIEW

The current work is carried out on five different datasets, in which four are public dataset and one is newly built by the us through manual collection and segregation. The publicly available datasets are MalayaKew Leaf (Two variants - MK_D1 and MK_D2), Swedish Leaf, Leaf Snap, Flavia. The new dataset collected and organized during this work is named as LeafMX.

	Mean	Standard deviation
Flavia	59	1.591
MK_D1	32	1.057
MK_D2	494	3.181
Swedish_leaf	75	2.236
Leaf_snap	58	3.107

Table 1 Dataset Mean & Standard deviation

3.2 MALAYAKEW LEAF DATASET

MalayaKew [Lee et al., 2017, Lee et al., 2015] is the leaf dataset collected at the Royal Botanic Gardens, Kew, England. It consists of scan-like images of leaves from 44 species classes. This dataset is very challenging as leaves from different species classes have very similar appearance.

There are two variant of this dataset namely MK-D1 and MK-D2

Mk-D1 dataset: It consists of segmented leaf images with size 256 * 256 pixels.
Number of training and testing images are 2288 and 528 respectively.
Dataset Groundtruth (MK-D2): It consists of cropped image patches of leaf with size 256 * 256 pixels.

Number of training and testing images are 34672 and 8800 respectively.

#	DATASET	No of Images	No of Classes	Image Resolution
1.	MalayaKew Leaf (MK_D1)	2816	44	256*256
2.	MalayaKew Leaf (MK_D2)	43472	44	256*256

Table 2 MALAYAKEW LEAF DATASET

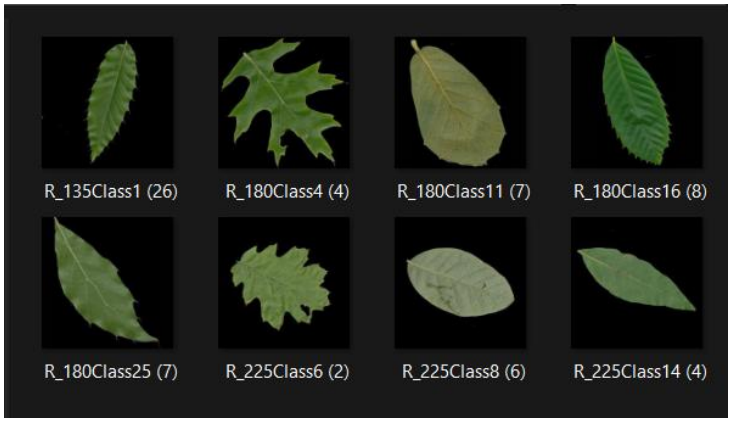


Figure 1 MK D1 Dataset

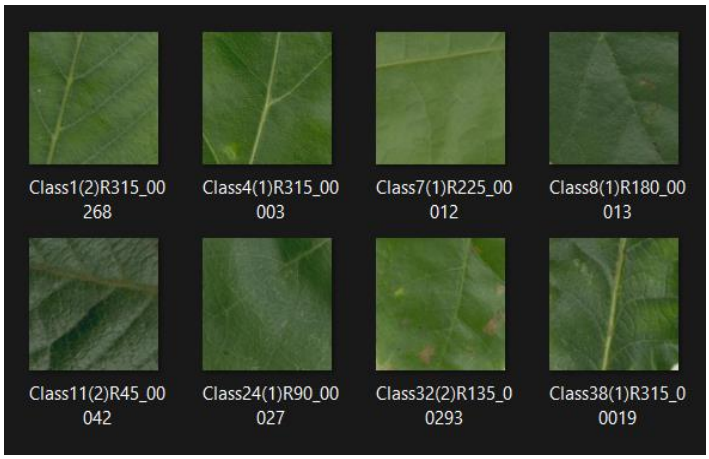


Figure 2 MK D2 Dataset

3.3 SWEDISH LEAF DATASET

Swedish Leaf dataset is another widely used public dataset prepared at Linkoping University and the Swedish Museum of Natural History under a leaf classification project [32]. This dataset consists of 15 species of leaves, with 75 images per species for a total of 1,125 images. The Swedish leaf dataset contain leaf having a lot of similarity, making it a challenging task for classification.

#	DATASET	No of Images	No of Classes	Image Resolution
1.	Swedish Leaf	1125	15	1457*2482

Table 3 SWEDISH LEAF DATASET

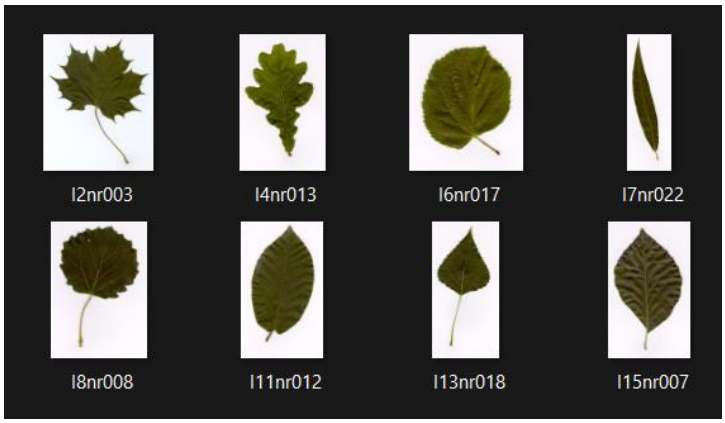


Figure 3 Swedish Leaf Dataset

Figure 4 Swedish Leaf Masked Dataset

3.4 LEAF SNAP DATASET

LeafSnap [http://leafsnap.com/dataset/] another large dataset consisting of 30866 images of leaves with coverage for all of the 185 tree species of the Northeastern United States. The dataset has two categories – field and lab. The field category contains 7719 images which are taken in outdoor field through mobile cameras. The lab-category has 23147 high-quality images prepared in a highly controlled setup of a Laboratory. Each image is labeled with tree species associated with the leaf.

#	DATASET	No of Images	No of Classes	Image Resolution
1.	Leaf Snap (Field)	7719	184	800*597
2.	Leaf Snap (Lab)	23147	185	783*800

Table 4 LEAF SNAP DATASET

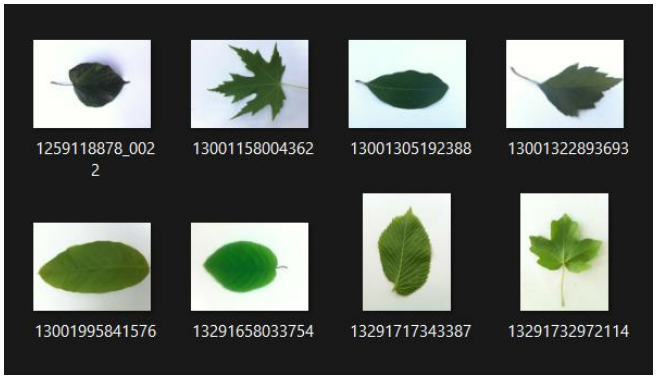


Figure 5 Leaf Snap Dataset

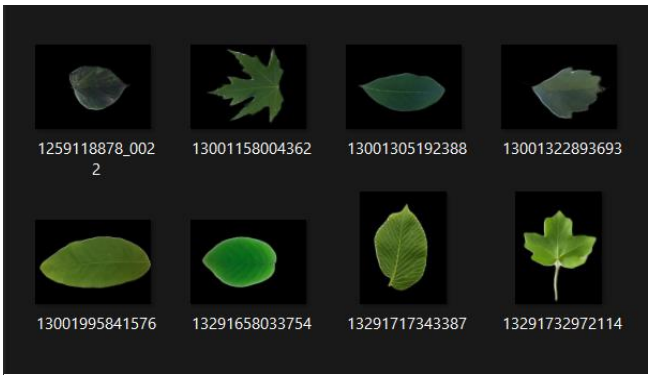


Figure 6 Leaf Snap Masked Dataset

3.5 FLAVIA [WU ET AL., 2007]

Flavia is a well-known dataset for leaf identification. It consists of 1907 leaf images of 32 different species. The images are sized 1600x1200 pixels. It contains images of only leaves without the petiole and any complex background. The leaves are common plants found in the region of Yangtze Delta, China. The dataset contains 50 images from each class.

#	DATASET	No of Images	No of Classes	Image Resolution
1.	Flavia	1907	32	1600*1200

Table 5 FLAVIA DATASET

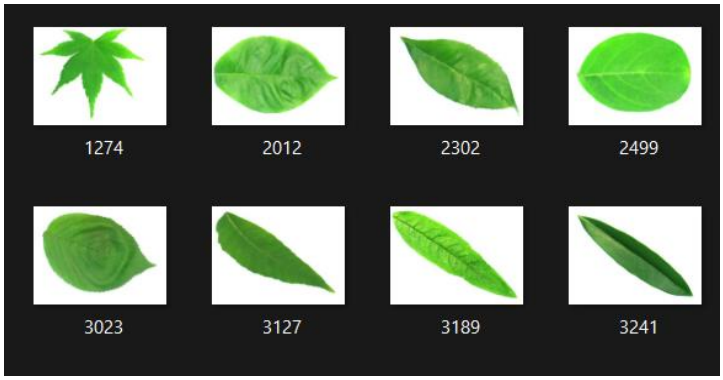


Figure 7 Flavia Dataset

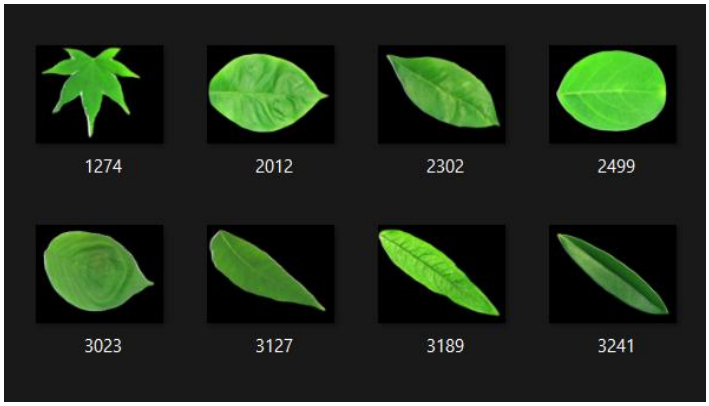


Figure 8 Flavia Masked Dataset

3.6 LEAFMX DATASET

LeafMX is the name of our new dataset that was collected and build during this work for training the background removal U2-net model.

The LeafMX dataset contain many pictures of complex scenes that are captures from different types of mobile camera with different lighting conditions to train the model on a realistic environment that the optimal environment of a lab.

#	DATASET	No of Images	No of Classes	Image Resolution
1.	LeafMX (Our Dataset)	450	1	960*1280

Table 6 LEAFMX DATASET

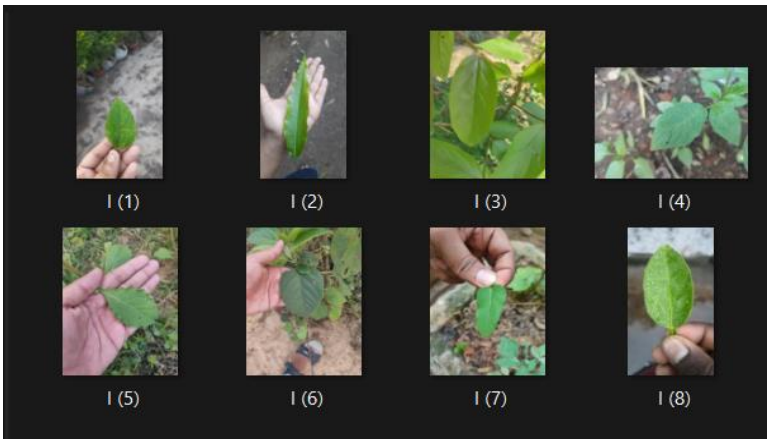


Figure 9 LeafMX Dataset

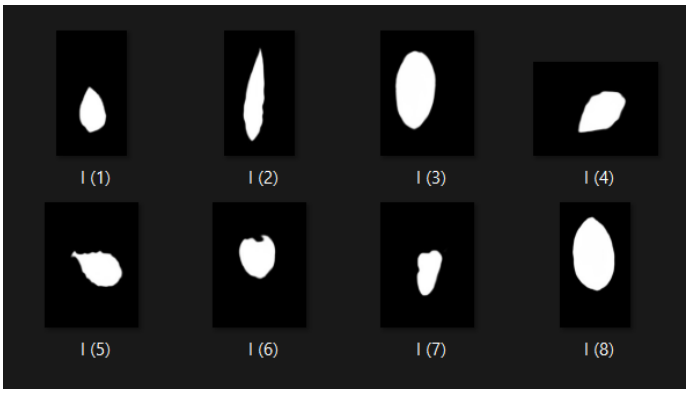


Figure 10 LeafMX Dataset Generated Masks

Figure 11 LeafMX Masked Results

Chapter 4: IMPLEMENTATION

4.1 PLATFORM

4.1.1 SOFTWARE REQUIREMENTS

- Anaconda interpreter (to run deployed project i.e., Python file)
- Jupyter Notebook (For Model building, training and testing)
- Visual Studio Code
- Python Libraries: cv2, Numpy, Pandas, Matplotlib, TensorFlow, TensorFlow, Object Detection API, OpenCV, PyTorch.

