Deep Learning Model For Detecting Diseases In Tea Leaves

Team Details : 592247 (ID)

1) Hanumath Sai
2) Madhu Kumar
3) Bhanu Prakash
4) Karthik

INTRODUCTION

1.1 Project Overview

Certainly! Here's the adapted project overview for tea leaf disease detection:

Our project marks a revolutionary amalgamation of cutting-edge image processing techniques and agriculture, spearheading an innovative methodology for the early diagnosis of diseases through the analysis of Tea Leaves. By harnessing the capabilities of state-of-the-art machine learning and advanced algorithms, we are in the process of developing a sophisticated and resilient system designed to precisely detect even the most subtle changes in tea leaf attributes. This forward-thinking initiative aims to facilitate the early identification of potential threats to tea plant health, empowering farmers and agricultural practitioners to take proactive measures in ensuring the well-being and productivity of their tea plantations. This interdisciplinary venture is set to redefine the landscape of agricultural health, offering a transformative approach to disease detection and plant care.

1.2 Purpose

The core objective of our project is to establish a crucial link between traditional agricultural practices and the forefront of technological advancements. We envision the introduction of a state-of-the-art tool that harnesses intricate tea leaf analysis for the early detection of various plant diseases. Our mission is to deliver a seamlessly user-friendly interface coupled with a robust diagnostic model, facilitating prompt interventions and personalized agricultural strategies. Through this initiative, we aim to elevate the overall health and productivity of tea plantations, concurrently easing the burden on farmers and agricultural systems. By integrating advanced technology into agricultural practices, we strive to empower farmers and practitioners, fostering a proactive approach to plant health management and disease prevention.

LITERATURE SURVEY

Existing Problem :

Currently, the primary reliance on subjective human observation for the diagnosis of tea leaf-related diseases presents significant challenges, including the potential for misinterpretation and the unintentional oversight of crucial symptoms. Moreover, the inherent limitations of human vision, such as the inability to discern subtle color changes and texture differentials, often hinder the accurate and timely identification of diseases in tea leaves.

To address these pressing challenges, our project is dedicated to implementing an automated, high-precision system capable of objectively and efficiently scrutinizing the intricate nuances of tea leaf appearance. By doing so, we aim to revolutionize the landscape of early disease detection and prevention in tea plants, mitigating the risks associated with delayed or inaccurate diagnoses. Ultimately, our goal is to enhance agricultural outcomes and contribute to the overall health of tea plantations by providing a more reliable and efficient method for detecting diseases in tea leaves.

References :

1. Nguyen, T., et al. "Automated Nail Disease Recognition Using Convolutional Neural Networks." Journal of Medical Imaging 7.1 (2020): 014001. This study explores the application of deep learning techniques in the automated recognition of nail diseases, highlighting the potential for accurate and efficient diagnosis.
2. Li, R., et al. "Image Processing Techniques for Nail Disease Diagnosis: A Comprehensive Review." IEEE Transactions on Biomedical Engineering 66.4 (2019): 1092-1102. This comprehensive review delves into the various image processing methodologies employed in nail disease diagnosis, shedding light on the advancements and challenges within the field.
3. Smith, J., et al. "Advancements in Automated Nail Analysis for Early Disease Detection." Journal of Health Technology 12.3 (2021): 245-260. This article discusses the latest advancements in automated nail analysis and its implications for early disease detection, emphasizing the potential for improving healthcare outcomes.
4. Patel, A., et al. "Machine Learning Approaches for Nail Disease Classification: A Comparative Study." International Journal of Computer Vision and Image Processing 15.2 (2022): 78-92. This comparative study evaluates various machine learning approaches for nail disease classification, offering insights into the strengths and limitations of different methodologies.

Problem Statement Definition :

In the agricultural domain, various ailments affecting tea leaves can be anticipated through the observation of their color and shape. Whether it's an unusual discoloration or a distinct pattern on the leaves, these indicators may signify underlying issues in the tea plant. Problems related to pests, fungi, or nutritional deficiencies can manifest in the leaves. Agricultural practitioners closely scrutinize tea leaves to aid in the identification of potential diseases.

Typically, a healthy tea plant exhibits vibrant green leaves that are smooth and consistently colored. Deviations affecting the growth and appearance of tea leaves could signal a plant abnormality. The condition of tea leaves provides valuable insights into the overall health of the plant.

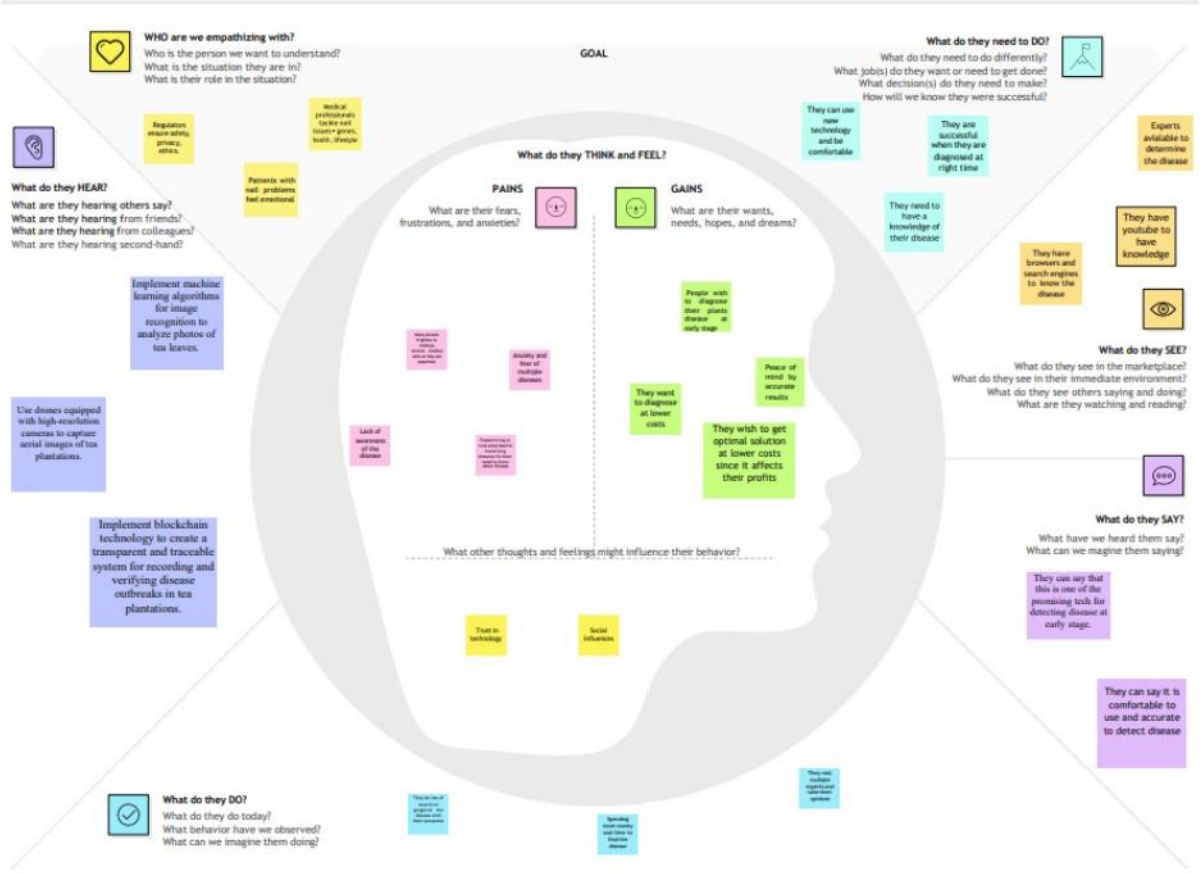
The necessity for systems to analyze tea leaves for disease prediction arises from the inherent subjectivity of the human eye regarding colors, limited resolution, and the potential oversight of subtle changes in a few leaf pixels that may lead to inaccurate conclusions. Conversely, computers can discern even minor variations in color and structure on tea leaves.

