# Deep Learning Model For Detecting Diseases In Tea Leaves

## Introduction:

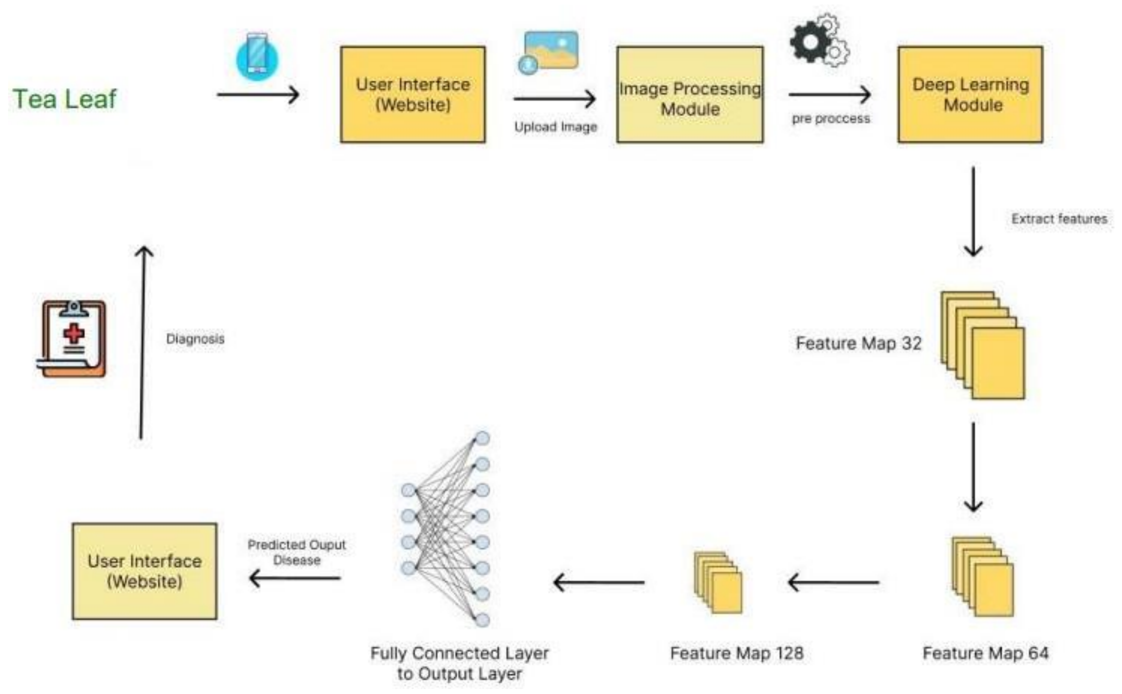
In the realm of agriculture, the observation of tea leaves' color and shape can often forewarn the presence of various diseases. Be it a small discoloration or a subtle change in leaf structure, even the slightest irregularity in the tea leaves' appearance might signify an underlying ailment. The condition of tea leaves can reveal much about the well-being of the plant, including potential issues related to pests, fungi, or nutritional deficiencies. Farmers closely examine tea leaves to aid in the identification of potential ailments.

Typically, a healthy tea plant displays vibrant green leaves that are smooth and consistently colored. Any deviation affecting the growth and appearance of tea leaves could indicate a plant anomaly. The condition of a tea plant's leaves can be indicative of its overall health status.

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

To tackle the aforementioned 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 scrutinizes the images and determines whether the tea plant is afflicted with a disease and identifies its type.

## Technical Architecture:



## Prerequisites:

To successfully execute this project, the following software, concepts, and packages are essential:

Anaconda Navigator: A free and open-source distribution of the Python and R programming languages, used for data science and machine learning applications. It is compatible with Windows, Linux, and macOS. Anaconda includes tools such as JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, RStudio, and Visual Studio Code. For this project, Google Colab and Spyder will be utilized.

Necessary Python Packages:

NumPy: An open-source numerical Python library that supports multidimensional arrays and matrix data structures, facilitating various mathematical operations.

Scikit-learn: A Python-based, free machine learning library that incorporates algorithms such as support vector machines, random forests, and k-nearest neighbours. It is compatible with Python numerical and scientific libraries like NumPy and SciPy.

Flask: A web framework employed for constructing web applications.

Python Packages Installation: Open the Anaconda prompt as an administrator and execute the following commands:

"pip install numpy"

"pip install pandas"

"pip install scikit-learn"

"pip install tensorflow==2.3.2"

"pip install keras==2.3.1"

"pip install Flask"

Deep Learning Concepts:

VGG16: Utilized as a transfer learning method, VGG16 constitutes a pre-trained model trained on 1000 classes of images, serving as a fundamental for image analysis.

Flask: A widely-used Python web framework, functioning as a third-party Python library for the development of web applications.

Project Flow:

User Interaction: The user engages with the UI (User Interface) to select the image.

Image Analysis: The chosen image is analysed by the integrated model within the Flask application.