4.1.2 HARDWARE REQUIREMENTS

- Windows 10- 64 Bit
- AMD Ryzen 5 Gen 3
- Graphics card GeForce GTX 1650 4GB
- RAM 8 GB
- SSD: 51

4.2 TOOLS

4.2.1 Python

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming. Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

4.2.2 Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and MacOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012.

As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free. Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source package can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator as a graphical alternative to the command line interface (CLI).

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

4.2.3 Jupyter Notebook

The Jupyter Notebook is an opensource web application that you can use to create and share documents that contain live code, equations, visualizations, and text. In other words, Jupyter Notebook is an open-source, web-based IDE with deep cross language integration that allows you to create and share documents containing live code, equations, visualizations, and narrative text. Data scientists and engineers use Jupyter for data cleaning and transformation, statistical modeling, visualization, machine learning, deep learning, and much more. Jupyter Notebook's format (ipynb) has become an industry standard and can be rendered in multiple IDEs, GitHub, and other places. Jupyter has support for over 40 programming languages, including Python, R, Julia, and Scala. Notebooks can be shared easily with others, and your code can produce rich, interactive output, including HTML, images, videos, and custom MIME types. It allows you to leverage big data tools such as Spark and explore that same data with pandas, scikit-learn, TensorFlow.

4.2.4 Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. At the core of the NumPy package, is the ndarray.

object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

4.2.5 Pandas

Pandas is a fast, powerful, flexible and easy to use opensource data analysis and manipulation tool, built on top of the Python programming language. Pandas is a Python package providing fast, flexible and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis is python.

4.2.6 Matplotlib

Matplotlib is quite possibly the simplest way to plot data in Python. It is similar to plotting in MATLAB, allowing users full control over fonts, line styles, colors, and axes properties. This allows for complete customization and fine control over the aesthetics of each plot, albeit with a lot of additional lines of code. Plotly is another great Python visualization tool that's capable of handling geographical, scientific, statistical, and financial data. Plotly has several advantages over matplotlib. One of the main advantages is that only a few lines of codes are necessary to create aesthetically pleasing, interactive plots. The interactivity also offers a number of advantages over static matplotlib plots.

4.2.7 TensorFlow

Tensorflow is an open-source library for numerical computation and large-scale machine learning that ease Google Brain TensorFlow, the process of acquiring data, training models, serving predictions, and refining future results.

Tensorflow bundles together Machine Learning and Deep Learning models and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++.Tensorflow allows developers to create a graph of computations to perform. Each node in the graph represents a mathematical operation and each connection represents data. Hence, instead of dealing with low-details like figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application.

4.2.8 OpenCV

OpenCV (Open-Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel. OpenCV features GPU acceleration for real-time operations. is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception.

4.2.9 PyTorch

PyTorch is an open-source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Meta AI. A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers, PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).
- Deep neural networks built on a tape-based automatic differentiation system.

4.2.10Postman

Postman is a standalone software testing API (Application Programming Interface) platform to

build, test, design, modify, and document APIs. It is a simple Graphic User Interface for sending and viewing HTTP requests and responses.

In Postman, for testing purposes, one doesn't need to write any HTTP client network code. Instead, we build test suites called collections and let Postman interact with the API.

In this tool, nearly any functionality that any developer may need is embedded. This tool has the ability to make various types of HTTP requests like GET, POST, PUT, PATCH, and convert the API to code for languages like JavaScript and Python.

4.3 METHODOLOGY

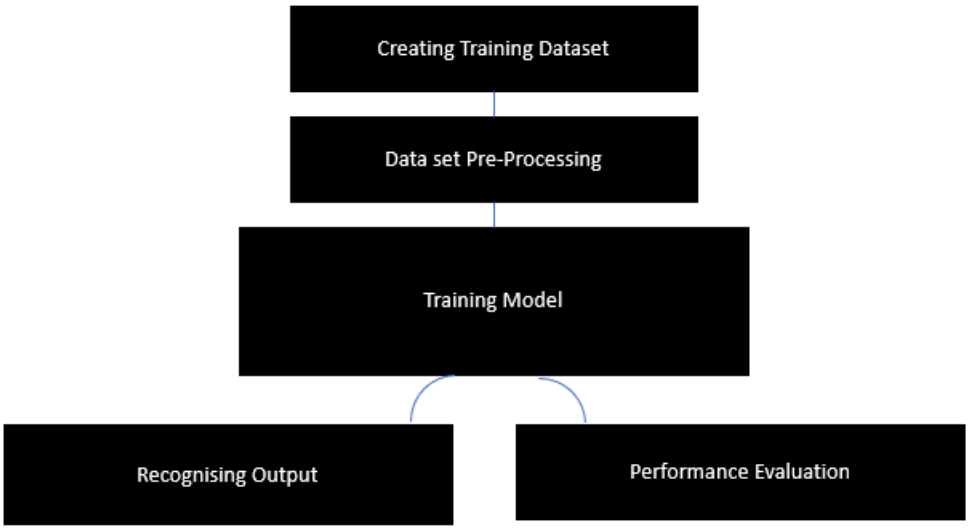


Figure 12 Methodology

The Methodology of this project consists of following steps, first step is creating dataset it can be done by creating our own dataset from scratch or collection existing dataset from the web. Second step is data pre-processing in which we clean and remove the corrupted data and transform or augment the data if necessary. In the next step we train a model, after that we evaluate the performance metrics and recognise the output.

4.4 CNN (Convolutional Neural Network) Models

4.4.1 OVERVIEW

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what colour each pixel should be.

The human brain processes a huge amount of information the second we see an image. Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a

CNN processes data only in its receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along. By using a CNN, one can enable sight to computers.

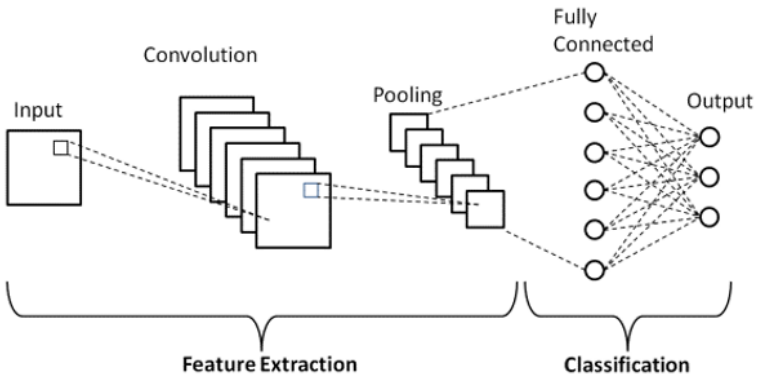


Figure 13: CNN model

I. Convolution Layer

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

II. Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

III. Fully Connected Layer

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The Fully Connected layer helps to map the representation between the input and the output.

IV. Non-Linearity Layers

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

There are several types of non-linear operations, the popular ones being:

a. Sigmoid

The sigmoid non-linearity has the mathematical form $\sigma(\kappa) = 1/(1+e^{-}\kappa)$. It takes a real-valued number and "squashes" it into a range between 0 and 1.

b. Tanh

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the

activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

c. ReLU

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function $f(\kappa)=\max(0, \kappa)$. In other words, the activation is simply threshold at zero.

4.4.2 Inception-V3

Inception V3 model was published in the paper "Rethinking the Inception Architecture for Computer Vision", by Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Inception V3 Architecture was published in the same paper as Inception V2 in 2015, and we can consider it as an improvement over the previous Inception Architectures.

It is a convolutional neural network for assisting in image analysis and object detection, and got its start as a module for GoogLeNet. It is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge. The design of Inceptionv3 was intended to allow deeper networks while also keeping the number of parameters from growing too large: it has "under 25 million parameters", compared against 60 million for AlexNet.

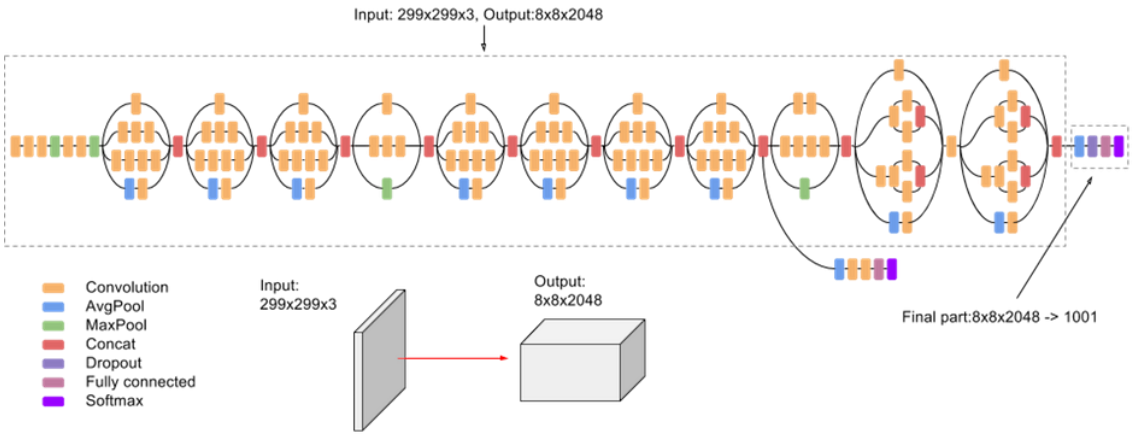


Figure 14 Inception-V3

4.4.3 VGG16

VGGNet is a Convolutional Neural Network architecture proposed by Karen Simonyan and Andrew Zisserman from the University of Oxford in 2014. This paper mainly focuses on the effect of the convolutional neural network depth on its accuracy. The original paper of VGGNet which is titled Very Deep Convolutional Networks for Large Scale Image Recognition.
The input to VGG based convNet is a 224*224 RGB image. Preprocessing layer takes the RGB image with pixel values in the range of 0–255 and subtracts the mean image values which is calculated over the entire ImageNet training set.

The input images after preprocessing are passed through these weight layers. The training images are passed through a stack of convolution layers. There are total of 13 convolutional layers and 3 fully connected layers in VGG16 architecture. VGG has smaller filters (3*3) with more depth instead of having large filters. It has ended up having the same effective receptive field as if you only have one 7 x 7 convolutional layers.

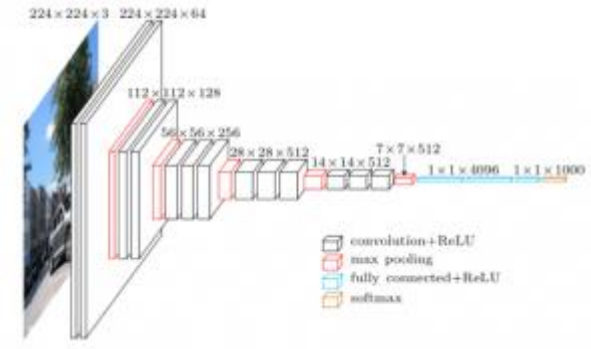


Figure 15 VGG16

4.4.4 ResNet-50

ResNet-50 is a convolutional neural network that is 50 layers deep. ResNet, short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers. It is an innovative neural network that was first introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 computer vision research paper titled 'Deep Residual Learning for Image Recognition'.