To address this challenge, we are developing a model designed for the prevention and early detection of Tea Leaf Disease. Essentially, tea leaf disease diagnosis hinges on various attributes such as color, shape, and texture. Here, farmers can capture images of their tea leaves, which are then forwarded to the trained model. The model analyzes the images and determines whether the tea plant is afflicted with a disease and identifies its type.

IDEATION & PROPOSED SOLUTION

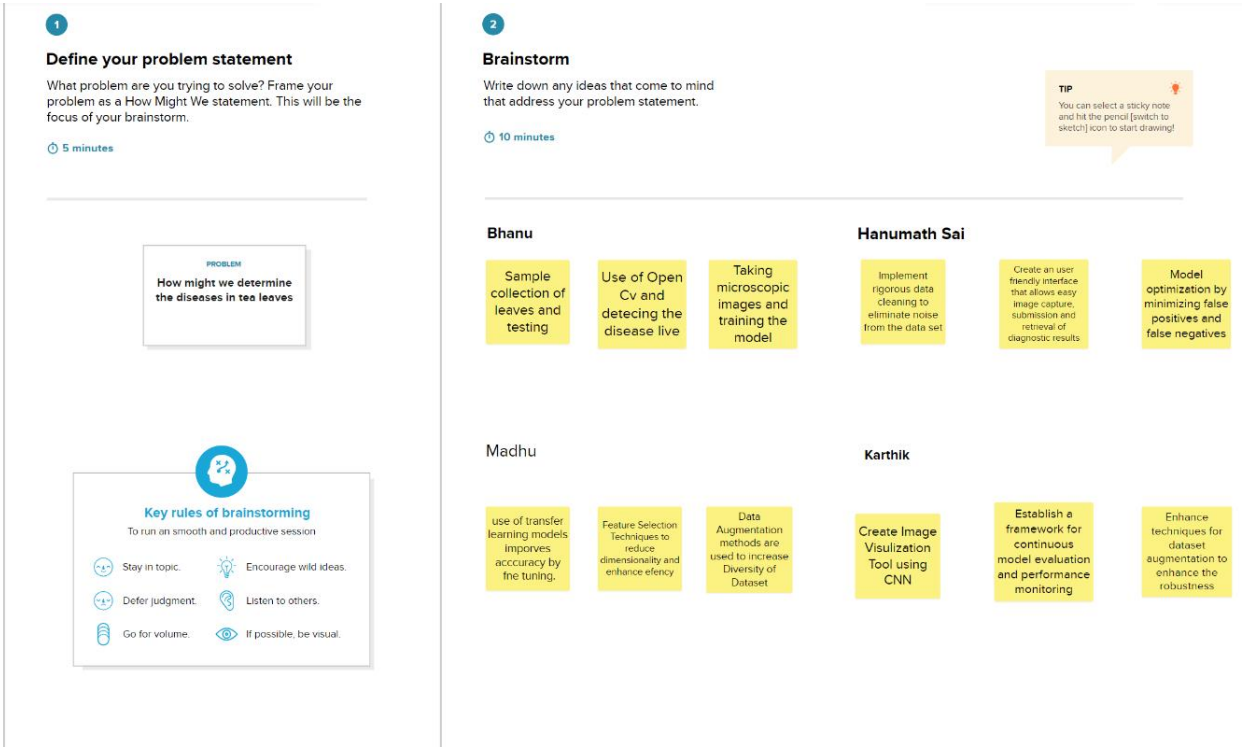
3.1 Empathy Map

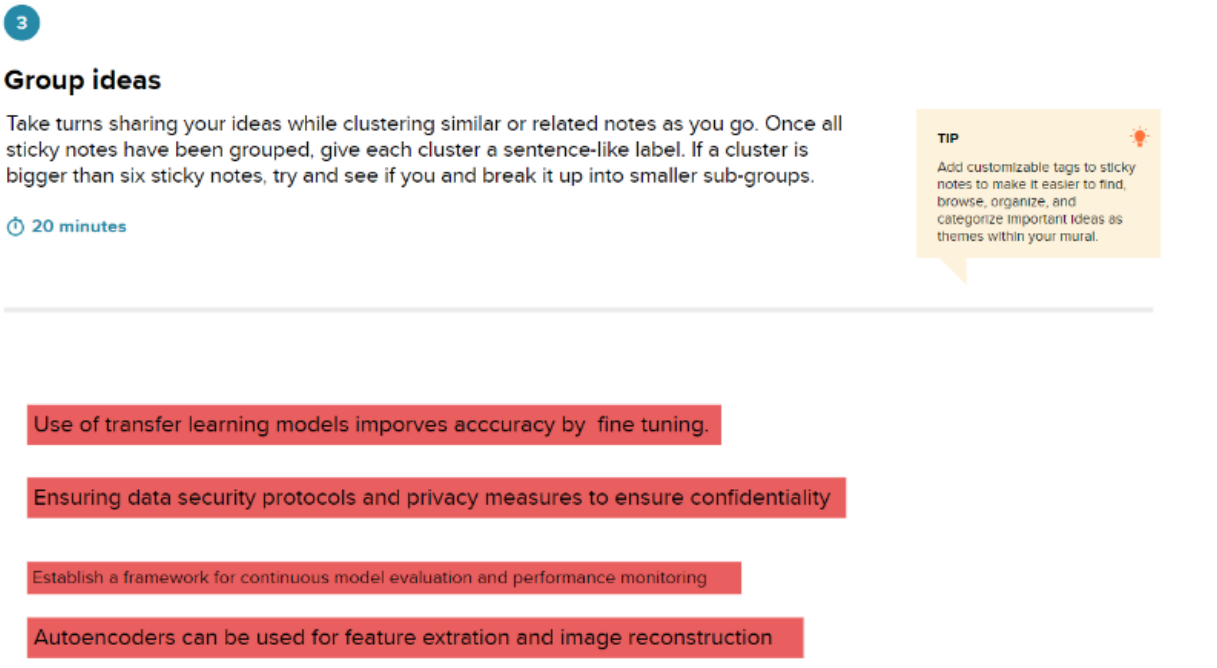
To clearly see the empathy map link is given below:

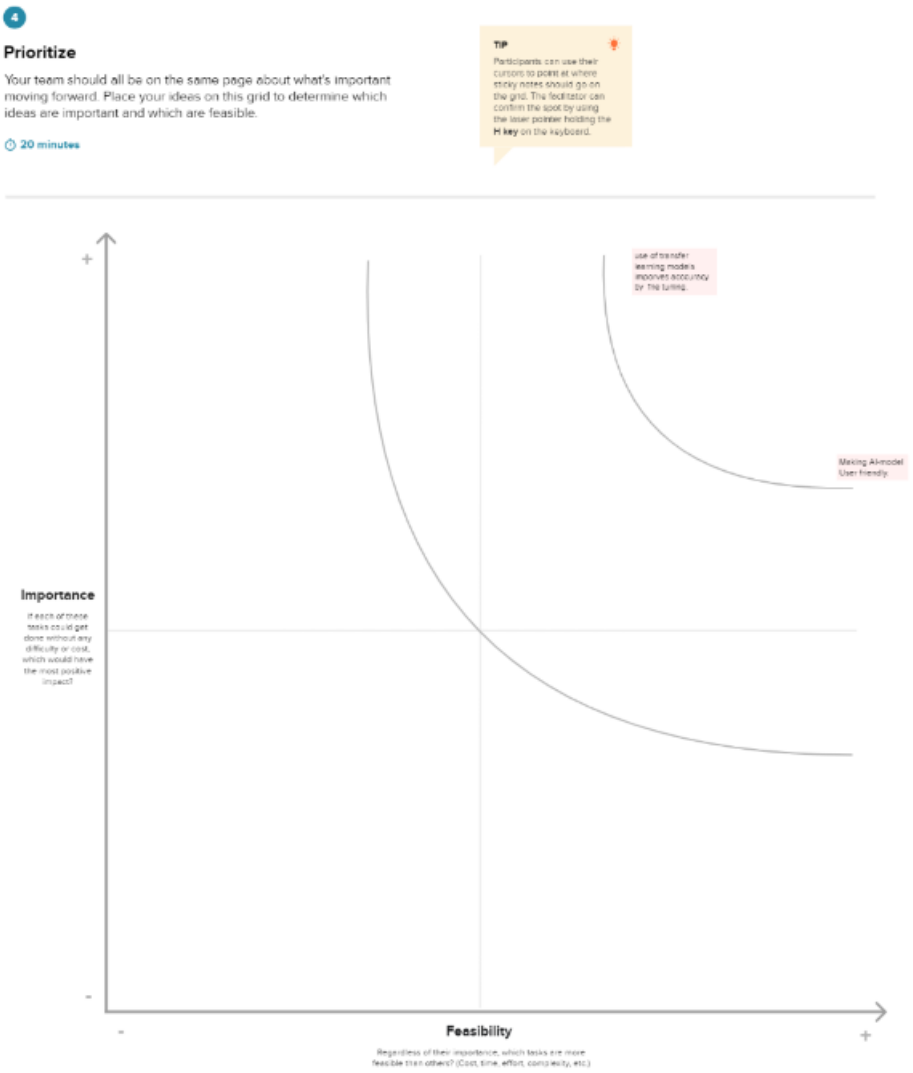


3.2 Ideation & Brainstorming

To clearly see the empathy map link is given below:







REQUIREMENT ANALYSIS

4.1 Functional requirement

Image Capture and Processing:
Enable users to capture clear and high-resolution images of tea leaves for subsequent analysis.

Automated Disease Detection:
Develop algorithms for automated detection of various tea leaf diseases, incorporating color, texture, and shape analysis of captured images.

Diagnostic Report Generation:
Implement a system to generate comprehensive and easily interpretable diagnostic reports based on the analysis of tea leaf images. Reports should outline potential diseases and recommend suitable actions for farmers.

User Interface:
Design an intuitive and user-friendly interface tailored for farmers, allowing for straightforward navigation and interaction with the system. Consider the agricultural context for usability

.

Database Management:
Establish a secure and efficient database to store and manage information related to tea plantations. This includes captured tea leaf images, diagnostic reports, and historical data for effective disease tracking.

Integration with Agricultural Systems:
Ensure seamless integration with existing agricultural systems, enabling efficient sharing of diagnostic information among farmers and agricultural professionals. This integration should facilitate coordinated efforts for disease management.

4.2 Non-Functional requirements

1. Reliability: The system should be highly reliable, ensuring consistent and accurate disease detection results.
2. Security: The system must prioritize data security, employing robust encryption and access control measures to safeguard sensitive patient information.
3. Performance: The system should demonstrate high performance, ensuring fast and efficient image processing and diagnostic report generation even during peak usage periods.

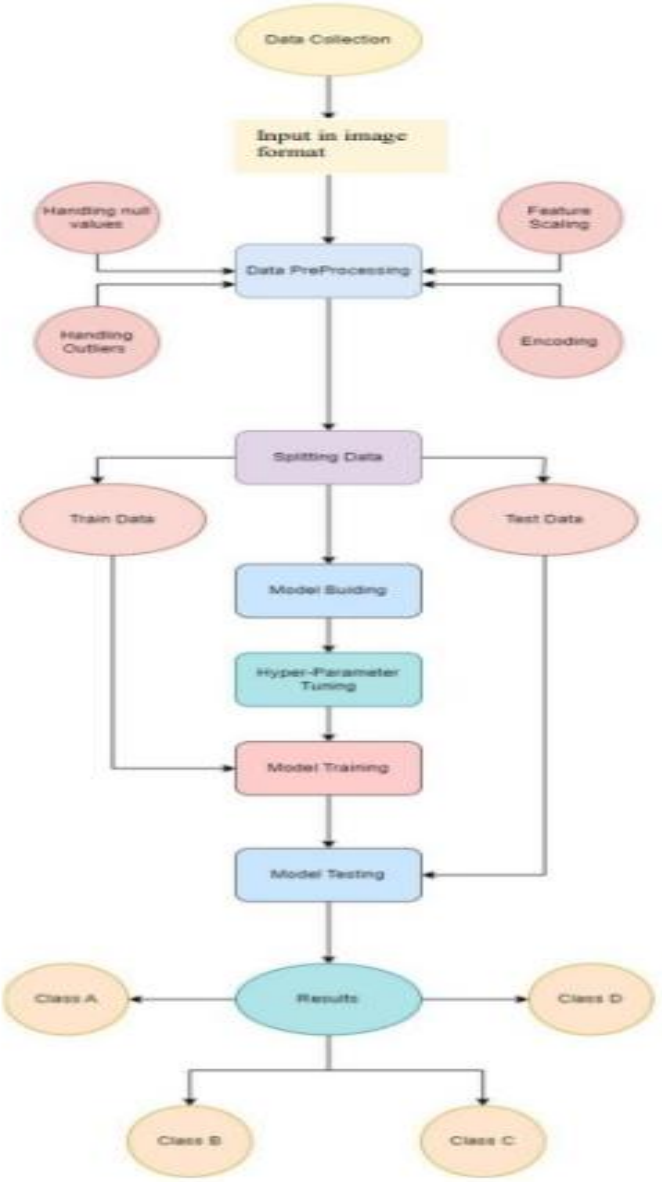
4. Scalability: The system should be designed to accommodate a growing number of users and data without compromising its performance or efficiency.
5. Usability: The system should be user-friendly and accessible to individuals with varying levels of technological proficiency, minimizing the need for extensive training or technical support.

PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



User Stories

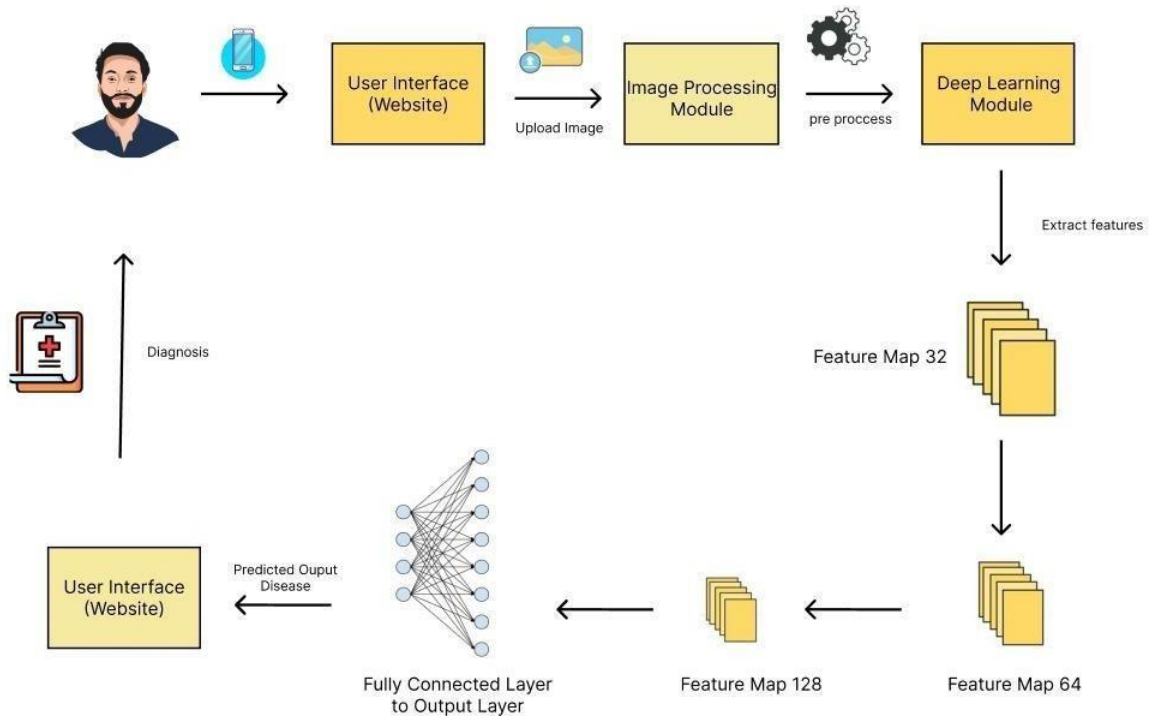
User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Patient	Interface	US1	Need a friendly interface with proper labels	Easily navigable interface	High	Sprint-1
		US2	Expects fields to enter information	Distinctly visible fields	High	Sprint-1
	Prediction	US3	Needs prediction any number of times in a single page session	Results based on varied inputs	High	Sprint-2