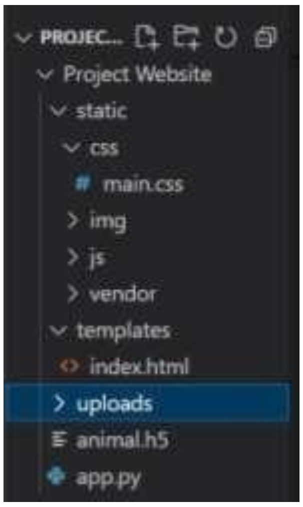
VGG16 Model Analysis: The VGG16 Model scrutinizes the image, and the predictions are displayed on the Flask UI.
To accomplish this, the activities and tasks outlined below need to be completed:

1. Data Collection
2. Importing Model Building Libraries
3. Loading the Model
4. Addition of Flatten Layers
5. Addition of Output Layer
6. Creation of a Model Object
7. Configuration of the Learning Process
8. Importation of the Image Data Generator Library
9. Configuration of the Image Data Generator Class
10. Application of Image Data Generator functionality to Trainset and Test set
11. Model Training
12. Model Saving
13. Model Testing
14. HTML File Creation
15. Python Code Building
16. Application Execution
17. Final Output Display

## Project Structure:

Organize the project in the following structure within a folder named "Project Website":

- Flask Application Files: Within the "Project Website" folder, the following files are essential for the Flask application:
- A folder named "templates" that houses the following HTML pages: index.html
- A Python script named "app.py" responsible for server-side scripting.
- Model File: Store the saved model, "Vgg-16-nail-disease.h5," within the "Project Website" folder.
- Within the "Project Website" folder, the following additional subfolders are required:
- A folder named "static" which includes the following subfolders:
    1. A "css" folder containing the "main.css" file.
    2. An "img" folder.
    3. A "js" folder.
    4. A "vendor" folder.



```
PROJEC...
  Project Website
    static
      css
        main.css
      img
      js
      vendor
    templates
      index.html
    uploads
    animal.h5
    app.py
```

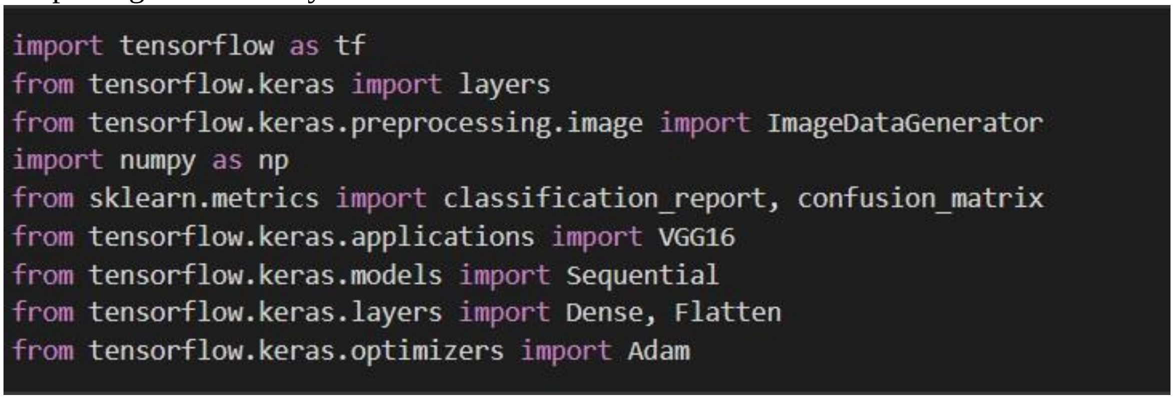
## Task – 1 : Data Collection

The following dataset has been used.

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

## Task – 2 : Importing Model Building Libraries

Importing the necessary libraries.