Convolutional Neural Networks have a major disadvantage — 'Vanishing Gradient Problem'. During backpropagation, the value of gradient decreases significantly, thus hardly any change comes to weights. To overcome this, ResNet is used. It make use of "SKIP CONNECTION".

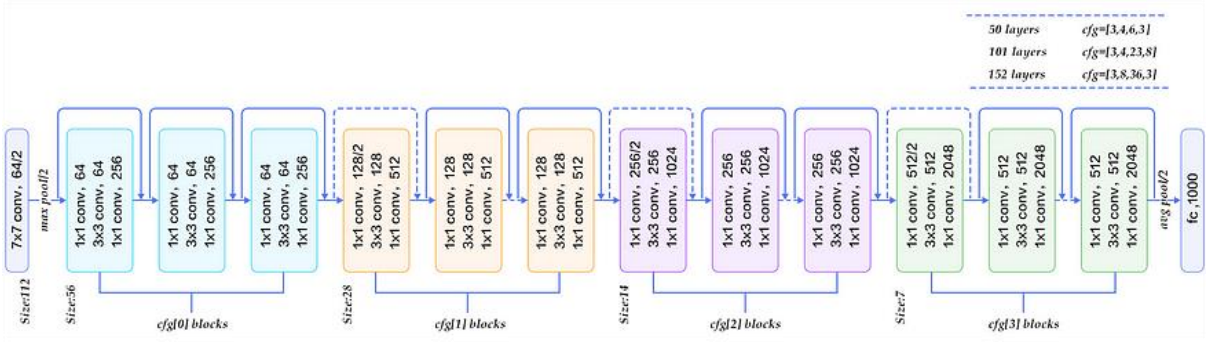


Figure 16 ResNet-50

4.4.5 MobileNetV2

MobileNetV2 builds upon the ideas from MobileNetV1 in the paper MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, by Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M and Adam H. Using depth wise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture: 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks.

The intuition is that the bottlenecks encode the model's intermediate inputs and outputs while the inner layer encapsulates the model's ability to transform from lower-level concepts such as pixels to higher level descriptors such as image categories. Finally, as with traditional residual connections, shortcuts enable faster training and better accuracy.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 17 MobileNetV2

4.4.6 EfficientNet-B0

EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient. Unlike conventional practice that arbitrary scales these factors, the EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. For example, if we want to use 2^N times more computational resources, then we can simply increase the network depth by α^N, width by β^N, and image size by Υ^N, where α,β,Υ are constant coefficients determined by a small grid search on the original small model. EfficientNet uses a compound coefficient to uniformly scales network width, depth, and resolution in a principled way.

The compound scaling method is justified by the intuition that if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image.

The base EfficientNet-B0 network is based on the inverted bottleneck residual blocks of MobileNetV2, in addition to squeeze-and-excitation blocks.

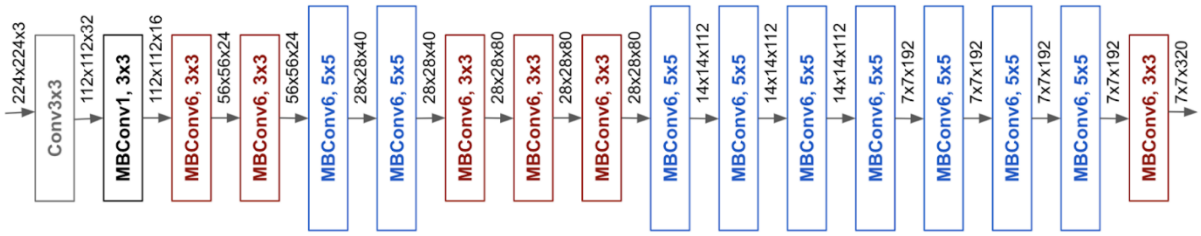


Figure 18 EfficientNet-B0

4.5 U^2 -Net

Salient Object Detection (SOD) aims at segmenting the most visually attractive objects in an image. It is widely used in many fields, such as visual tracking and image segmentation. Recently, with the development of deep convolutional neural networks (CNNs), especially the rise of Fully Convolutional Networks (FCN) in image segmentation, the salient object detection has been improved significantly.

There is a common pattern in the design of most SOD networks, these all have backbones that are all originally designed for image classification, such as Alexnet , VGG, ResNet , ResNeXt, DenseNet , etc. They extract features that are representative of semantic meaning rather than local details and global contrast information, which are essential to saliency detection. As these backbones are trained on ImageNet data which is data-inefficient especially if the target data follows a different distribution than ImageNet.

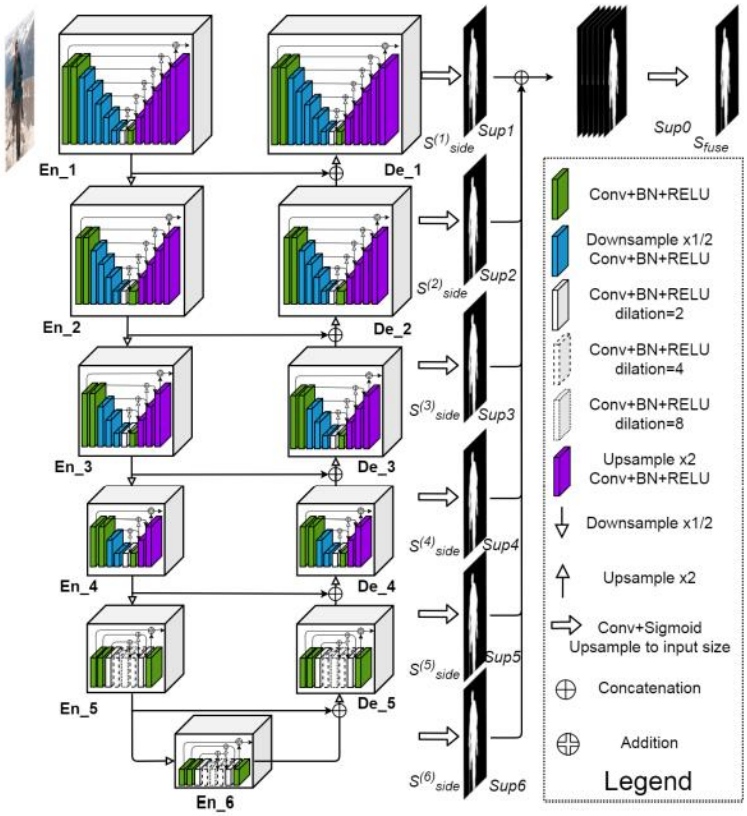


Figure 19 u2 - Net Architecture

U2 -Net is a two-level nested U-structure that is designed for SOD without using any pre-trained backbones from image classification. It can be trained from scratch to achieve competitive performance. Second, the novel architecture allows the network to go deeper, attain high resolution, without significantly increasing the memory and computation cost. This is achieved by a nested U-structure: on the bottom level, with a novel ReSidual U-block (RSU), which is able to extract intra-stage multi-scale features without degrading the feature map resolution; on the top level, there is a U-Net like structure, in which each stage is filled by a RSU block as shown in the Figure-1.

Based on RSU, the researchers developed U²-Net, their novel stacked U-shaped structure for salient object detection. U²-Net consists of a 6-stage encoder, a 5-stage decoder, and a saliency graph fusion module attached to the decoder stages and the last encoder stage.

Overall, the U²-Net design builds a deep architecture with rich multiscale features and low computational and memory costs. In addition, since the U²-Net architecture is built on RSU blocks alone and does not use any pretrained backbone network for image classification processing, it can be flexibly and easily adapted to different working environments with minimal performance loss.

ReSidual U-block (RSU)

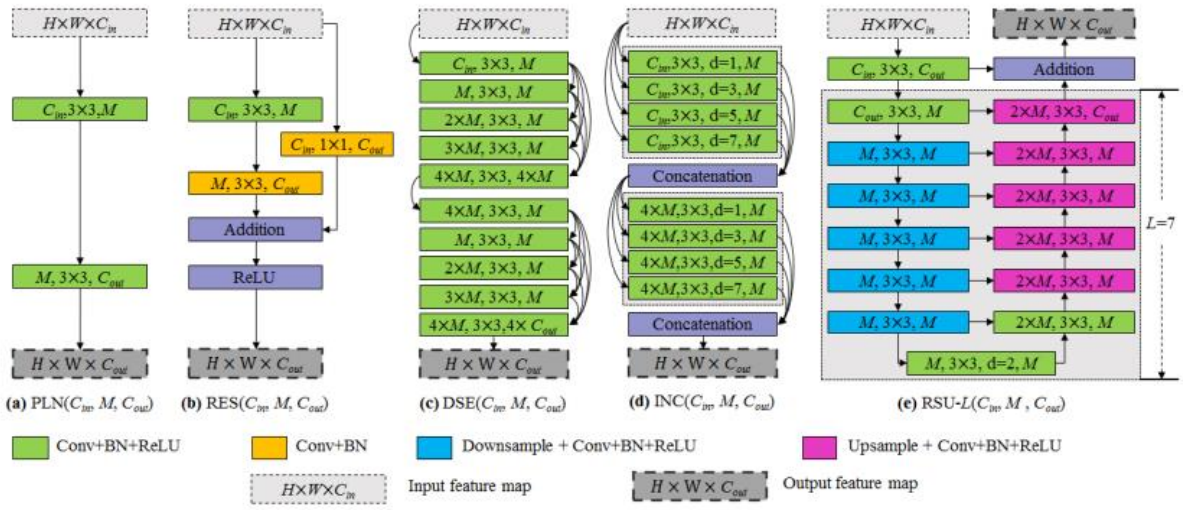


Figure 20 Illustration of existing convolution blocks and residual U-block RSU: (a) Plain convolution block PLN, (b) Residual-like block RES, (c) Dense-like block DSE, (d) Inception-like block INC and (e) Our residual U-block RSU.

Both local and global contextual information are very important for salient object detection and other segmentation tasks. In modern CNN designs, such as VGG, ResNet, DenseNet and so on, small convolutional filters with size of 1×1 or 3×3 are the most frequently used components for feature extraction. They are in favour since they require less storage space and are computationally efficient. Figures 2(a)-(c) illustrates typical existing convolution blocks with small receptive fields. The output feature maps of shallow layers only contain local features because the receptive field of 1×1 or 3×3 filters are too small to capture global information. To achieve more global information at high resolution feature maps from shallow layers, the most direct idea is to enlarge the receptive field. Fig. 2 (d) shows an inception like block [50], which tries to extract both local and non-local features by enlarging the receptive fields using dilated convolutions [3]. However, conducting multiple dilated convolutions on the input feature map (especially in the early stage) with original resolution requires too much computation and

memory resources. To decrease the computation costs, PoolNet [22] adapt the parallel configuration from pyramid pooling modules (PPM) [57], which uses small kernel filters on the down sampled feature maps other than the dilated convolutions on the original size feature maps. But fusion of different scale features by direct up sampling and concatenation (or addition) may lead to degradation of high-resolution features.

The RSU has three main components: an input convolutional layer, a U-Net-like symmetric encoder-decoder structure of L height, and a residual connection to fuse local and multiscale features through summation.

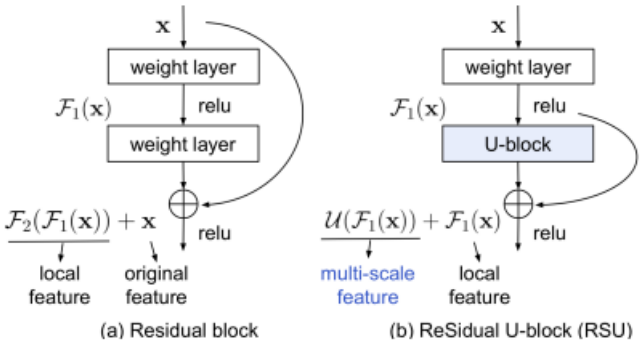


Figure 21 Comparison of the residual block and RSU

The main differences between the RSU and the original residual block is that the RSU replaces the ordinary single-flow convolution with a U-Net-like structure, and replaces original features with a local feature transformed via a weighting layer.