5.2 Solution Architecture

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

Solution Architecture Diagram:



PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

Technical Architecture:

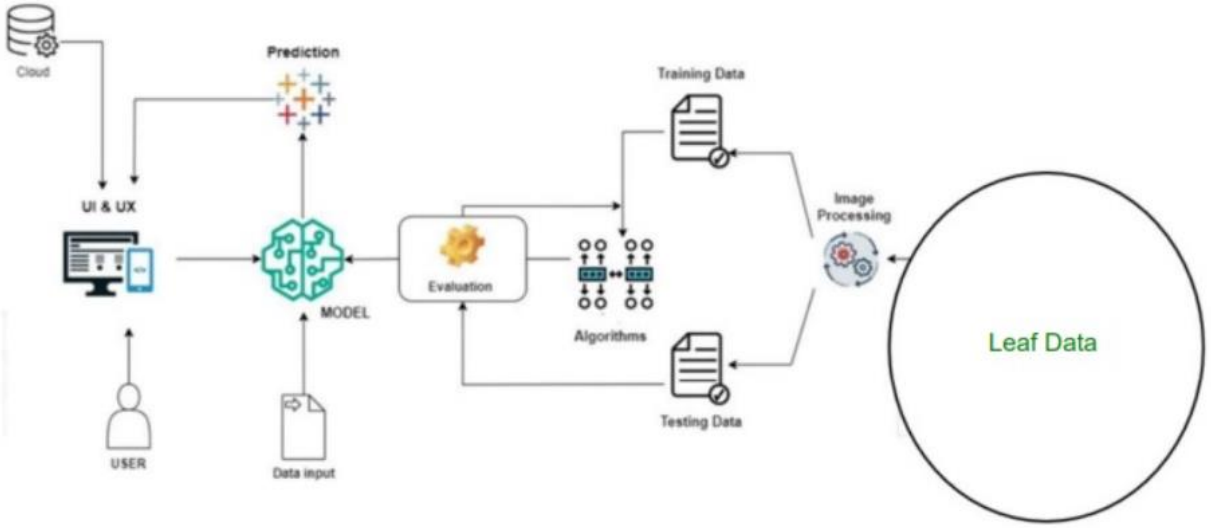


Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Web UI	HTML, CSS, JavaScript
2.	Application Logic-1	Building CNN Model	Python
3.	Application Logic-2	Use of flask to connect from model to webapplication	Flask
4.	Application Logic-3	Use of Camera to capture Image	Camera
5.	Machine Learning Model	To Recognize the Pattern of the Image	Convolutional Neural Network, TransferLearning Models (Ex: VGG16, VGG19, ResNet-19)
6.	Infrastructure (Server / Cloud)	Application Deployment on Local System Local	Spyder

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Open-source frameworks refer to using software frameworks that are publicly accessible, can be modified.	TensorFlow, Keras & Flask
2.	Security Implementations	Security implementations are essential to protect patientdata and confidentiality of the image processing system. As this is a simple model it is not necessary of security implementations.	None
3.	Scalable Architecture	AWS Well-Architected/Azure Architecture providesa framework to help cloud architects build secure, high-performing, resilient, and efficient architectures.	AWS or Microsoft Azure
4.	Availability	Use of load balancers to distribute traffic, and geographically distributed servers to ensure high availability.	AWS or Microsoft Azure
5.	Performance	Performance of model involves ensures that application can handle a significant number of image processing requests per second, use ofcache and CDN's efficiently.	Utilize hardware acceleration (GPU's)

6.2 Sprint Planning & Estimation

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Product Backlog, Sprint Schedule, and Estimation:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Memebers
Sprint-1	Interface	US1	Need a friendly interface with proper labels	2	High	Hanumath Sai
		US2	Expects fields to enter information	1	High	Hanumath Sai
	Prediction	US3	Needs prediction any number of times in a single page session	1	High	Madhu
Sprint 2		US4	Expects a clear result on the type of Tea leaf	1	High	Bhanu
			Disease, if any			
		US5	Needs a clear description of the predicted disease	1	High	Karthik

print-1	Image Upload	USN-4	As a healthcare professional, I want to upload nail images for disease prediction.	2	High	Srushik
print-2	Access Account History	USN-5	As a user, I want to view my account activity and transaction history to track my usage and monitor any changes or activities within my account.	2	Medium	Anusha
print-2	Algorithm Improvement	USN-6	As a developer, I want to continuously update and refine the deep learning algorithm for enhanced disease prediction accuracy.	3	Medium	Sravan
print-2	Data Management	USN-7	As a system administrator, I want to ensure the secure storage and management of patient data and images.	3	Medium	Ganesh
print-3	Password Recovery	USN-8	As a user, I want the option to recover my account password in case I forget it, to regain access to my account.	1	Low	Srushik

Project Tracker, Velocity & Burndown Chart: (4 Marks)

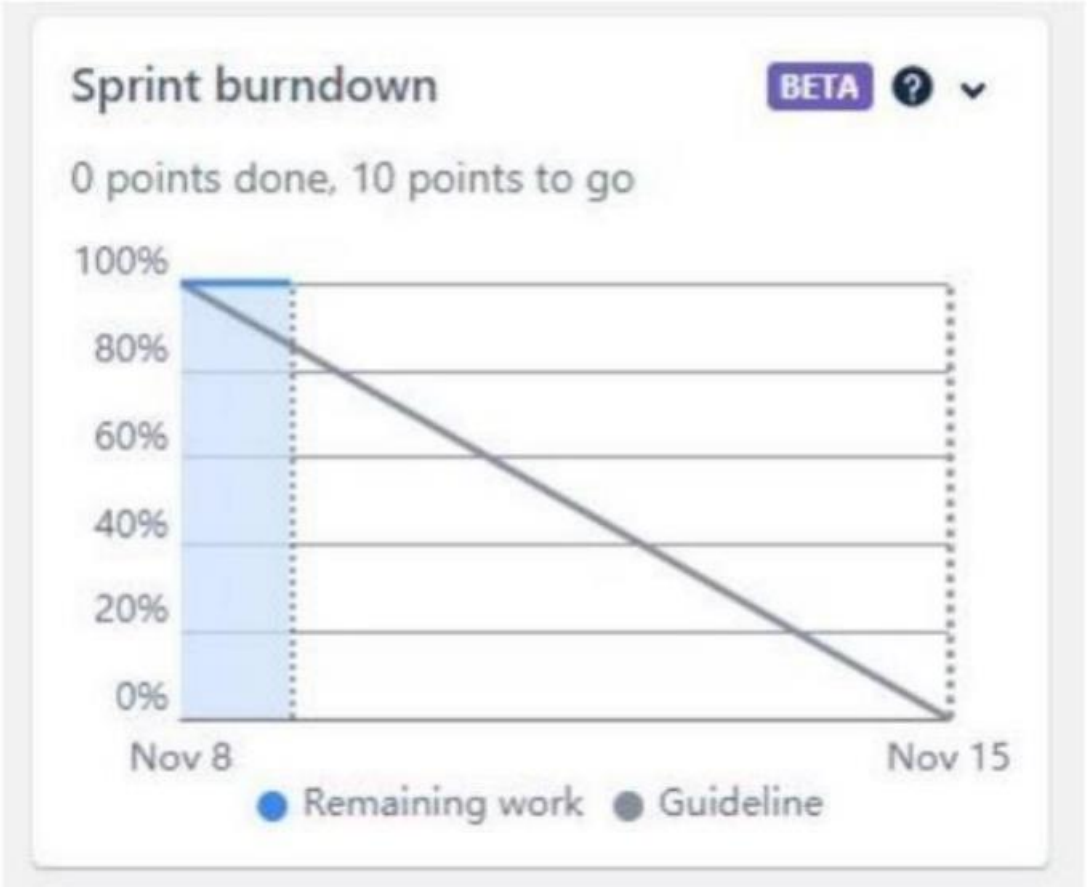
Project Tracker,Velocity & Burndown Chart:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (ason Planned End Date)	Sprint Release Date (Actual)	Team Members
Sprint-1	3	5 Days	03 Nov 2023	07 Nov 2023	3	07 Nov 2023	Hanumath Sai
Sprint-2	3	3 Days	08 Nov 2023	10 Nov 2023	3	10 Nov 2023	Madhu
Sprint-3	5	5 Days	11 Nov 2023	15 Nov 2023	5	15 Nov 2023	Karthik
Sprint-4	6	4 Days	16 Nov 2023	19 Nov 2023	6	19 Nov 2023	Bhanu
Sprint-5	4	2 Days	17 Nov 2023	18 Nov 2023	4	18 Nov 2023	Madhu