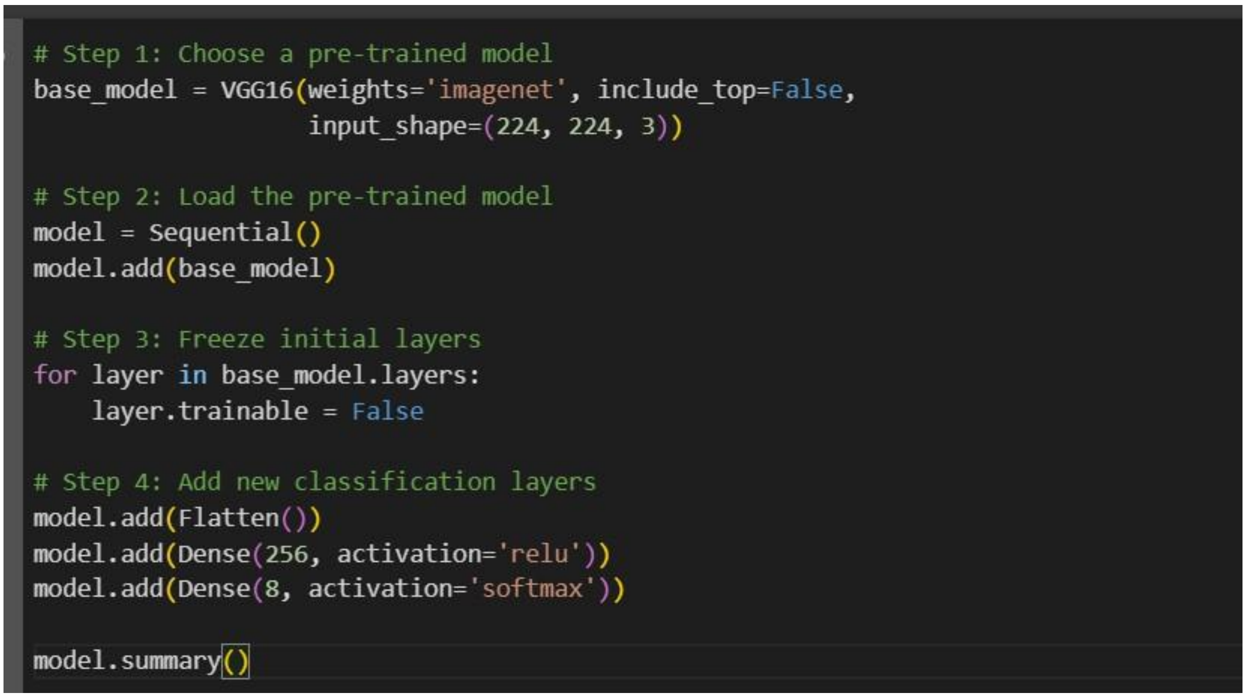


```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import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

## Task – 3 : Loading the Model &
## Task – 4 : Addition of Flatten Layers&
## Task – 5 : Addition of Output Layer

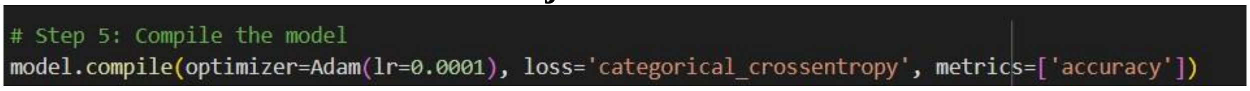
For VGG16 model, we need to keep the Hidden layer training as false, because it has trained weights

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



```
# 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()
```

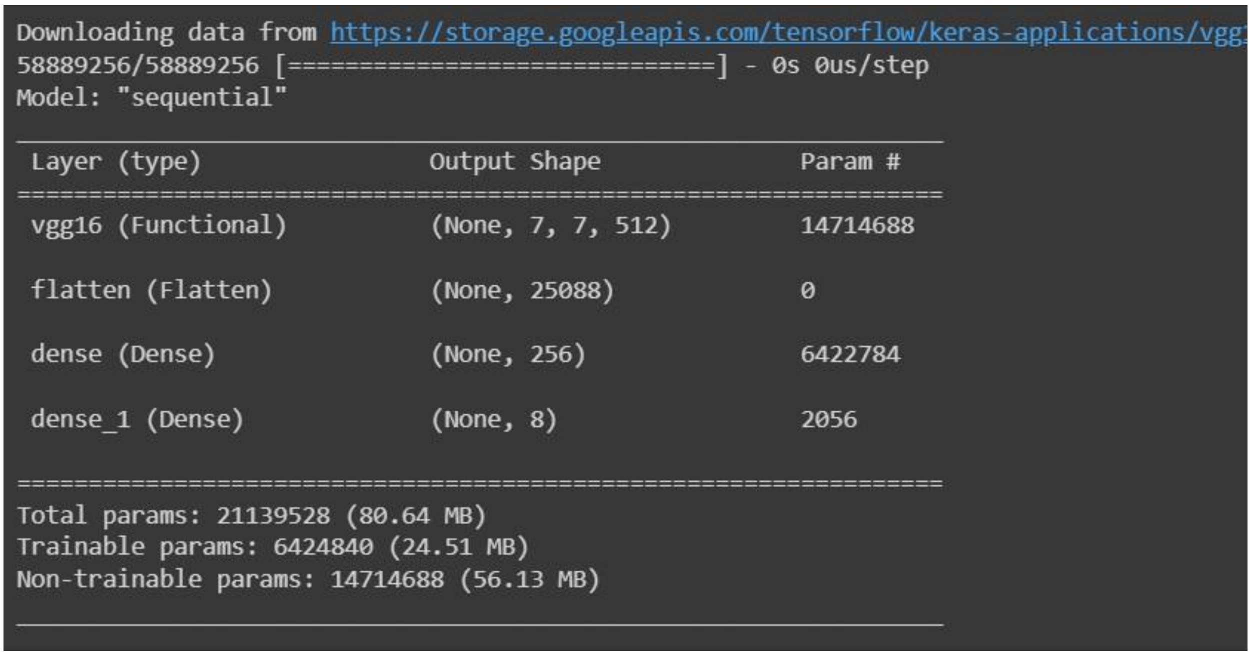
## Task – 6 : Creation of a Model Object



```
# 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 the model:



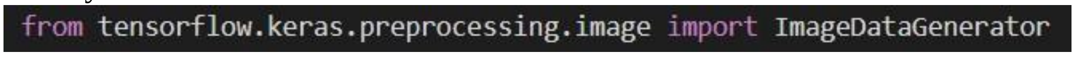
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 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)
_________________________________________________________________
```

## Task – 7 : Configuration of the Learning Process

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.

## Task – 8 : Importation of the Image Data Generator Library

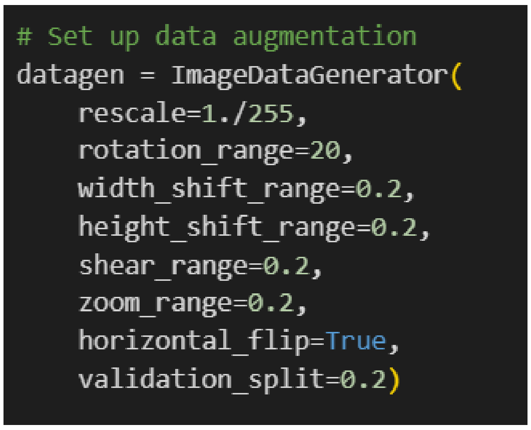
Within this phase, we will focus on enhancing the image data to mitigate undesirable distortions and accentuate critical image features essential for subsequent processing. This involves the implementation of various geometric transformations such as rotation, scaling, translation, and more.
Image data 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.



```python
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.



```python
# 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)
```

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

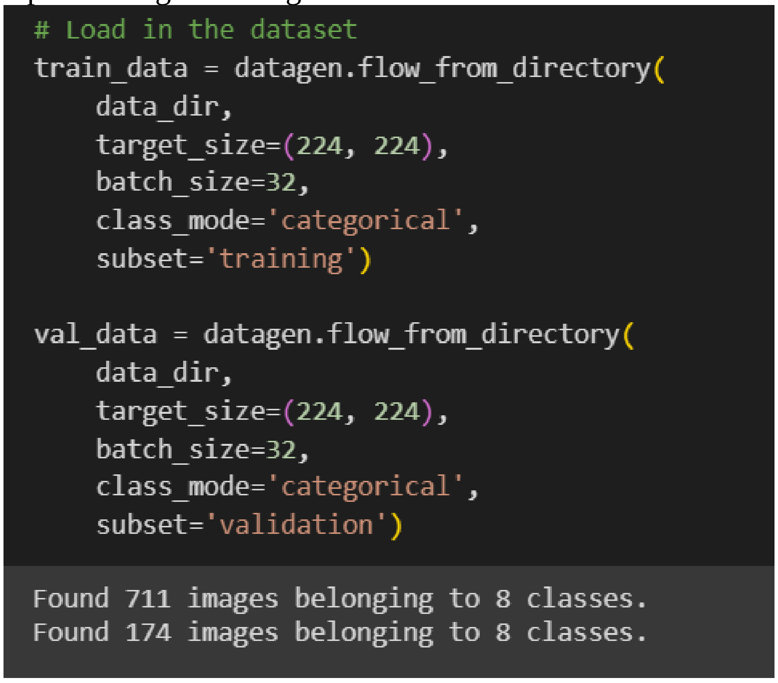
## Task – 10 : Application of Image Data Generator functionality to Trainset and Test set

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.

Loading our data and performing Data Augmentation



```
# 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.
```