4.6 FLUTTER MOBILE FRAMEWORK

The app in this work has been developed in Flutter. Flutter is a cross-platform UI toolkit that is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services.

During development, Flutter apps run in a VM that offers stateful hot reload of changes without needing a full recompile. For release, Flutter apps are compiled directly to machine code, whether Intel x64 or ARM instructions, or to JavaScript if targeting the web.

Architectural layers:
Flutter is designed as an extensible, layered system. It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable.

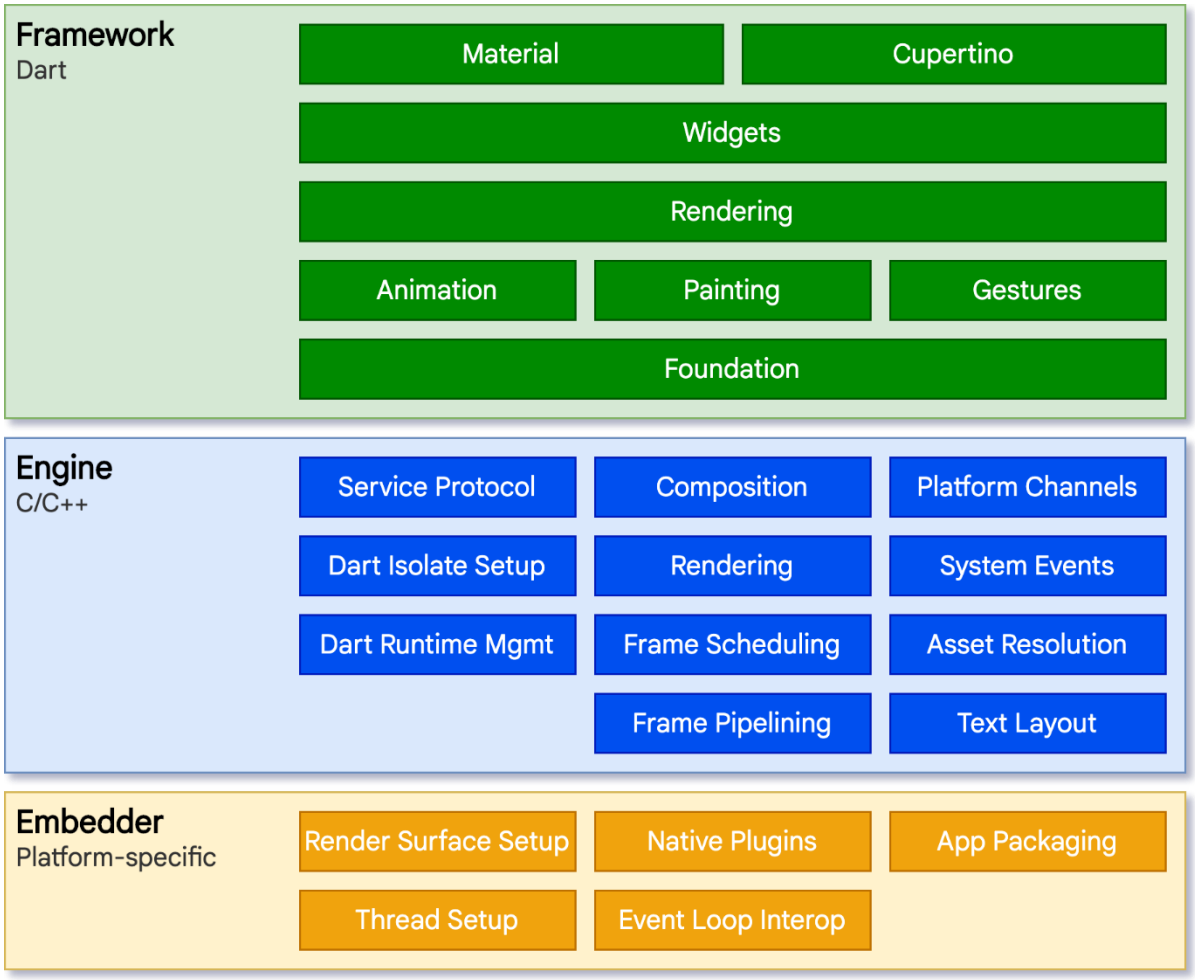


Figure 22 Flutter Architecture

Dart is an object-oriented, class-based, garbage-collected language with C-style syntax. Dart can compile to either native code or JavaScript. It supports interfaces, mixins, abstract classes, reified generics, and type inference.

The GeoAI AppServer databases are implemented using CloudFirestore. Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Cloud Firestore is a cloud-hosted, **NoSQL database** that iOS, Android, and web apps can access directly via native SDKs.

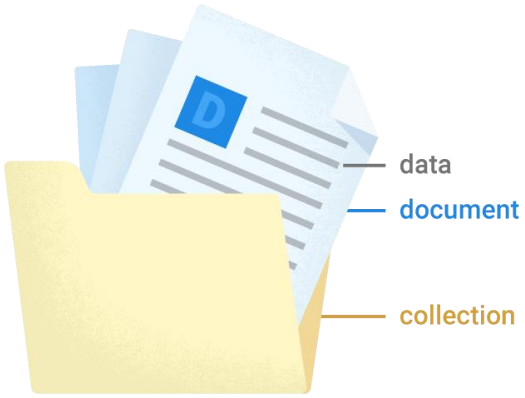


Figure 23 Firestore File Storing Structure

4.7 SYSTEM

A system is built to study the concepts related to leaf geotagging and classification using CNN methods. The system works on a client-server model. The client module is designed for an android based platform. The server module will carry out the storage and processing of leaf dataset. In fact, there are two servers at the backend - AppServer and GeoAI Server.The mobile client will take the image through device camera and initiate the connection to the AppServer. Based on server availability, the image data will be transferred to the AppServer for further processing. The database of the complete leaf images is stored in the AppServer. The database will have leaf images with the location information. The classification of the images are carried out in the GeoAI Server. The App Server will automatically initiate the connection GeoAI server for classification and retrieve the results. The complete process will be transparent to the client. The client will be updated with the results for display.

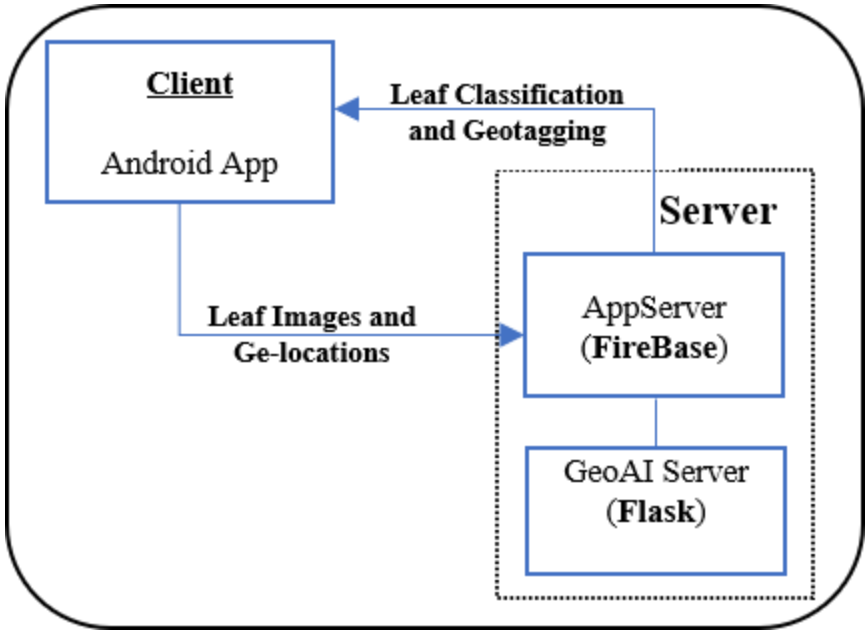


Figure 24 System Architecture

4.7.1 APPSERVER

The app server is designed using Firebase. Firebase is a set of backend cloud computing services and application development platforms provided by Google. It hosts databases, services, authentication, and integration for a variety of applications, including Android, iOS, JavaScript, Node.js, Java, Unity, PHP, and C++.

The client app first connects with the firebase server and then send the data for storage in the server. The data is stored in firebase-Firestore which is basically a NOSQL database. The firebase Storage is used to store images received from both client app and GeoAI server.
There are three core steps in this process. First, the data is stored in Firestore and firebase storage. After that, a new document object is created in Firestore. Second, a Firebase Cloud

function which is pre-configured with the firebase gets triggered and the image data is sent to the GeoAI server. Third, the received image data from GeoAI server is stored in the storage and the document object gets updated. With the completion of this step the client app is notified about the completion of the processing. The client now pulls and display the results.

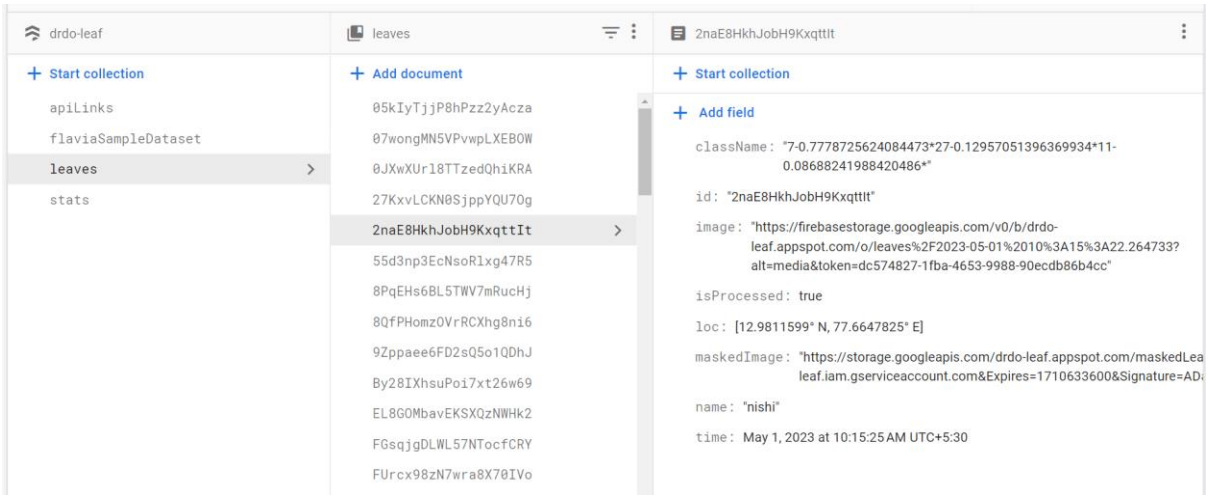


Figure 25 Firestore Database

The above figure is a screenshot of the firebase database, here the "leaf_doc" has fields like className, Id, image, isProcessed, loc, maskedImage, name and time. The id is to identify each doc uniquely, the image and masked image fields are links of the uploaded image and masked image respectively, loc field stores the location of the uploaded image and the time field stores the time the image was taken from the mobile. IsProcessed is Boolean filed specifying is the image is processed or not.

4.7.2 GEOAI SERVER

The geo AI server is implemented using Flask. Flask is a web framework, it's a Python module that lets you develop web applications easily. It's has a small and easy-to-extend core: it's a microframework that doesn't include an ORM (Object Relational Manager) or such features.It does have many cool features like url routing, template engine. It is a WSGI(Web Server Gateway Interface) web app framework.

It was developed by Armin Ronacher, who led a team of international Python enthusiasts called Poocco. Flask is based on the Werkzeg WSGI toolkit and the Jinja2 template engine which are both Pocco projects.

In this server we have two Deep Learning model are running i.e. one for image masking and another for image classification task. We trained our classification model (i.e., resnet50 with highest accuracy) and U2net trained with our LeafMX dataset. we load the model in the Flask app and the incoming requests from Firebase are first processed by the u2net and then the masked output image is given to the classification model for class prediction. After this the class-Name with the confidence is sent back to the Firebase as a response for further processing.