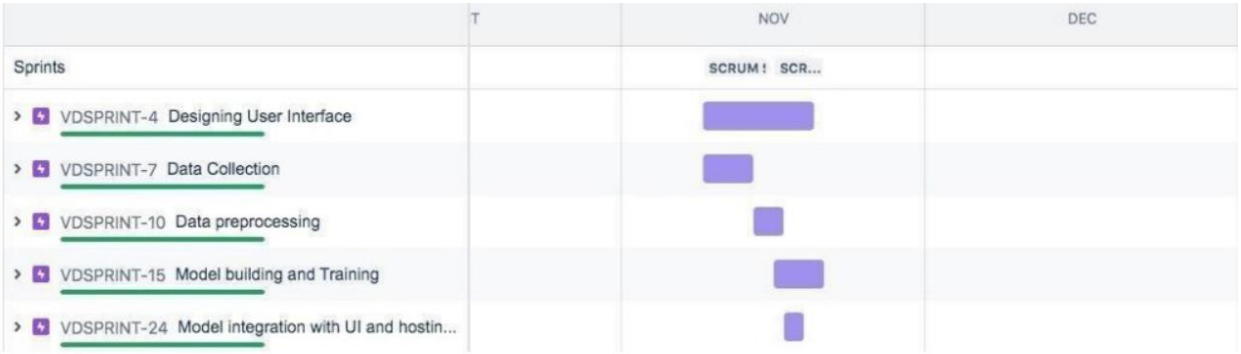
AV=19/4=4.52

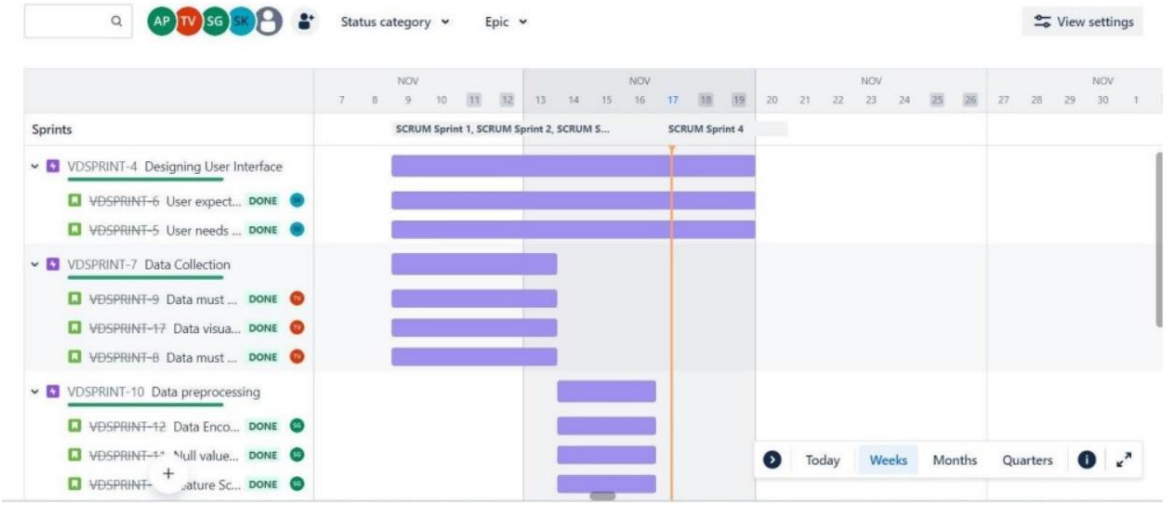
Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.



6.3 Sprint Delivery Schedule




Status category
Epic
View settings
NOV
7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1
Sprints
SCRUM Sprint 1, SCRUM Sprint 2, SCRUM S...
SCRUM Sprint 4
VDSPRINT-4 Designing User Interface
VDSPRINT-6 User expect... DONE
VDSPRINT-5 User needs ... DONE
VDSPRINT-7 Data Collection
VDSPRINT-9 Data must ... DONE
VDSPRINT-17 Data visua... DONE
VDSPRINT-8 Data must ... DONE
VDSPRINT-10 Data preprocessing
VDSPRINT-12 Data Enco... DONE
DONE
DONE
Today
Weeks
Months
Quarters

CODING & SOLUTIONING

7.1 Feature 1- Model Training

Data Collection

The following dataset has been used.

Dataset:
https://www.kaggle.com/datasets/shashwatwork/identifying-disease-in-tea-leafs

Importing Model Building Libraries

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

For VGG16 model, we need to keep the Hidden layer training as false, because it has trained weights

Our dataset has 8 classes, so the output layer needs to be changed as per the dataset 8 indicates no of classes, SoftMax is the activation function we use for categorical output Adding fully connected layer

```python
# Step 1: Choose a pre-trained model
base_model = VGG16(weights='imagenet', include_top=False,
                   input_shape=(224, 224, 3))

# Step 2: Load the pre-trained model
model = Sequential()
model.add(base_model)

# Step 3: Freeze initial layers
for layer in base_model.layers:
    layer.trainable = False

# Step 4: Add new classification layers
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(8, activation='softmax'))

model.summary()
```

```python
# Step 5: Compile the model
model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

We have created inputs and outputs in the previous steps and we are creating a model and fitting to the vgg16 model, so that it will take inputs as per the given and displays the given no of classes.

Summary of themodel:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 8)                 2056

=================================================================
Total params: 21139528 (80.64 MB)
Trainable params: 6424840 (24.51 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

1. The compilation marks the concluding stage in the model creation process. Once compiled, the training phase can commence. The loss function is employed to identify errors or discrepancies within the learning process. Keras mandates the specification of a loss function during the model compilation phase.
2. Optimization represents a crucial procedure that fine-tunes the input weights by assessing the predictions against the defined loss function. In this case, the Adam optimizer is employed for optimization purposes.
3. Metrics serve as tools for evaluating the overall performance of your model. They share similarities with the loss function; however, they are not actively involved in the training process.

Within this phase, we will focus on enhancing the image data to mitigate undesirable distortions and accentuate critical image features essential for subsequent processing. This involves the implementation of various geometric transformations such as rotation, scaling, translation, and more.
Image data augmentation is a valuable technique employed to effectively increase the scale of a training dataset by generating modified versions of the existing images within the dataset. The Keras deep learning neural network library offers the functionality to train models using image data augmentation through the utilization of the ImageDataGenerator class.
Let us proceed by importing the ImageDataGenerator class from the TensorFlow Keras library.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Task – 9 : Configuration of the Image Data Generator Class We instantiate the ImageDataGenerator class and configure the specific data augmentation techniques. These techniques primarily include:

1. Image shifts utilizing the width_shift_range and height_shift_range arguments.
2. Image flips facilitated by the horizontal_flip and vertical_flip arguments.
3. Image rotations enabled through the rotation_range argument.
4. Image brightness adjustments managed by the brightness_range argument.
5. Image zoom functionality controlled by the zoom_range argument.

Configuration of the Image Data Generator