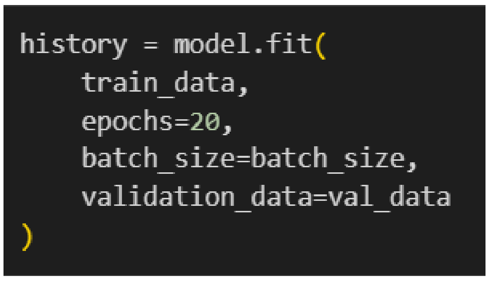
## Task – 11 : Model Training

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

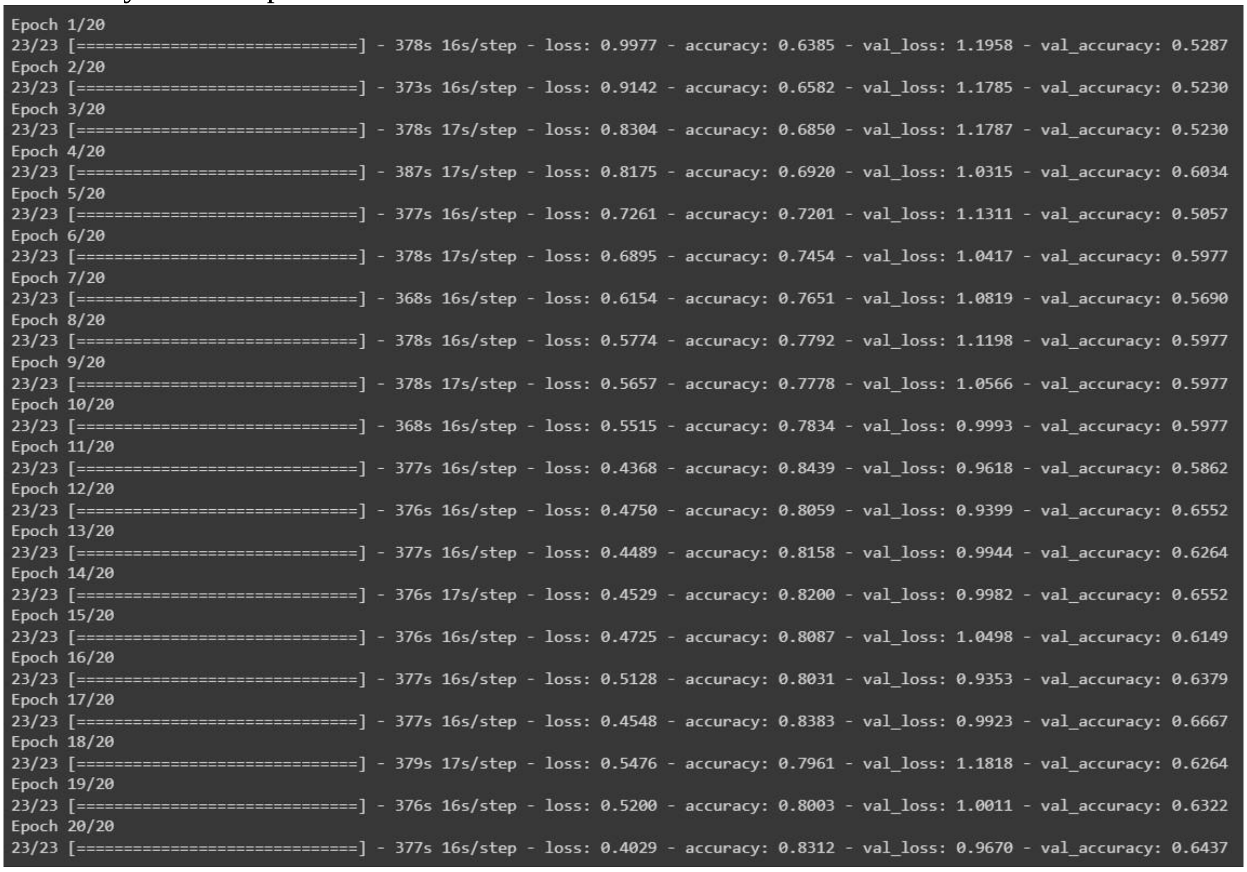
steps_per_epoch can be calculated by dividing the total number of samples in thedataset by the batch size.

- 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.



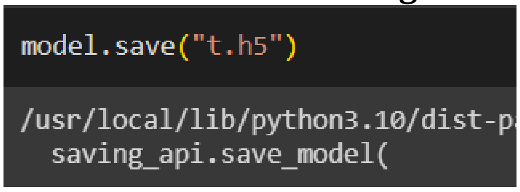
```
history = model.fit(
    train_data,
    epochs=20,
    batch_size=batch_size,
    validation_data=val_data
)
```