We like to specify one more detail that as these models are quite large, and we used are local

machine to run this server. But as we need access of this flask app through the internet, we used Ngrok secure tunnel that allowed us, to openly access our local system in the internet.

4.7.3 MOBILE CLIENT FRAMEWORK

The client app is developed using Flutter framework which is an opensource framework by Google with a support for generating cross-platform applications from a single code base.
The frame work is chosen to increase the wide platform accessibility of our system. So, the same app will work both on Android and IOS platform and also from a website through browser. The client app has utilized many libraries such as firebase connection, image picker, GPS location and integrated google maps API. The Image Picker helps to get image from camera as well as from the gallery. The Location tool is used for getting the current GPS location of the mobile device. The firebase and Google maps packages help for connection and map display activities respectively.

The App first captures the data from the user either through camera or from the local storage. The data is then sent to the firebase and waits for further processing. It waits for the notification from the firebase server about the completion of processing. After that, it shows the classification results with location information to the user. A Map is used in the background to position the leaf info at the exact location.

4.7.4 LEAF CLASSIFICATION USING CNN MODELS

Plant identification is a well-established area of study among biologists and plant specialists. There are a lot of literature available to understand and identify plants based on some characteristics. The building of plant information database has been a constant effort since time immemorial. In fact, there are historical accounts of efforts for building a knowledge database about plants and their information.

Leaf classification is the one of core activity in Plant identification process. The classification task works based on various characteristics of leaf - color, texture, shape etc. The shape feature is quite deterministic in classification process as the shape of a leaf stops changing after the growth of plant. The leaf features need to be extracted and analyzed for further classification. The leaf feature extraction is a critical step in classification task. For faster and accurate results, the classification tasks are being solved either by Machine Learning (ML) or Deep Learning (DL) techniques. Machine learning approaches need features to be extracted manually and feed to ML classifiers. This is needs expertise and time-consuming step for leaf classification. In Comparison to ML, DL methods are successfully applied to automate the feature extraction and classification task, thereby simplifying the automatic leaf classification system. Especially, CNN based DL models are showing promising results to classify leaf data.

In this work we used five CNN based DL models namely InceptionV3, VGG16, ResNet50, MobileNetv2, EfficeintNetB0 and U2net for background removal.

4.8 CODE TEMPLATES

4.8.1 Training u2-Net

u2net_train.py

```
import os
import torch
import torchvision
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F

from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
import torch.optim as optim
import torchvision.transforms as standard_transforms

import numpy as np
import glob
import os

from data_loader import Rescale
from data_loader import RescaleT
from data_loader import RandomCrop
from data_loader import ToTensor
from data_loader import ToTensorLab
from data_loader import SalObjDataset

from model import U2NET
from model import U2NETP

# ------- 1. define loss function --------
if __name__ == '__main__':

    bce_loss = nn.BCELoss(size_average=True)

    def muti_bce_loss_fusion(d0, d1, d2, d3, d4, d5, d6, labels_v):

        loss0 = bce_loss(d0,labels_v)
        loss1 = bce_loss(d1,labels_v)
        loss2 = bce_loss(d2,labels_v)
        loss3 = bce_loss(d3,labels_v)
        loss4 = bce_loss(d4,labels_v)
        loss5 = bce_loss(d5,labels_v)
        loss6 = bce_loss(d6,labels_v)
        loss = loss0 + loss1 + loss2 + loss3 + loss4 + loss5 + loss6
        print("l0: %3f, l1: %3f, l2: %3f, l3: %3f, l4: %3f, l5: %3f, l6:
%3f\n"%(loss0.data.item(),loss1.data.item(),loss2.data.item(),loss3.data.item(),loss4.data.ite
m(),loss5.data.item(),loss6.data.item()))
```

```
    return loss0, loss

# ------- 2. set the directory of training dataset --------

    model_name = 'u2net' #'u2netp'
    data_dir = os.path.join(os.getcwd(), 'train_data' + os.sep)
    tra_image_dir = os.path.join('im_aug' + os.sep)
    tra_label_dir = os.path.join('gt_aug' + os.sep)

    image_ext = '.jpg'
    label_ext = '.jpg'

    model_dir = os.path.join(os.getcwd(), 'saved_models', model_name + os.sep)

    epoch_num = 100000
    batch_size_train = 4
    batch_size_val = 1
    train_num = 0
    val_num = 0

    tra_img_name_list = glob.glob(data_dir + tra_image_dir + '*' + image_ext)

    tra_lbl_name_list = []
    for img_path in tra_img_name_list:
        img_name = img_path.split(os.sep)[-1]
        aaa = img_name.split(".")
        bbb = aaa[0:-1]
        imidx = bbb[0]
        for i in range(1,len(bbb)):
            imidx = imidx + "." + bbb[i]
        tra_lbl_name_list.append(data_dir + tra_label_dir + imidx + label_ext)

    print("---")
    print("train images: ", len(tra_img_name_list))
    print("train labels: ", len(tra_lbl_name_list))
    print("---")

    train_num = len(tra_img_name_list)
    salobj_dataset = SalObjDataset(
     img_name_list=tra_img_name_list,
    lbl_name_list=tra_lbl_name_list,
    transform=transforms.Compose([
      RescaleT(320),
      RandomCrop(288),
      ToTensorLab(flag=0)]))
    salobj_dataloader = DataLoader(salobj_dataset, batch_size=batch_size_train,
shuffle=True, num_workers=1)

# ------- 3. define model --------
# define the net
```

```
    if(model_name=='u2net'):
        net = U2NET(3, 1)
    elif(model_name=='u2netp'):
        net = U2NETP(3,1)

    if torch.cuda.is_available():
        net.cuda()

# ------- 4. define optimizer --------
    print("---define optimizer...")
    optimizer = optim.Adam(net.parameters(), lr=0.001, betas=(0.9, 0.999), eps=1e-08,
weight_decay=0)

# ------- 5. training process --------
    print("---start training...")
    ite_num = 0
    running_loss = 0.0
    running_tar_loss = 0.0
    ite_num4val = 0
    save_frq = 200 # save the model every 2000 iterations

    print('in main')
    if torch.cuda.is_available():
            print("GPU is here (:")
    else:
        print(";(")

    for epoch in range(0, epoch_num):
        net.train()

        for i, data in enumerate(salobj_dataloader):
            ite_num = ite_num + 1
            ite_num4val = ite_num4val + 1

            inputs, labels = data['image'], data['label']

            inputs = inputs.type(torch.FloatTensor)
            labels = labels.type(torch.FloatTensor)

      # wrap them in Variable
            if torch.cuda.is_available():
               #   print("GPU is here")
                inputs_v, labels_v = Variable(inputs.cuda(), requires_grad=False),
Variable(labels.cuda(),
                                                        requires_grad=False)
            else:
               #   print("GPU isnot here")
                 inputs_v, labels_v = Variable(inputs, requires_grad=False), Variable(labels,
requires_grad=False)
```

```
        # y zero the parameter gradients
            optimizer.zero_grad()

        # forward + backward + optimize
            d0, d1, d2, d3, d4, d5, d6 = net(inputs_v)
            loss2, loss = muti_bce_loss_fusion(d0, d1, d2, d3, d4, d5, d6, labels_v)

            loss.backward()
            optimizer.step()

        # # print statistics
            running_loss += loss.data.item()
            running_tar_loss += loss2.data.item()

        # del temporary outputs and loss
            del d0, d1, d2, d3, d4, d5, d6, loss2, loss

            print("[epoch: %3d/%3d, batch: %5d/%5d, ite: %d] train loss: %3f, tar: %3f " % (
            epoch + 1, epoch_num, (i + 1) * batch_size_train, train_num, ite_num, running_loss
/ ite_num4val, running_tar_loss / ite_num4val))

            if ite_num % save_frq == 0:
                torch.save(net.state_dict(),                    model_dir                    +
model_name+"_bce_itr_%d_train_%3f_tar_%3f.pth"    %    (ite_num,    running_loss    /
ite_num4val, running_tar_loss / ite_num4val))
                running_loss = 0.0
                running_tar_loss = 0.0
                net.train()  # resume train
                ite_num4val = 0
```

4.8.2 Training Classification Models

Training Dataset containing .png/.jpg files

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np
import tifffile as tiff
import os
import cv2
from skimage.transform import resize
from sklearn.preprocessing import LabelEncoder
```

```
import os
from PIL import Image
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2

batch_size = 64
img_size=128
epochs=100
NUM_CLASSES=32

dir='/kaggle/input/flavia-masked/flavia_masked'


train_ds, val_ds = tf.keras.utils.image_dataset_from_directory(
    dir,
    validation_split=0.2,
    label_mode='categorical',
    subset="both",
    seed=1337,
    image_size=(img_size,img_size),
    batch_size=batch_size,
)




def get_label_array(ds):
    y = []
    for batch in ds:
        batch_images, batch_labels = batch
        y.append(batch_labels.numpy())
    y = np.concatenate(y)
    y=np.argmax(y,axis=1)
    return y


y_test=get_label_array(val_ds)


import matplotlib.pyplot as plt
```

```
def plot_hist(hist):
    fig, axs = plt.subplots(1, 2, figsize=(12, 6)) # create a figure with 1 row and 2 columns

    # plot accuracy for training and validation sets
    axs[0].plot(hist.history["accuracy"])
    axs[0].plot(hist.history["val_accuracy"])
    axs[0].set_title("model accuracy")
    axs[0].set_ylabel("accuracy")
    axs[0].set_xlabel("epoch")
    axs[0].legend(["train", "validation"], loc="upper left")

    # plot loss for training and validation sets
    axs[1].plot(hist.history["loss"])
    axs[1].plot(hist.history["val_loss"])
    axs[1].set_title("model loss")
    axs[1].set_ylabel("loss")
    axs[1].set_xlabel("epoch")
    axs[1].legend(["train", "validation"], loc="upper left")

    plt.show()

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


def print_stats(y_test,y_pred) :
    confusion = confusion_matrix(y_test, y_pred)
    print('Confusion Matrix\n')
    print(confusion)
    print('\nAccuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))

    print('Micro Precision: {:.2f}'.format(precision_score(y_test, y_pred, average='micro')))
    print('Micro Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='micro')))
    print('Micro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred, average='micro')))

    print('Macro Precision: {:.2f}'.format(precision_score(y_test, y_pred, average='macro')))
    print('Macro Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='macro')))
    print('Macro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred, average='macro')))

    print('Weighted Precision: {:.2f}'.format(precision_score(y_test, y_pred,
average='weighted')))
    print('Weighted Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='weighted')))
    print('Weighted F1-score: {:.2f}'.format(f1_score(y_test, y_pred, average='weighted')))


def create_model_train_print_hist(model,modelname) :
    inputs = layers.Input(shape=(img_size,img_size, 3))
    # x = img_augmentation(inputs)
```

```
    x = layers.Rescaling(scale=1./255)(inputs)
    # x = layers.experimental.preprocessing.Resizing(299, 299)(x)
    output =model(x)
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()(output)
    prediction_layer = tf.keras.layers.Dense(NUM_CLASSES,
activation='softmax',kernel_regularizer=regularizers.l2(0.001))(global_average_layer)
    model = tf.keras.Model(inputs,prediction_layer)
    model.compile( optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
    model.summary()
    hist=model.fit(train_ds, epochs=epochs, validation_data=val_ds, verbose=2)
    model.save(modelname+'flavia.h5')
    plot_hist(hist)
    y_test_pred=model.predict(val_ds)
    y_test_pred=np.argmax(y_test_pred,axis=1)
    print_stats(y_test,y_test_pred)



model=InceptionV3(include_top=False, weights=None, classes=NUM_CLASSES)
mod=create_model_train_print_hist(model,'inception_v3')

model=VGG16(include_top=False, weights=None, classes=NUM_CLASSES)
create_model_train_print_hist(model,'vgg16')

model=ResNet50(include_top=False, weights=None, classes=NUM_CLASSES)
create_model_train_print_hist(model,'resnet50')

model=MobileNetV2(include_top=False, weights=None, classes=NUM_CLASSES)
create_model_train_print_hist(model,'mobilenetv2')

model=EfficientNetB0(include_top=False, weights=None, classes=NUM_CLASSES)
create_model_train_print_hist(model,'mobilenetv2')
```

4.8.3 Building Flask Server