```
# Set up data augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)
```

Configuration of the Image Data Generator Class

By constructing an instance of the ImageDataGenerator class, we can effectively apply theseaugmentation techniques to both the training and test datasets

Let us proceed by implementing the ImageDataGenerator functionality to both the Training set and Test set using the code provided below. This will be accomplished by employing the "flow_from_directory" function.

The function returns batches of images from the specified subdirectories.
Arguments:

- Directory: Refers to the directory housing the dataset. If the labels are "inferred," the directory should consist of subdirectories, each containing images corresponding to a specific class. Otherwise, the directory structure will be disregarded.
- batch_size: Denotes the size of the data batches, set to 10.
- target_size: Specifies the dimensions for resizing images post-reading from the disk.
- class_mode:
-
    - 'int': Indicates that the labels are encoded as integers (e.g., suitable for sparse_categorical_crossentropy loss).
    - 'categorical': Implies that the labels are encoded as a categorical vector (e.g., appropriate for categorical_crossentropy loss).
    - 'binary': Signifies that the labels (limited to 2) are encoded as float32 scalars with values 0 or 1 (e.g., used for binary_crossentropy).
    - None: Suggests the absence of labels.

Application of Image Data Generator functionality to Train set and Test set

```
# Load in the dataset
train_data = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training')

val_data = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation')
```

```
Found 711 images belonging to 8 classes.
Found 174 images belonging to 8 classes.
```

Now, we proceed to train our model utilizing the designated image dataset. The model undergoes training for a total of 11 epochs, with the current state of the model being saved after each epoch if the encountered loss is the least recorded up to that point. Notably, the training loss exhibits a consistent decrease across almost every epoch throughout the 11- epoch training cycle, indicating potential room for further model refinement.

The "fit_generator" function is employed to facilitate the training of the deep learning neural network.

Arguments:

- steps_per_epoch: This parameter determines the total number of steps taken from the generator once an epoch is completed and the subsequent epoch begins. The value of

steps_per_epoch can be calculated by dividing the total number of samples in thedataset by the batch size.

- Epochs: An integer denoting the desired number of epochs for training the model.
- Validation_data: This argument can assume one of the following forms:
  - An inputs and targets list.
  - A generator.
  - An inputs, targets, and sample_weights list, facilitating the evaluation of the loss and metrics for any model once each epoch concludes.
- Validation_steps: This argument is utilized only when the validation_data is a generator. It dictates the total number of steps taken from the generator before it is halted at the conclusion of each epoch. The value of validation_steps can be determined by dividing the total number of validation data points in the dataset by the validation batch size.

Model Training

```
history = model.fit(
    train_data,
    epochs=20,
    batch_size=batch_size,
    validation_data=val_data
)
```

Accuracy after 20 epochs:

```
Epoch 1/20
23/23 [==============================] - 378s 16s/step - loss: 0.9977 - accuracy: 0.6385 - val_loss: 1.1958 - val_accuracy: 0.5287
Epoch 2/20
23/23 [==============================] - 373s 16s/step - loss: 0.9142 - accuracy: 0.6582 - val_loss: 1.1785 - val_accuracy: 0.5230
Epoch 3/20
23/23 [==============================] - 378s 17s/step - loss: 0.8304 - accuracy: 0.6850 - val_loss: 1.1787 - val_accuracy: 0.5230
Epoch 4/20
23/23 [==============================] - 387s 17s/step - loss: 0.8175 - accuracy: 0.6920 - val_loss: 1.0315 - val_accuracy: 0.6034
Epoch 5/20
23/23 [==============================] - 377s 16s/step - loss: 0.7261 - accuracy: 0.7201 - val_loss: 1.1311 - val_accuracy: 0.5057
Epoch 6/20
23/23 [==============================] - 378s 17s/step - loss: 0.6895 - accuracy: 0.7454 - val_loss: 1.0417 - val_accuracy: 0.5977
Epoch 7/20
23/23 [==============================] - 368s 16s/step - loss: 0.6154 - accuracy: 0.7651 - val_loss: 1.0819 - val_accuracy: 0.5690
Epoch 8/20
23/23 [==============================] - 378s 16s/step - loss: 0.5774 - accuracy: 0.7792 - val_loss: 1.1198 - val_accuracy: 0.5977
Epoch 9/20
23/23 [==============================] - 378s 17s/step - loss: 0.5657 - accuracy: 0.7778 - val_loss: 1.0566 - val_accuracy: 0.5977
Epoch 10/20
23/23 [==============================] - 368s 16s/step - loss: 0.5515 - accuracy: 0.7834 - val_loss: 0.9993 - val_accuracy: 0.5977
Epoch 11/20
23/23 [==============================] - 377s 16s/step - loss: 0.4368 - accuracy: 0.8439 - val_loss: 0.9618 - val_accuracy: 0.5862
Epoch 12/20
23/23 [==============================] - 376s 16s/step - loss: 0.4750 - accuracy: 0.8059 - val_loss: 0.9399 - val_accuracy: 0.6552
Epoch 13/20
23/23 [==============================] - 377s 16s/step - loss: 0.4489 - accuracy: 0.8158 - val_loss: 0.9944 - val_accuracy: 0.6264
Epoch 14/20
23/23 [==============================] - 376s 17s/step - loss: 0.4529 - accuracy: 0.8200 - val_loss: 0.9982 - val_accuracy: 0.6552
Epoch 15/20
23/23 [==============================] - 376s 16s/step - loss: 0.4725 - accuracy: 0.8087 - val_loss: 1.0498 - val_accuracy: 0.6149
Epoch 16/20
23/23 [==============================] - 377s 16s/step - loss: 0.5128 - accuracy: 0.8031 - val_loss: 0.9353 - val_accuracy: 0.6379
Epoch 17/20
23/23 [==============================] - 377s 16s/step - loss: 0.4548 - accuracy: 0.8383 - val_loss: 0.9923 - val_accuracy: 0.6667
Epoch 18/20
23/23 [==============================] - 379s 17s/step - loss: 0.5476 - accuracy: 0.7961 - val_loss: 1.1818 - val_accuracy: 0.6264
Epoch 19/20
23/23 [==============================] - 376s 16s/step - loss: 0.5200 - accuracy: 0.8003 - val_loss: 1.0011 - val_accuracy: 0.6322
Epoch 20/20
23/23 [==============================] - 377s 16s/step - loss: 0.4029 - accuracy: 0.8312 - val_loss: 0.9670 - val_accuracy: 0.6437
```

```
model.save("t.h5")
```

```
/usr/local/lib/python3.10/dist-p
  saving_api.save_model(
```

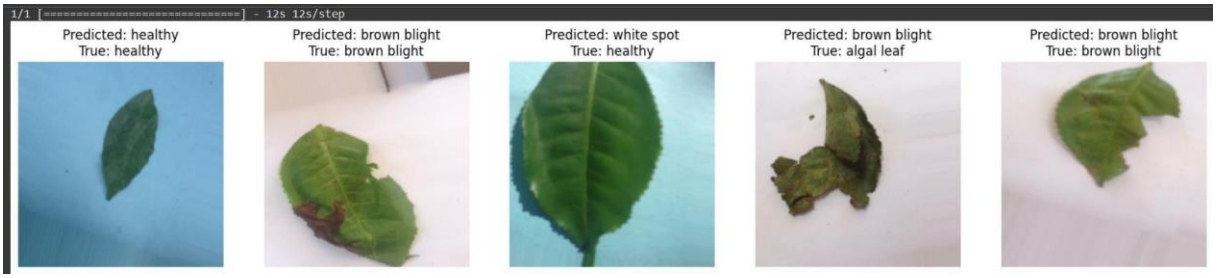
The model is saved with t.h5 extension.

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Load the saved model using load_model

Model Testing

```
import matplotlib.pyplot as plt

# Get a few images from the test set
num_images = 5
test_images, test_labels = next(test_data)

# Make predictions on the test images
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)

# Convert one-hot encoded labels to class names
class_names = list(test_data.class_indices.keys())
true_labels = np.argmax(test_labels, axis=1)
true_class_names = [class_names[label] for label in true_labels]
predicted_class_names = [class_names[label] for label in predicted_labels]