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
```

## Task – 12 : Model Saving



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

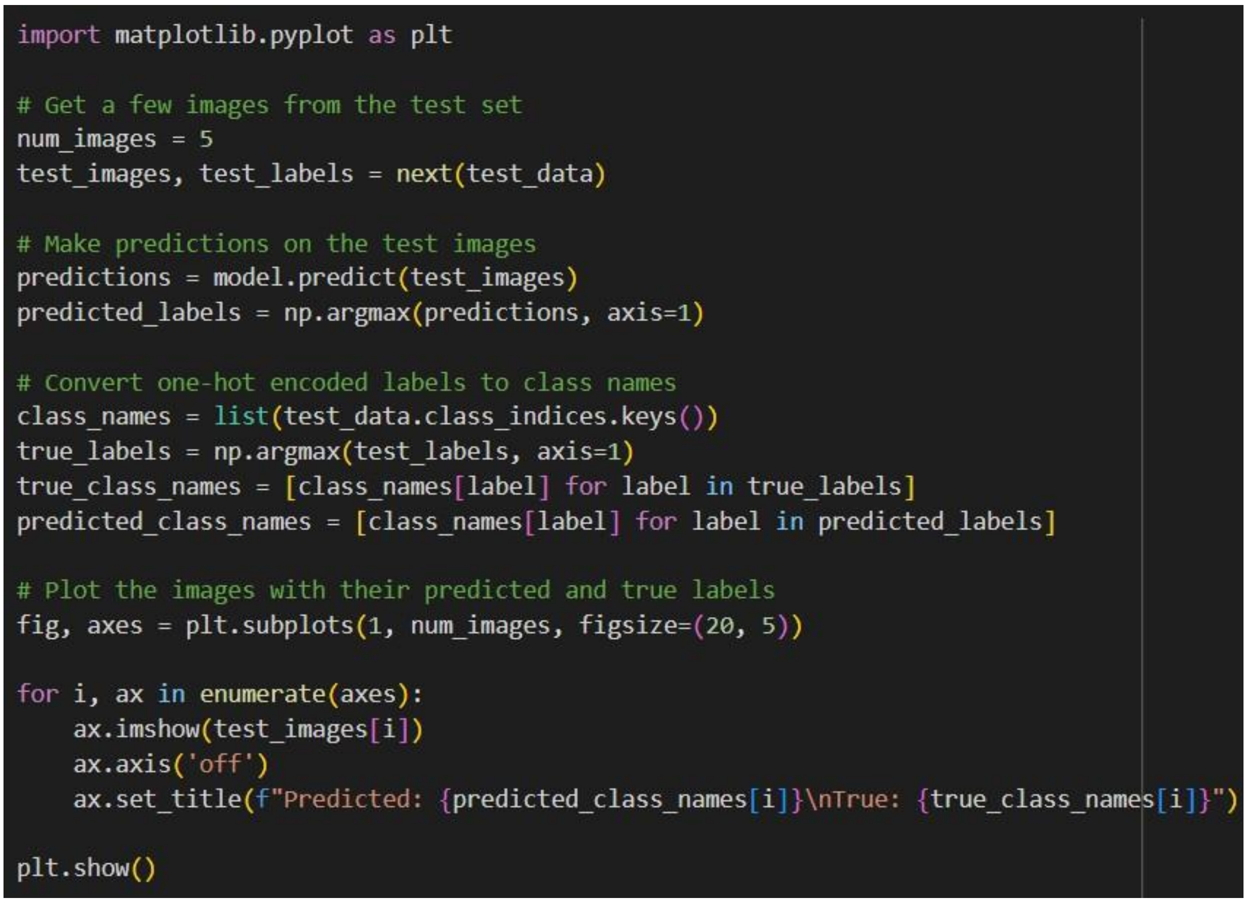
```
/usr/local/lib/python3.10/dist-p
  saving_api.save_model(
```

The model is saved with .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

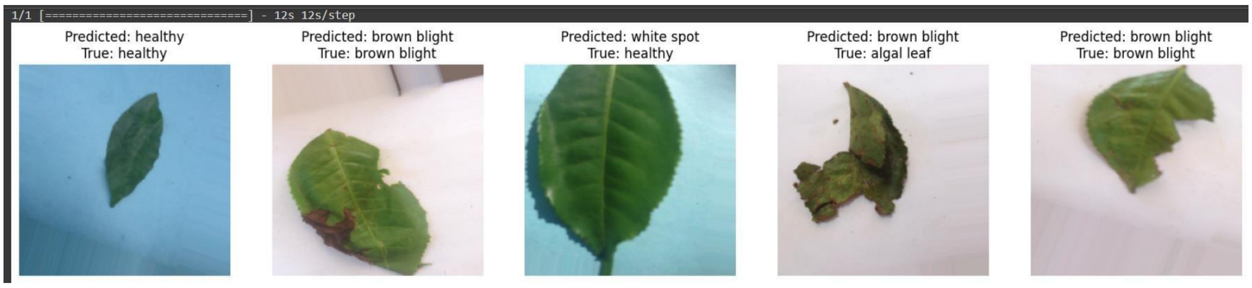
## Task – 13 : Model Testing

Taking an image as input and checking the results



```python
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:

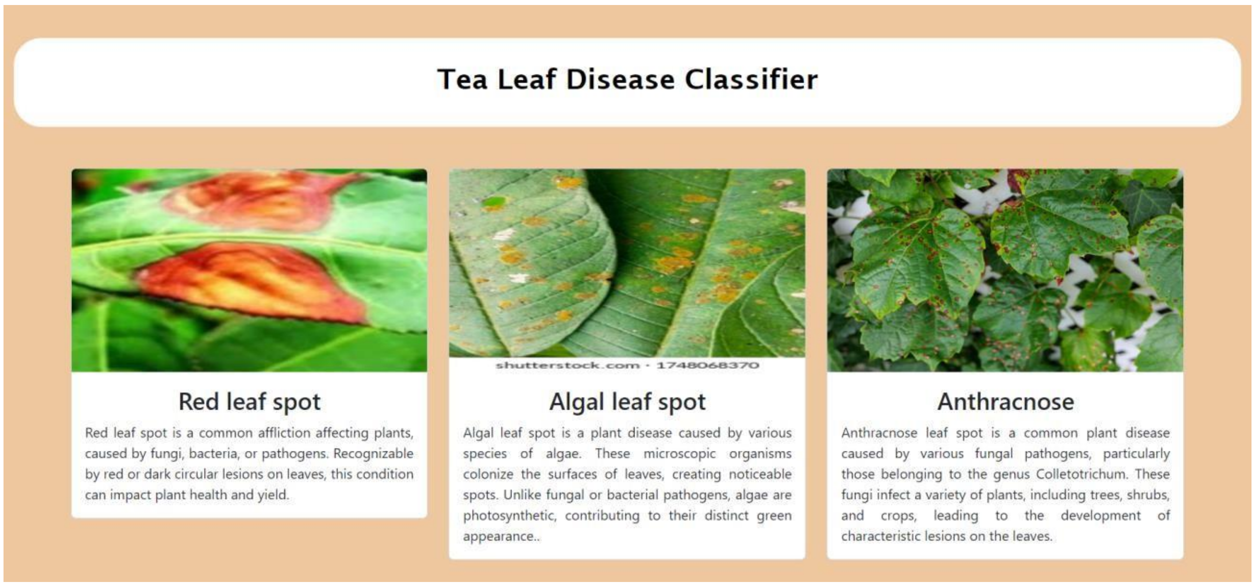


```
1/1 [==============================] - 12s 12s/step
```