```
from flask import Flask,jsonify
from flask import Flask, make_response
from flask import request
from PIL import Image
from keras.models import  load_model
import numpy as np
import io
from skimage import  transform
import cv2
import requests
import base64
from datetime import date
from datetime import datetime
```

```
import torch
from torch.autograd import Variable
from torchvision import transforms
from numpy import asarray
import numpy as np
from PIL import Image
from model import U2NET # full size version 173.6 MB
from model import U2NETP # small version u2net 4.7 MB

app=Flask(__name__)



label2Class=[---label array---]


def normPRED(d):
    ma = torch.max(d)
    mi = torch.min(d)

    dn = (d-mi)/(ma-mi)

    return dn


model_path="resnet50.h5"
#classification model path
img_size=128

model = load_model(model_path)
print('hi classification model loaded')


net = U2NET(3,1)
model_dir='u2net.pth'

if torch.cuda.is_available():
    net.load_state_dict(torch.load(model_dir))
    net.cuda()
else:
 net.load_state_dict(torch.load(model_dir, map_location='cpu'))
net.eval()
print('hi u2net model loaded')
transform = transforms.Compose([
    transforms.Resize((320, 320)),    # Rescale the image to 320x320
    transforms.ToTensor(),           # Convert the image to a PyTorch tensor
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),   # Normalize the pixel
values to the range [-1, 1]
```

```
])


@app.route('/')
def hello_world():
    now = datetime.now()
    return 'hello,world'+' '+str(now)




@app.route("/imsnew", methods=["POST"])
def process_image2():
    imglink = request.form["link"]
    f = open('tempimg.jpg','wb')
    f.write(requests.get(imglink).content)
    f.close()
    # file = request.files['image']
    # Read the image via file.stream
    img = Image.open('tempimg.jpg')



    imz=img.size
    uimg = transform(img)
    umig = uimg.type(torch.FloatTensor)
    uimg = uimg.unsqueeze(0)

    if torch.cuda.is_available():
        inputs_test = Variable(uimg.cuda())
    else:
       inputs_test = Variable(uimg)

    d1,d2,d3,d4,d5,d6,d7= net(inputs_test)

    pred = d1[:,0,:,:]
    pred = normPRED(pred)
    pred = pred.squeeze()
    predict_np = pred.cpu().data.numpy()
    im = Image.fromarray(predict_np*255).convert('RGB')
    imo = im.resize((imz[0],imz[1]),resample=Image.BILINEAR)

    img_org=asarray(img)
    img_mask=asarray(imo)
    res= cv2.bitwise_and(img_org,img_mask, mask=None)

    cimg = img.resize((img_size,img_size))
    cimg = np.array(cimg,dtype="uint8")
    cimg=np.array([cimg])
```

```
    output = model(cimg)
    arr=output[0]
    sorted=np.argsort(-arr)
    # className=int(np.argmax(output[0]))
    classIds=sorted[:3]

    # confidence=output[0][className].numpy()
    resString=""
    for i in classIds:
      resString+=str(i)+"-"+str(float(arr[i]))+"*"

    res=Image.fromarray(res)
    imgByteArr = io.BytesIO()
    res.save(imgByteArr, format='PNG')
    imgByteArr.seek(0)
    bimg=imgByteArr.getvalue()


    # resString=label2Class[className]+'*'+"confidence :"+str(confidence)
    # resString=str(className)+'*'+"confidence :"+str(confidence)


    response = make_response(bimg)
    response.headers.set('Content-Type', 'image/png')
    response.headers.set('Content-Disposition',str(resString))

    del d1,d2,d3,d4,d5,d6,d7

    return response


if __name__=="__main__":
    app.run(debug=True)
```

Building Flutter App

main.dart

```
import 'package:drdo_public/firebase_options.dart';
import 'package:drdo_public/home.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
```

```
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
          primarySwatch: Colors.blue,
          textTheme: TextTheme(
              headlineMedium: TextStyle(
            color: Colors.greenAccent,
            fontSize: 20.0,
            fontWeight: FontWeight.bold,
          )),
          appBarTheme: AppBarTheme(
            color: Colors.black,
            iconTheme: IconThemeData(color: Colors.green),
            titleTextStyle: TextStyle(
              color: Colors.green,
              fontSize: 20.0,
              fontWeight: FontWeight.bold,
            ),
          )),
      // home: DatasetSampleUpload(),
      home: Home(),
    );
  }
}
```

home.dart

```
import 'dart:async';
import 'dart:io';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:drdo_public/enterName.dart';
import 'package:drdo_public/globalValue.dart';
import 'package:drdo_public/leafDetails.dart';
import 'package:drdo_public/model/leaf.dart';
import 'package:drdo_public/model/leaf_map_points.dart';
import 'package:flutter/foundation.dart' show kIsWeb;
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
```

```
import 'package:image_picker/image_picker.dart';
import 'package:loading_indicator/loading_indicator.dart';
import 'package:location/location.dart';
import 'package:shared_preferences/shared_preferences.dart';

import 'services/cloud_firestore.dart';
import 'services/sorage.dart';

class Home extends StatefulWidget {
  const Home({Key? key}) : super(key: key);

  @override
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  bool flag = false;
  bool isSelected = false;
  final ImagePicker imagePicker = ImagePicker();
  XFile? selectedImage;
  LatLng loc2 = const LatLng(0.0, 0.0);
  var loc;

  String selectedImageString =
      "";

  static const LatLng _kMapCenter =
      LatLng(19.018255973653343, 72.84793849278007);

  final CameraPosition _kInitialPosition = const CameraPosition(
      target: _kMapCenter, zoom: 11.0, tilt: 0, bearing: 0);
  List<Marker> _markers = <Marker>[];
  List<Marker> updatedMarkers = [];
  // List<Markers> markers; //This t

  List<LeafMapPoints> lflist = [];

  Future _initLocationService() async {
    var location = Location();

    try {
      var serviceEnabled = await location.serviceEnabled();
    } on PlatformException catch (err) {
      // _serviceEnabled = false;

      // location service is still not created

      _initLocationService(); // re-invoke himself every time the error is catch, so until the
location service setup is complete
    }
```

```
    if (!await location.serviceEnabled()) {
      if (!await location.requestService()) {
        return;
      }
    }

    var permission = await location.hasPermission();
    if (permission == PermissionStatus.denied) {
      permission = await location.requestPermission();
      if (permission != PermissionStatus.granted) {
        return;
      }
    }

    loc = await location.getLocation();
    flag = true;
    setState(() {});
    loc2 = LatLng(loc.latitude, loc.longitude);

    final prefs = await SharedPreferences.getInstance();

    final String? nameVal = prefs.getString('name');

    if (nameVal == null) {
      name = "";
    } else {
      name = nameVal;
    }
  }

  @override
  void initState() {
    super.initState();
    _initLocationService();
  }

  void getMapPins() async {
    FireStoreService fireStoreService = FireStoreService();

    lflist = await fireStoreService.getLeafMapPoints();
    // final Uint8List markerIcon = await getBytesFromAsset("assets/images/waste-bin.png");
    // final Uint8List markerIcon2 = await getBytesFromAsset("assets/images/garbage-
truck.png");
    updatedMarkers = [];
    lflist.forEach((lf) async {
      ImageConfiguration configuration = const ImageConfiguration();
      updatedMarkers.add(Marker(
          markerId: MarkerId(lf.id),
          icon: await BitmapDescriptor.fromAssetImage(
```

```
            const ImageConfiguration(size: Size(12, 12)),
            "assets/icons8-oak-tree-48.png"),
          infoWindow: InfoWindow(
            title: "Tree data",
            snippet:
              "Class Name : ${lf.className} ,loc: [ ${lf.location.latitude}
${lf.location.longitude} ]"),
          position: lf.location));
    });

    _markers = updatedMarkers;
    setState(() {});
  }

  @override
  Widget build(BuildContext context) {
    FireStoreService fireStoreService = FireStoreService();
    StorageService storageService = StorageService();

    double height = MediaQuery.of(context).size.height;
    double width = MediaQuery.of(context).size.width;

    // print();

    var leafDocStream = FirebaseFirestore.instance
        .collection('leaves')
        .doc(currentDocId)
        .snapshots();

    var dlServerStatusStream = FirebaseFirestore.instance
        .collection('apiLinks')
        .doc('mlLocalHost')
        .snapshots();

    return DefaultTabController(
      length: 2,
      child: Scaffold(
          appBar: AppBar(
            title: const Text(
              'Leaf Geo AI',
              style: TextStyle(color: Colors.green),
            ),
            backgroundColor: Colors.black,
            bottom: const TabBar(
              tabs: [
                Tab(
                  child: Text(
                    'Identify leaf',
                    style: TextStyle(color: Colors.greenAccent),
                  ),
```

```
            ),
            Tab(
              child: Text(
                'Tree Map',
                style: TextStyle(color: Colors.greenAccent),
              ),
            )
          ],
        ),
      ),
      body: TabBarView(
        children: [
          isNewDoc
              ? StreamBuilder(
                  stream: leafDocStream,
                  builder:
                      (context, AsyncSnapshot<DocumentSnapshot> snapshot) {
                    var data;
                    var isProcessed = false;
                    if (snapshot.hasData) {
                      data = snapshot.data as DocumentSnapshot;
                      isProcessed = data['isProcessed'];
                    }
                    if (!isProcessed) {
                      return Center(
                        child: Column(
                          mainAxisAlignment: MainAxisAlignment.center,
                          crossAxisAlignment: CrossAxisAlignment.center,
                          children: [
                            SizedBox(
                              height: height * 0.2,
                              child: const LoadingIndicator(
                                  indicatorType:
                                      Indicator.ballScaleMultiple,

                                  /// Required, The loading type of the widget
                                  colors: [Colors.green],

                                  /// Optional, The color collections
                                  strokeWidth: 2,

                                  /// Optional, The stroke of the line, only applicable to widget which
contains line
                                  // backgroundColor: Colors.grey,       /// Optional, Background of the
widget
                                  pathBackgroundColor: Colors.blueAccent

                                  /// Optional, the stroke backgroundColor
                                  ),
                            ),
```

```
              const Text(
                 "Uploaded image is being Processed Please wait...\n\n"),
              const Text("may take upto 2min"),
            ],
          ),
        );
      } else {
        var arr = data['className'].toString();

        return SingleChildScrollView(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              LeafDetails(
                response: arr,
                maskedImage: data['maskedImage'],
              ),
              Padding(
                padding:
                   const EdgeInsets.symmetric(vertical: 50),
                child: TextButton(
                   style: globalButtonStyle,
                   onPressed: () {
                     isNewDoc = false;
                     setState(() {});
                   },
                   child: const Text(
                     'Analyze another leaf',
                   )),
              ),
            ],
          ),
        );
      }
    })
: (isSelected
    ? SingleChildScrollView(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            const SizedBox(height: 50),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Center(
                  child: SizedBox(
                     height:
                        MediaQuery.of(context).size.height *
```

```
                0.5,
          child: kIsWeb
              ? Image.network(selectedImage!.path)
              : Image.file(
                  File(selectedImage!.path),
                  fit: BoxFit.cover,
                )),
    ),
  ],
),

//todo change to x file
const SizedBox(
  height: 60,
),
TextButton(
    style: globalButtonStyle,
    onPressed: () async {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Processing...'),
        ),
      );
      selectedImageString = await storageService
          .uploadPic(selectedImage!)!;
      await fireStoreService.addLeafDoc(
          l: Leaf(
              img: selectedImageString,
              name: name,
              loc: loc!,
              time: Timestamp.fromDate(
                  DateTime.now())));
      isSelected = false;
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('request sent'),
        ),
      );
      setState(() {});
    },
    child: const Text('analyze')),
const SizedBox(
  height: 20,
),
TextButton(
    style: globalButtonStyle,
    onPressed: () async {
      selectedImage = await imagePicker.pickImage(
          source: ImageSource.camera);
      // if (selectedImages!.isNotEmpty) {
```