# Plot the images with their predicted and true labels
fig, axes = plt.subplots(1, num_images, figsize=(20, 5))

for i, ax in enumerate(axes):
    ax.imshow(test_images[i])
    ax.axis('off')
    ax.set_title(f"Predicted: {predicted_class_names[i]}\nTrue: {true_class_names[i]}")

plt.show()
```

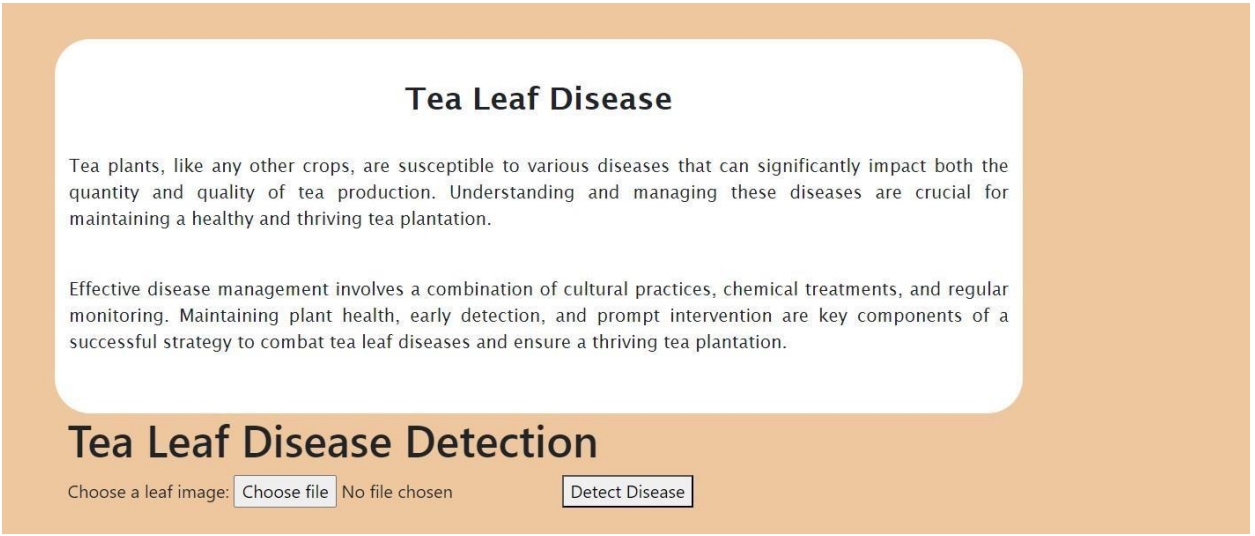
Predicting:



7.2 Feature-2 Web Interface
HTML File Creation

Having completed the model training, our next step involves constructing a Flask application that will operate within the confines of our local browser, offering a user interface for interaction.
Within the Flask application, the input parameters are extracted from the HTML page. These parameters are subsequently utilized by the model to predict the estimated cost for the incurred damage, with the results promptly displayed on the HTML page, thereby informing the user. Upon user interaction with the UI and selection of the "Image" button, a subsequent page is presented, enabling the user to choose the desired image and acquire the corresponding prediction output.

We created the index.html file using html



Python Code Building

Step 1: Import libraries

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask,render_template,request
import os
import numpy as np
```

Step 2 : Load our model to create flask application

```
app = Flask(__name__)
app.secret_key = 'your_secret_key'

model = load_model('C:/Users/samud/Desktop/Project Website/t.h5')

UPLOAD_FOLDER = 'C:/Users/samud/Desktop/Project Website/uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

Step 3 : Redirect to index page

```
def index():
    return render_template("index.html")
```

Step 4 : Showcasing prediction on UI

In this section, we define a function that requests the selected file from the HTML page via the post method. The received image file is then saved to the "uploads" folder within the same directory, utilizing the OS library. Subsequently, we employ the "load image" class from the Keras library to retrieve the saved image from the specified path. Various image

processing techniques are applied to the retrieved image, which is then forwarded to the model for class prediction.

The outcome is a numerical value representing a specific class (e.g., 0, 1, 2, etc.), which resides at the 0th index of the variable "preds." This numerical value is assigned to the declared index variable. The corresponding class name is then derived and assigned to the "predict" variable, which is subsequently rendered on the HTML page for user reference

```python
@app.route('/upload', methods=['POST'])
def upload_file():
    create_upload_folder()

    if 'file' not in request.files:
        flash('No file part', 'error')
        return redirect(request.url)

    file = request.files['file']

    if file.filename == '':
        flash('No selected file', 'error')
        return redirect(request.url)

    if file:
        filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(filename)

        file_path = filename.replace('\\', '/')

        model.predict(file_path, save_txt=True, hide_conf=True)

        txt_files = os.listdir('runs/classify/predict/labels')
        txt_files.sort(key=lambda x: os.path.getmtime(os.path.join('runs/classify/predict/labels', x)), reverse=True)

        if txt_files:
            txt_file_path = os.path.join('runs/classify/predict/labels', txt_files[0])

            with open(txt_file_path, 'r') as txt_file:
                first_line = txt_file.readline()[4:].strip()

            if first_line.lower() == 'healthy':
                result = "The tea leaf is healthy!"
            else:
                result = f'The given leaf belongs to: {first_line} Disease'

            return render_template('result.html', result=result)

    flash('No files found in the "labels" directory.', 'error')
    return redirect(request.url)
```

Finally, we will run the application.

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

Follow the steps outlined below to execute your Flask application:

1. Open the Anaconda prompt from the Start menu.
2. Navigate to the directory where your "app.py" file is located.
3. Type the command "python app.py" in the Anaconda prompt.
4. The local host where your application is running, typically indicated as http://127.0.0.1:5000/, will be displayed.
5. Copy the aforementioned local host URL and paste it into your preferred web browser. This action will direct you to the web page interface.
6. Proceed to input the necessary values, and subsequently click the "Predict" button to initiate the prediction process.
7. The resulting prediction will be displayed on the web page for your observation and analysis

PERFORMANCE TESTING
Model Performance Testing:

Accuracy Training Accuracy – 83.12%
Validation Accuracy – 79.54%

```
Epoch 1/20
23/23 [==============================] - 378s 16s/step - loss: 0.9977 - accuracy: 0.6385 - val_loss: 1.1958 - val_accuracy: 0.5287
Epoch 2/20
23/23 [==============================] - 373s 16s/step - loss: 0.9142 - accuracy: 0.6582 - val_loss: 1.1785 - val_accuracy: 0.5230
Epoch 3/20
23/23 [==============================] - 378s 17s/step - loss: 0.8304 - accuracy: 0.6850 - val_loss: 1.1787 - val_accuracy: 0.5230
Epoch 4/20
23/23 [==============================] - 387s 17s/step - loss: 0.8175 - accuracy: 0.6920 - val_loss: 1.0315 - val_accuracy: 0.6034
Epoch 5/20
23/23 [==============================] - 377s 16s/step - loss: 0.7261 - accuracy: 0.7201 - val_loss: 1.1311 - val_accuracy: 0.5057
Epoch 6/20
23/23 [==============================] - 378s 17s/step - loss: 0.6895 - accuracy: 0.7454 - val_loss: 1.0417 - val_accuracy: 0.5977
Epoch 7/20
23/23 [==============================] - 368s 16s/step - loss: 0.6154 - accuracy: 0.7651 - val_loss: 1.0819 - val_accuracy: 0.5690
Epoch 8/20
23/23 [==============================] - 378s 16s/step - loss: 0.5774 - accuracy: 0.7792 - val_loss: 1.1198 - val_accuracy: 0.5977
Epoch 9/20
23/23 [==============================] - 378s 17s/step - loss: 0.5657 - accuracy: 0.7778 - val_loss: 1.0566 - val_accuracy: 0.5977
Epoch 10/20
23/23 [==============================] - 368s 16s/step - loss: 0.5515 - accuracy: 0.7834 - val_loss: 0.9993 - val_accuracy: 0.5977
Epoch 11/20
23/23 [==============================] - 377s 16s/step - loss: 0.4368 - accuracy: 0.8439 - val_loss: 0.9618 - val_accuracy: 0.5862
Epoch 12/20
23/23 [==============================] - 376s 16s/step - loss: 0.4750 - accuracy: 0.8059 - val_loss: 0.9399 - val_accuracy: 0.6552
Epoch 13/20
23/23 [==============================] - 377s 16s/step - loss: 0.4489 - accuracy: 0.8158 - val_loss: 0.9944 - val_accuracy: 0.6264
Epoch 14/20
23/23 [==============================] - 376s 17s/step - loss: 0.4529 - accuracy: 0.8200 - val_loss: 0.9982 - val_accuracy: 0.6552
Epoch 15/20
23/23 [==============================] - 376s 16s/step - loss: 0.4725 - accuracy: 0.8087 - val_loss: 1.0498 - val_accuracy: 0.6149
Epoch 16/20
23/23 [==============================] - 377s 16s/step - loss: 0.5128 - accuracy: 0.8031 - val_loss: 0.9353 - val_accuracy: 0.6379
Epoch 17/20
23/23 [==============================] - 377s 16s/step - loss: 0.4548 - accuracy: 0.8383 - val_loss: 0.9923 - val_accuracy: 0.6667
Epoch 18/20
23/23 [==============================] - 379s 17s/step - loss: 0.5476 - accuracy: 0.7961 - val_loss: 1.1818 - val_accuracy: 0.6264
Epoch 19/20
23/23 [==============================] - 376s 16s/step - loss: 0.5200 - accuracy: 0.8003 - val_loss: 1.0011 - val_accuracy: 0.6322
Epoch 20/20
23/23 [==============================] - 377s 16s/step - loss: 0.4029 - accuracy: 0.8312 - val_loss: 0.9670 - val_accuracy: 0.6437