## Task – 14 : 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



Tea Leaf Disease Classifier

Red leaf spot

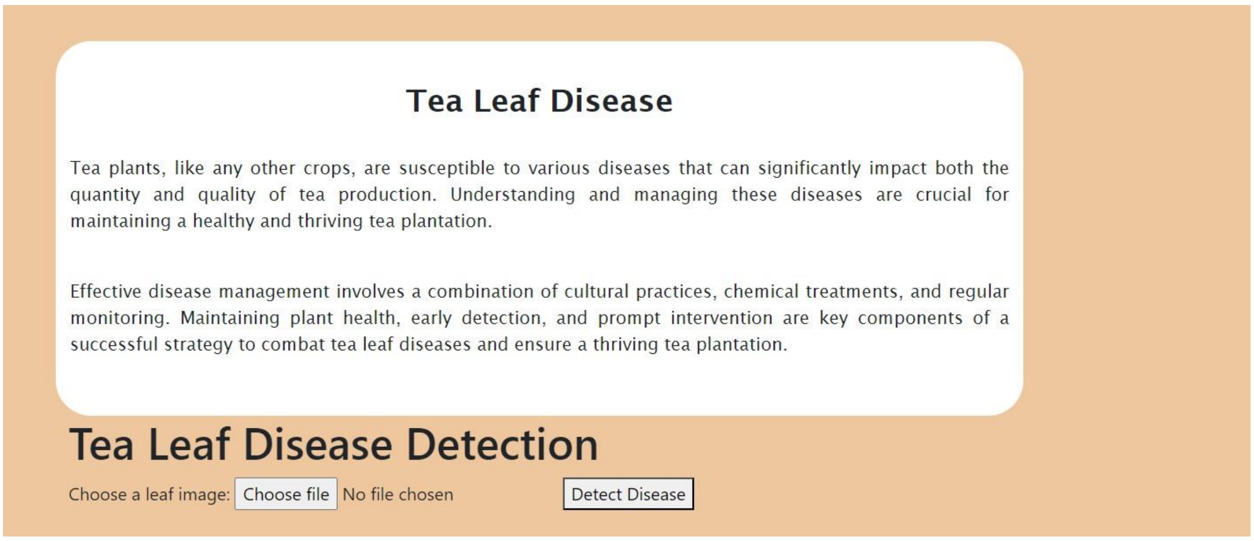
Red leaf spot is a common affliction affecting plants, caused by fungi, bacteria, or pathogens. Recognizable by red or dark circular lesions on leaves, this condition can impact plant health and yield.

Algal leaf spot

Algal leaf spot is a plant disease caused by various species of algae. These microscopic organisms colonize the surfaces of leaves, creating noticeable spots. Unlike fungal or bacterial pathogens, algae are photosynthetic, contributing to their distinct green appearance..

Anthracnose

Anthracnose leaf spot is a common plant disease caused by various fungal pathogens, particularly those belonging to the genus Colletotrichum. These fungi infect a variety of plants, including trees, shrubs, and crops, leading to the development of characteristic lesions on the leaves.



Tea Leaf Disease

Tea plants, like any other crops, are susceptible to various diseases that can significantly impact both the quantity and quality of tea production. Understanding and managing these diseases are crucial for maintaining a healthy and thriving tea plantation.

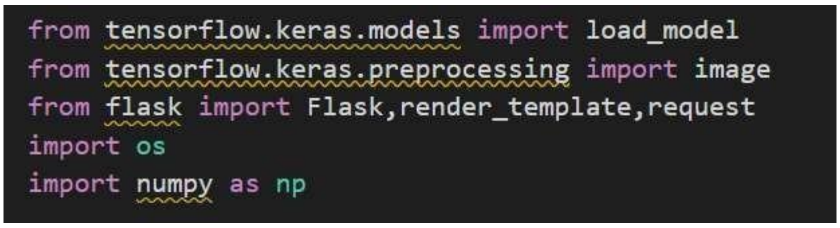
Effective disease management involves a combination of cultural practices, chemical treatments, and regular monitoring. Maintaining plant health, early detection, and prompt intervention are key components of a successful strategy to combat tea leaf diseases and ensure a thriving tea plantation.