```
            //   imageFileList!.addAll(selectedImages);
            // }
            isSelected = true;
            setState(() {});
          },
          child: const Text('retake')),
      ],
    ),
  )
: StreamBuilder(
    stream: dlServerStatusStream,
    builder: (context,
        AsyncSnapshot<DocumentSnapshot> snapshot) {
      bool isActive = false;
      if (snapshot.hasData) {
        var data = snapshot.data as DocumentSnapshot;
        isActive = data['isActive'];
      }

      return Center(
          child: !isActive
              ? const Column(
                  crossAxisAlignment:
                      CrossAxisAlignment.center,
                  mainAxisAlignment:
                      MainAxisAlignment.center,
                  children: [
                    Text(
                      "AI Server is offline",
                    ),
                    Text(
                        "Please make it online to proceed.")
                  ],
                )
              : Column(
                  mainAxisAlignment:
                      MainAxisAlignment.center,
                  children: [
                    Row(
                      mainAxisAlignment:
                          MainAxisAlignment.center,
                      children: [
                        FloatingActionButton(
                          focusColor: Colors.white,
                          foregroundColor: Colors.green,
                          backgroundColor: Colors.black,
                          onPressed: () async {
                            if (name == "") {
                              await Navigator.push(
                                  context,
```

```
            MaterialPageRoute(
                builder: (context) =>
                    const EnterName()));
        }
        selectedImage =
            await imagePicker
                .pickImage(
                    source:
                        ImageSource
                            .camera,
                    imageQuality: 90);

        isSelected = true;
        setState(() {});
      },
      child: const Icon(Icons.camera),
    ),
    const SizedBox(
      width: 20,
    ),
    FloatingActionButton(
      focusColor: Colors.white,
      foregroundColor: Colors.green,
      backgroundColor: Colors.black,
      onPressed: () async {
        if (name == "") {
          await Navigator.push(
              context,
              MaterialPageRoute(
                  builder: (context) =>
                      const EnterName()));
        }
        selectedImage =
            await imagePicker
                .pickImage(
          source: ImageSource.gallery,
        );

        isSelected = true;
        setState(() {});
      },
      child: const Icon(
          Icons.file_upload),
    ),
  ],
),
Padding(
  padding: const EdgeInsets.only(
      top: 30.0),
  child: loc != null
```