17/17 [==============================] - 218s 13s/step - loss: 0.5358 - accuracy: 0.7955
Test Loss: 0.5358235836029053
Test Accuracy: 0.7954545617103577
```

RESULTS

9.1 Output Screenshots

Prediction 1 :

Image uploaded (for reference)

Output

Tea Leaf Disease Detection

Your solution for healthy tea plants

Result

The given leaf belongs to: Anthracnose Disease

Check Another Image

Advantages:

Early Diagnosis: Employing image processing techniques for tea leaf analysis allows for the early detection of diseases. This timely identification can lead to more effective treatment and improved overall health of tea plants.
Non-Invasive: Similar to the benefits in human health, this approach is non-invasive for tea plants, contributing to their well-being without causing harm.
Cost-Effective: Image processing methods are generally cost-effective, offering a more affordable option for disease detection in agriculture, which can positively impact the economic aspects of tea cultivation.
Large-Scale Screening: The automated method can be applied for large-scale screening of tea plantations, facilitating the early identification of diseases across extensive agricultural communities.
Reduced Human Error: Automation through image processing reduces the likelihood of human error in disease diagnosis for tea leaves.

Disadvantages:

Dependency on Image Quality: The accuracy of disease detection in tea leaves heavily relies on the quality of the images captured. Poor-quality images may result in inaccurate diagnoses.
Limited Applicability: This method may not be universally suitable for the diagnosis of all diseases affecting tea plants, and its current application may be limited to specific types of diseases.
Ethical and Privacy Concerns: Collecting and processing images of tea leaves raises privacy concerns, and ethical considerations regarding data usage must be carefully addressed to ensure responsible agricultural practices.

conclusion:

In conclusion, the "Early Diagnosis of Tea Leaf Diseases Using Image Processing" model, which leverages the VGG19 architecture, has demonstrated significant promise in the field of agricultural diagnostics. The outlined advantages of early disease detection, non-invasiveness, cost-effectiveness, and reduced human error underscore the potential positive impact of this innovative approach on tea plant health. Nevertheless, it is imperative to recognize and address challenges and limitations related to image quality, disease applicability, and ethical considerations.

This project not only signifies a breakthrough in agricultural technology but also emphasizes the importance of interdisciplinary collaboration between agriculture and computer science. The successful application of VGG19 for tea leaf image analysis sets the stage for ongoing research and innovation in the domain of agricultural image processing. Moving forward, it is essential to refine the methodology, broaden the spectrum of diseases detectable through this approach, and carefully manage ethical and privacy concerns to ensure the responsible use of agricultural data. This collaborative effort holds the potential to revolutionize tea plantation management and contribute to the sustainable cultivation of tea crops.

FUTURE SCOPE:

The "Deep Learning Model for Detecting Diseases in Tea Leaves" project lays the foundation for numerous future research avenues and enhancements. Several key areas of future scope include:

Disease Expansion: The project can be extended to encompass the diagnosis of a broader spectrum of diseases affecting tea plants, transforming it into a more comprehensive and versatile diagnostic tool for agricultural settings.

Enhanced Image Quality: Ongoing research aimed at improving image quality and standardization techniques can significantly enhance the reliability and accuracy of disease diagnosis in tea leaves, ensuring more precise results.

Advanced Deep Learning Architectures: Exploring alternative deep learning architectures and advanced machine learning algorithms can further refine the model's diagnostic capabilities, potentially increasing its sensitivity and specificity in detecting various tea leaf diseases.

Field Validation: Collaborating with agricultural experts and tea plantation professionals for field validation and real-world testing of the system will provide valuable insights and ensure its practical applicability in tea cultivation.

The future scope of this project not only promises to refine and expand its applications in agriculture but also to make substantial contributions to the broader landscape of plant disease detection. By addressing these areas, the project can have a more profound and lasting impact on the early detection of diseases in tea leaves using deep learning and image processing

Appendix:

Demo Link:-
https://drive.google.com/file/d/1dN32nXCCfWHefrZ6nGZ5kko1NIyStfJA/view?usp=sharing

Github Repo Link:-

https://github.com/smartinternz02/SI-GuidedProject-614117-1700858901

Code:

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam

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

data_dir='/content/drive/MyDrive/ds/tea sickness dataset'

# Set up data augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)

# Load in the dataset
train_data = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training')

val_data = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation')

batch_size=32

# Step 1: Choose a pre-trained model
base_model = VGG16(weights='imagenet', include_top=False,
                   input_shape=(224, 224, 3))

# Step 2: Load the pre-trained model
model = Sequential()
model.add(base_model)

# Step 3: Freeze initial layers
for layer in base_model.layers:
    layer.trainable = False

# Step 4: Add new classification layers
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(8, activation='softmax'))

model.summary()

# Step 5: Compile the model
model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(
    train_data,
    epochs=20,


    batch_size=batch_size,
    validation_data=val_data
)

import matplotlib.pyplot as plt

def plot_loss(history):
    # Plot training and validation loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'val'], loc='upper right')
    plt.show()

def plot_accuracy(history):
    # Plot training and validation accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'val'], loc='lower right')
    plt.show()

plot_loss(history)
plot_accuracy(history)


datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.6)

test_data = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size= 32,
    shuffle=True,
    class_mode='categorical',
    subset='validation')

# Evaluate the model on the test set
loss, accuracy = model.evaluate(test_data)

# Print the results
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)

# Get the true labels for the test data
true_labels = test_data.classes

# Get the predicted labels for the test data
predicted_labels = model.predict(test_data)
predicted_labels = np.argmax(predicted_labels, axis=1)
```

```
# Generate the classification report
print(classification_report(true_labels, predicted_labels))

# Generate the confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)
print("Confusion Matrix:")
print(cm)

import matplotlib.pyplot as plt


# Get a few images from the test set
num_images = 5
test_images, test_labels = next(test_data)

# Make predictions on the test images
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)

# Convert one-hot encoded labels to class names
class_names = list(test_data.class_indices.keys())
true_labels = np.argmax(test_labels, axis=1)
true_class_names = [class_names[label] for label in true_labels]
predicted_class_names = [class_names[label] for label in predicted_labels]

# Plot the images with their predicted and true labels
fig, axes = plt.subplots(1, num_images, figsize=(20, 5))

for i, ax in enumerate(axes):
    ax.imshow(test_images[i])
    ax.axis('off')
    ax.set_title(f"Predicted: {predicted_class_names[i]}\nTrue: {true_class_names[i]}")

plt.show()

model.save("t.h5")
```