```
                        ? Text(
                            "Location : [ ${loc.latitude}    ${loc.longitude} ]")
                        : const SizedBox(),
                    )
                  ],
                ));
          })),
  SingleChildScrollView(
    child: Padding(
      padding: const EdgeInsets.all(18.0),
      child: Container(
        color: Colors.greenAccent,
        width: width * 0.8,
        height: height * 0.7,
        child: GoogleMap(
          initialCameraPosition: _kInitialPosition,
          compassEnabled: true,
          myLocationButtonEnabled: true,
          myLocationEnabled: true,
          markers: Set<Marker>.of(_markers),
          onMapCreated: (GoogleMapController controller) async {
            // _mapConroller.complete(controller);
            getMapPins();
            setState(() {});
            controller.animateCamera(CameraUpdate.newCameraPosition(
                CameraPosition(
                    bearing: 20.0, target: loc2, zoom: 15)));
          },
        ),
      ),
    ),
  )
] )), ) }}
```

4.9 TESTING

Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include preventing bugs, reducing development costs and improving performance.

We tested the Firebase database in localhost to ensure the all the functions are working properly. We also first tested the Flask Server “Figure 21 Flask Server” in our local machine and also used the postman platform to verify its functionality and to measure the Flask server’s response time.

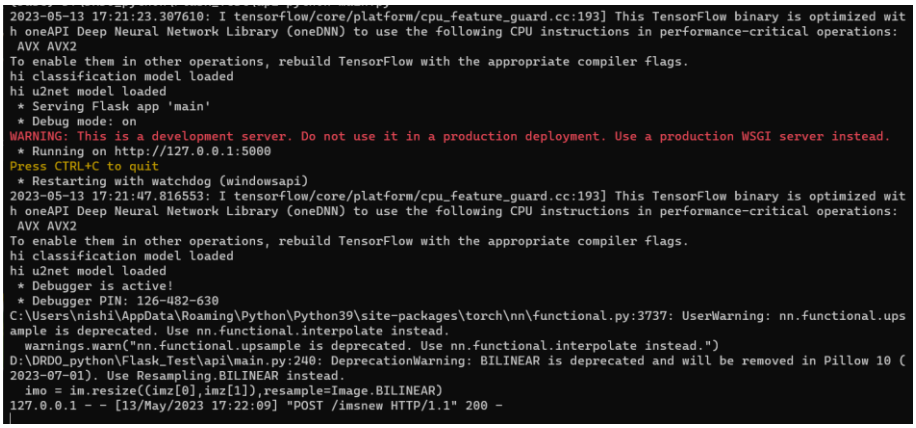


Figure 26 Flask Server

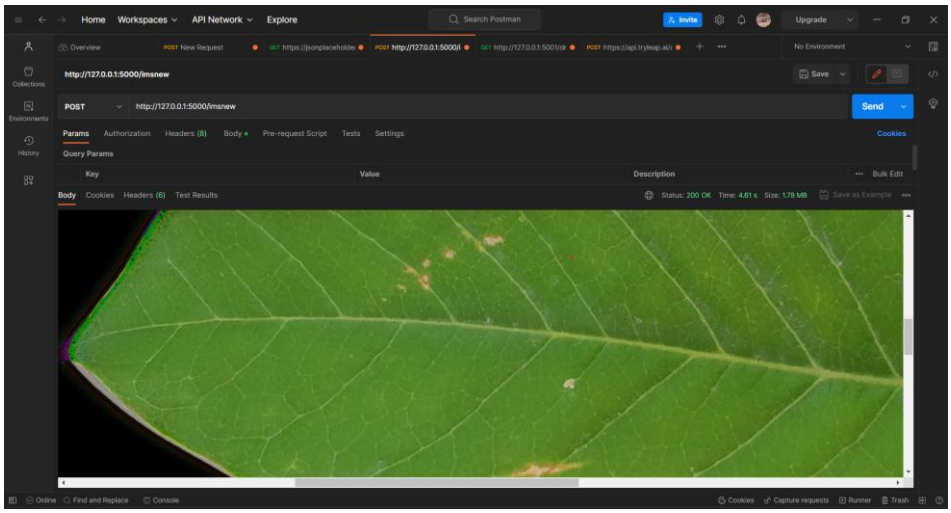


Figure 27 Postman

4.10 RESULTS

4.10.1 U^2-Net Performance

MAE (Mean Absolute Error)-0.072

The model is trained with our leafMX dataset for leaf image segmentation with MAE 0.072.(Mean Absolute Error). The model is also applied on other existing dataset and re-trained the classification model to get better accuracy

4.10.2 Classification Model Performance

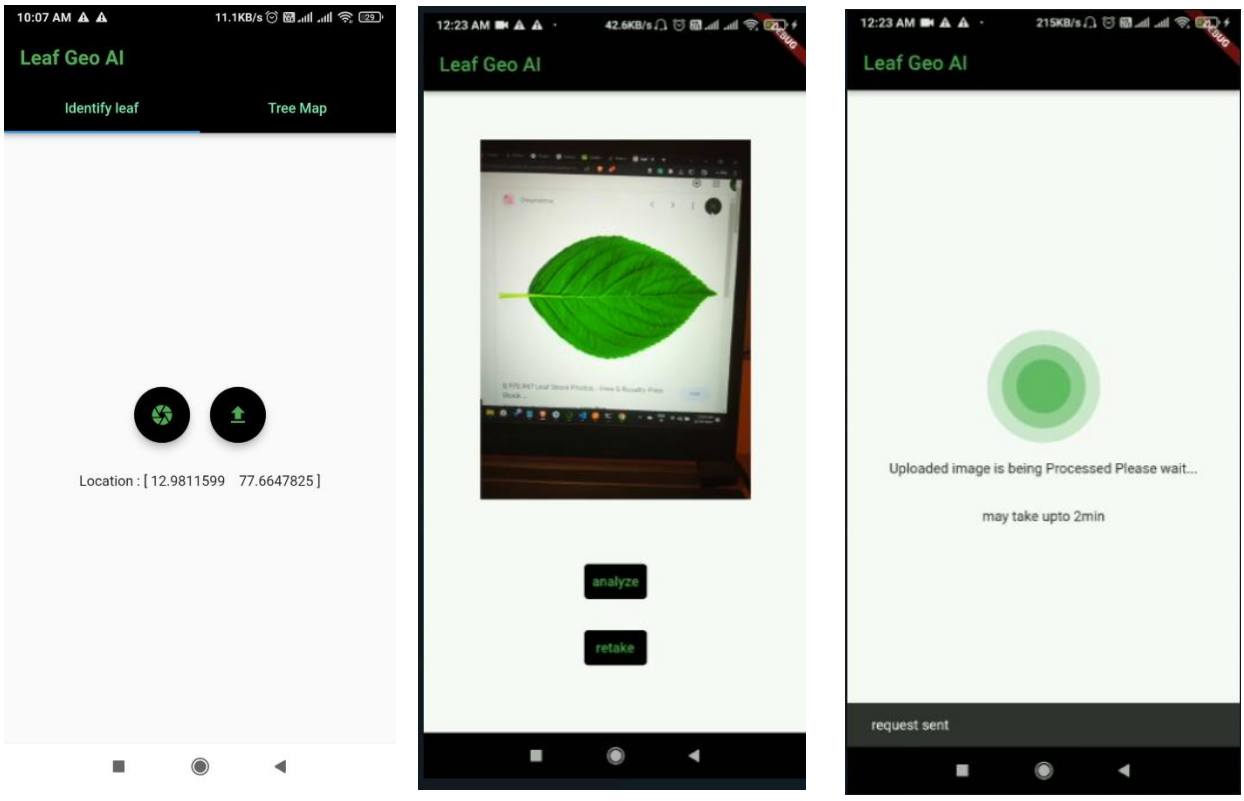
Here we have taken five image classification model and trained it on total 8 datasets, whose performance metrices are given in the below figure-5.

		Flavia	Flavia_masked	Leaf_snap	Leaf_snap_masked	Malaya_kew_D1	Malaya_kew_D2	Swedish_leaf	Swedish_leaf_masked
InceptionV3	Test_Accuracy	0.99	0.99	0.79	0.62	0.94	0.62	0.98	0.59
	Precision	0.99	0.99	0.8	0.6	0.95	0.75	0.98	0.72
	Recall	0.99	0.99	0.77	0.59	0.94	0.62	0.98	0.59
	f1-score	0.99	0.99	0.75	0.57	0.94	0.6	0.98	0.58
VGG16	Test_Accuracy	0.05	0.91	0.02	0.45	0.02	0.02	0.05	0.05
	Precision	0	0.91	0	0.42	0	0	0	0
	Recall	0.03	0.91	0.01	0.4	0.02	0.02	0.07	0.07
	f1-score	0	0.91	0	0.38	0	0	0.01	0.01
ResNet50	Test_Accuracy	0.96	0.98	0.5	0.46	0.2	0.7	0.96	0.95
	Precision	0.96	0.98	0.55	0.5	0.3	0.6	0.96	0.95
	Recall	0.96	0.98	0.47	0.42	0.2	0.7	0.95	0.95
	f1-score	0.96	0.98	0.44	0.4	0.2	0.6	0.96	0.95
MobileNetV2	Test_Accuracy	0.03	0.02	0.55	0.04	0.02	0.7	0.07	0.07
	Precision	0	0	0.59	0.02	0	0.6	0	0
	Recall	0.03	0.03	0.51	0.04	0.02	0.7	0.07	0.07
	f1-score	0	0	0.48	0.02	0	0.7	0.01	0.01
EfficientNetB0	Test_Accuracy	0.78	0.78	0.38	0.27	0.12	0.5	0.84	0.8
	Precision	0.82	0.81	0.38	0.29	0.13	0.4	0.86	0.81
	Recall	0.78	0.77	0.32	0.24	0.12	0.5	0.85	0.8
	f1-score	0.77	0.77	0.3	0.22	0.12	0.5	0.85	0.78

Table 7 classification model performance

From the above table we can observe that after removing the background model accuracy is increasing, which in case confirms our hypothesis that in a image background can act as noise hence removing that can give us better accuracy.

4.10.3 Mobile App Results



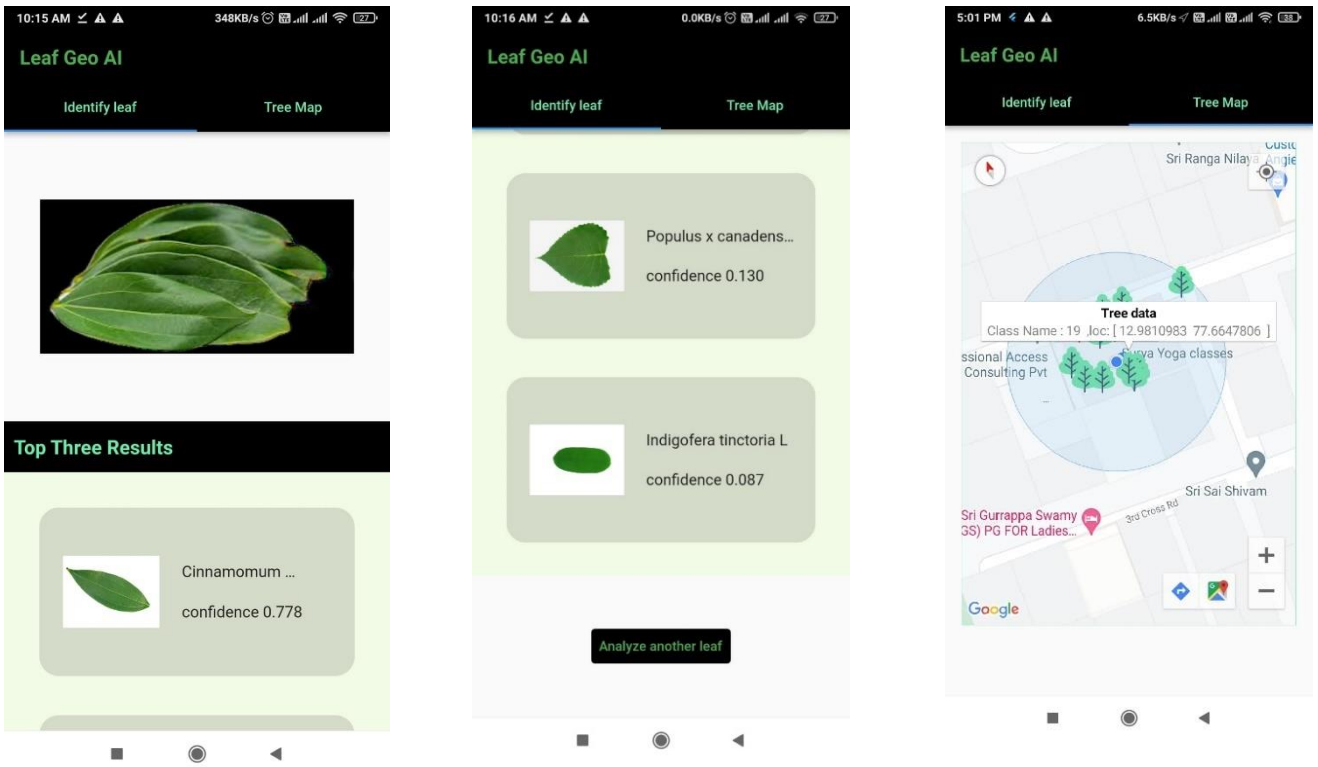


Figure 28 Application Screenshots

CHAPTER 5: CONCLUSION

5.1 Conclusion

The focus of the current work is on a leaf identification system using CNN based Deep learning methods. In fact, the problem of mobile based leaf identification has been investigated by many researchers and still has a large scope of research and study. There were many systems and solutions proposed previously for automatic analysis and classification of leaves. The deep learning techniques are increasingly being employed to automate the plant identification process. There many attempts in past for leaf classification using machine learning and deep learning techniques. The current study also investigates into design of an automated framework for leaf identification using CNN methods.

The work has made a systematic analysis for design of an automated mobile based leaf identification system. The design of such a system involves different activities, viz. mobile app, server, communication architecture, database design, AI models etc. The current study looked into aspects of all these components. Moreover, a protype system is also built to understand and test the system framework. Since the focus is on system level analysis, various open-source applications are utilized to ease the process of implementation. But the same concepts can be exploited and used to build a very robust automated real-time leaf identification system.

5.2 Future Scope

During the work, the mobile app is used to capture a leaf image and the location for further identification process. The location information is currently stored along image data. The classification is achieved using various pretrained CNN models on popular leaf datasets. The location info is only used to highlight the leaf collection at a particular place on a Map. The classification of leaf based on location info is still need to explored. The building of such a dataset along with location information may be future task for the work. The work also limits the study of CNN models to five important ones. The study can be further extended for other Deep learning models for more accuracy.

REFERENCES

[1] S. A. Riaz, S. Naz, and I. Razzak. Multipath deep shallow convolutional networks for large scale plant species identification in wild image. In 2020 International Joint Conference on Neural Networks (IJCNN), pages 1–7. IEEE, 2020.

[2] A. Beikmohammadi and K. Faez. Leaf classification for plant recognition with deep transfer learning. In 2018 4th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS), pages 21–26. IEEE, 2018.

[3] J. Hu, Z. Chen, M. Yang, R. Zhang, and Y. Cui. A multiscale fusion convolutional neural network for plant leaf recognition. IEEE Signal Processing Letters, 25(6):853–857, 2018.

[4] W. Jeon and S. Rhee, "Plant leaf recognition using a convolution neural network", Int. J. Fuzzy Log. Intell. Syst., Vol. 17, No. 1, pp. 26-34, Mar 2017.

[5] H. Zhou, C. Yan and H. Huang, "Tree species identification based on convolutional neural networks", In *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics,* 2016, pp. 103-106.

[6] P. B. Wable, "Neural network-based leaf recognition", In 2016 International Conference on Automatic Control and Dynamic Optimization Techniques, 2016, pp. 645-648.

[7] A. Beikmohammadi and K. Faez, "Leaf classification for plant recognition with deep transfer learning", In 2018 4th Iranian Conference on Signal Processing and Intelligent Systems, 2018, pp. 21- 26.

[8] S. G. Wu, F. S. Bao, E. Y. Xu, Y. Wang, Y. Chang, and Q. Xiang, "A leaf recognition algorithm for plant classification using probabilistic neural network", In *2007 IEEE International Symposium on Signal Processing and Information Technology,* 2007, pp. 11-16.

[9] J. Hang, D. Zhang, J. Zhang and B. Wang, "Classification of plant leaf diseases based on improved convolutional neural network", *Sensors,* Vol. 19, No. 19, pp. 4161, Jan 2019.

[10] Y. Ye, "A computerized plant species recognition system", In *Proceedings of the IEEE 2004 International Symposium on Intelligent Multimedia. Video and Speech Processing,* 2004, pp. 723-726.

[11] J. X. Du, X. F. Wang and G. J. Zhang. "Leaf shape-based plant species recognition", *Appl. Math. Comput.,* Vol. 185, No. 2, pp. 883-893, 2007.

[12] Ramcharan, A.; Baranowski, K.; McCloskey, P.; Ahmed, B.; Legg, J.; Hughes, D.P. Deep Learning for Image-Based Cassava Disease Detection. Front. Plant Sci. 2017, 8, 1852.

[13] Singh, J.; Kaur, H. Plant disease detection based on region-based segmentation and KNN classifier. In Proceedings of the International Conference on ISMAC in Computational Vision and Bio-Engineering 2018, Palladam, India, 16–17 May 2018; pp. 1667–1675.

[14] Lee, S.H.; Chan, C.S.; Mayo, S.J.; Remagnino, P. How deep learning extracts and learns leaf features for plant classification. Pattern Recognition. 2017, 71, 1-13.

[15] B. K. Varghese, A. Augustine, J. M. Babu, D. Sunny, and E. S. Cherian. Infoplant: Plant recognition using convolutional neural networks. In 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), pages 800–807. IEEE, 2020.

[16] https://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/. Oskar J. O. Söderkvist, "Computer vision classifcation of leaves from swedish trees," Master's Thesis, Linkoping University, 2001.

[17] https://education.nationalgeographic.org/resource/biodiversity/

[18] "Leafsnap: A Computer Vision System for Automatic Plant Species Identification," Neeraj Kumar, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida C. Lopez, João V. B. Soares, Proceedings of the 12th European Conference on Computer Vision (ECCV), October 2012

[19] 'Evaluation of Features for Leaf Discrimination', Pedro F.B. Silva, Andre R.S. Marcal, Rubim M. Almeida da Silva (2013). Springer Lecture Notes in Computer Science, Vol. 7950, 197-204.

[20] Pierre Barr´e, Ben C. St¨over, Kai F. M¨uller, and Volker Steinhage. 2017. Leafnet: A computer vision system for automatic plant species identification. Ecological Informatics 40:50 – 56

[21] Sue Han Lee, Chee Seng Chan, Paul Wilkin, and Paolo Remagnino. 2015. Deep-plant: Plant identification with convolutional neural networks. CoRRabs/1506.08425. http://arxiv.org/abs/1506.08425

[22] Sofiene Mouine†, Itheri Yahiaoui†‡, Anne Verroust-Blondet†, Laurent Joyeux†, Souheil Selmi†, Hervé Goëau†, "An Android Application for Leaf-based Plant Identification"

[23] Wäldchen J, Mäder P, "Plant Species Identification Using Computer Vision Techniques: A Systematic Literature Review," 2017, Arch Computat Methods Eng (2018) 25: 507

[24] P. Barré, B.C. Stöver, K.F. Müller, V. Steinhage, LeafNet: A computer vision system for automatic plant species identification

[25] S. H. Lee, C. S. Chan, P. Wilkin, and P. Remagnino, (2015) "Deep- plant: Plant identification with convolutional neural networks."International Conference on Image Processing (IEEE), pp. 452–456, DOI: 10.1109/ICIP.2015.7350839

[26] Jing Hu, Zhibo Chen∗, Meng Yang, Rongguo Zhang and Yaji Cui, (2018) "A Multi-Scale Fusion Convolutional Neural Network for Plant Leaf Recognition." IEEE Signal Processing Letters; vol. 25, pp. 853 – 857.

[27] Mohamed Abbas Hedjazi, IkramKourbane, YakupGenc, (2017) "On Identifying Leaves: A Comparison of CNN with Classical ML Methods." Signal Processing and Communications Applications Conference (SIU), IEEE, pp. 1-4, DOI: 10.1109/SIU.2017.7960257

[28] Bodhwani, Vinit & Acharjya, Debi & Bodhwani, Umesh. (2019). Deep Residual Networks for Plant Identification. Procedia Computer Science. 152. 186-194. 10.1016/j.procs.2019.05.042.

[29] Beikmohammadi, A., Faez, K., & Motallebi, A. (2020). SWP-LeafNET: A novel multistage approach for plant leaf identification based on deep CNN. ArXiv. https://doi.org/10.1016/j.eswa.2022.117470

[30] Zhao, Zhong-Qiu & Ma, Lin-Hai & Cheung, Yiu-ming & Wu, Xindong & Tang, Yuanyan & Chen, C.. (2015). ApLeaf: An efficient android-based plant leaf identification system. Neurocomputing. 151. 1112-1119. 10.1016/j.neucom.2014.02.077.

[31] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv*. /abs/1409.1556

[32] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *ArXiv*. /abs/1512.03385

[33] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. *ArXiv*. /abs/1512.00567

[34] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *ArXiv*. /abs/1801.04381

[35] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ArXiv*. /abs/1905.11946