Tea Leaf Disease Detection

index.html file using html

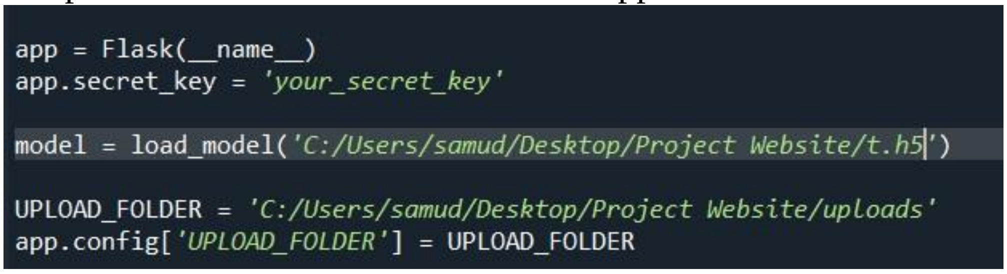
**Task – 15 : Python Code Building**

Step 1: Import libraries



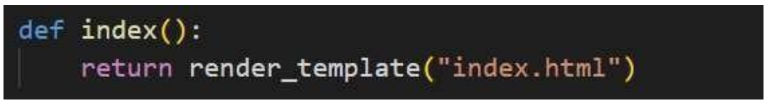
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask,render_template,request
import os
import numpy as np
```

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

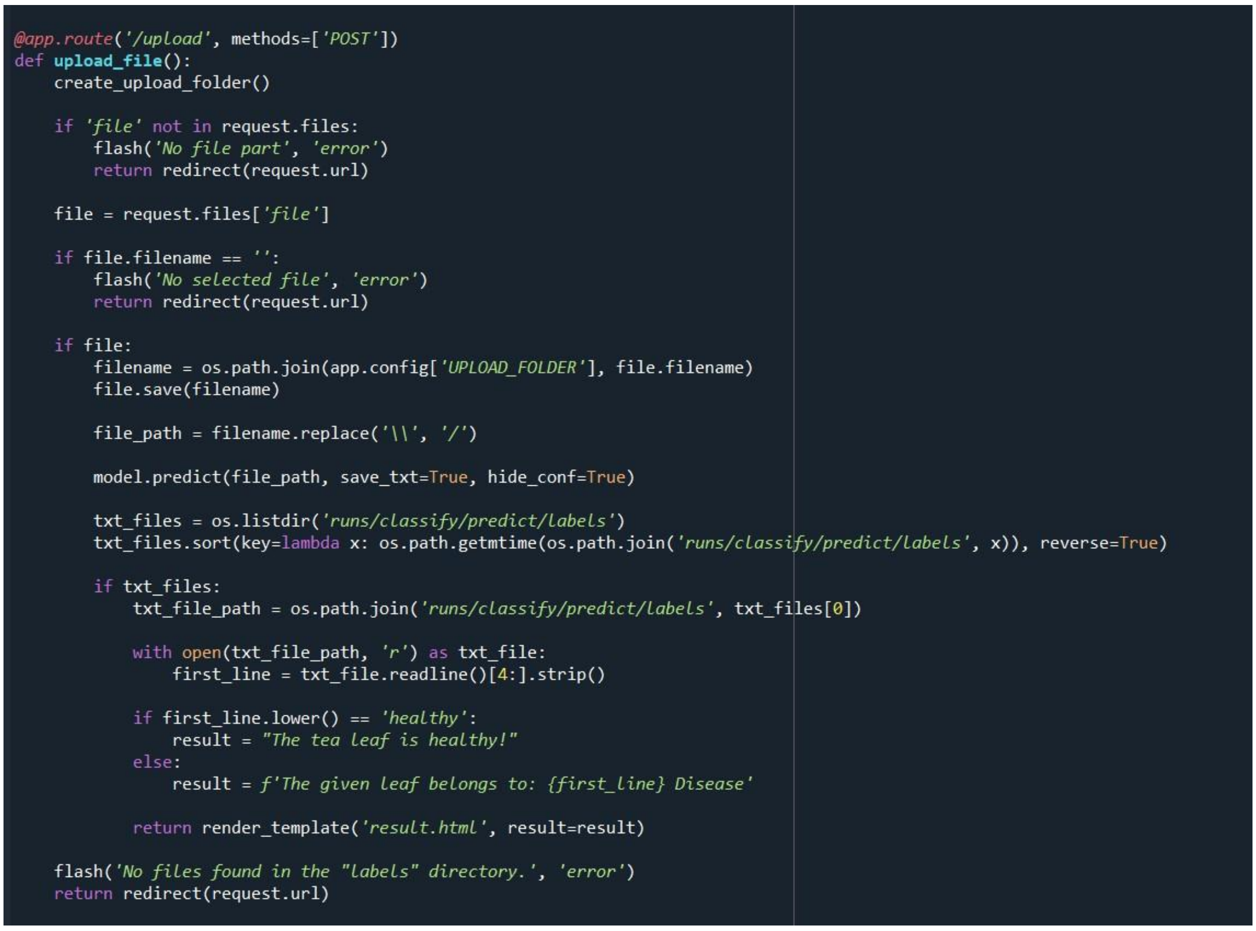


```
def index():
    return render_template("index.html")
```

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.

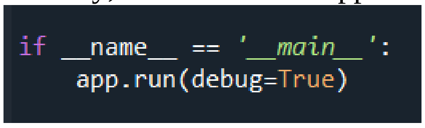
The outcome is a numerical value representing a specific class (e.g., 0, 1, 2, etc.), which resides at the 0th index of the variable "preds." This numerical value is assigned to the declared index variable. The corresponding class name is then derived and assigned to the "predict" variable, which is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

## Task – 16 : Application Execution

Finally, we will run the application.



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

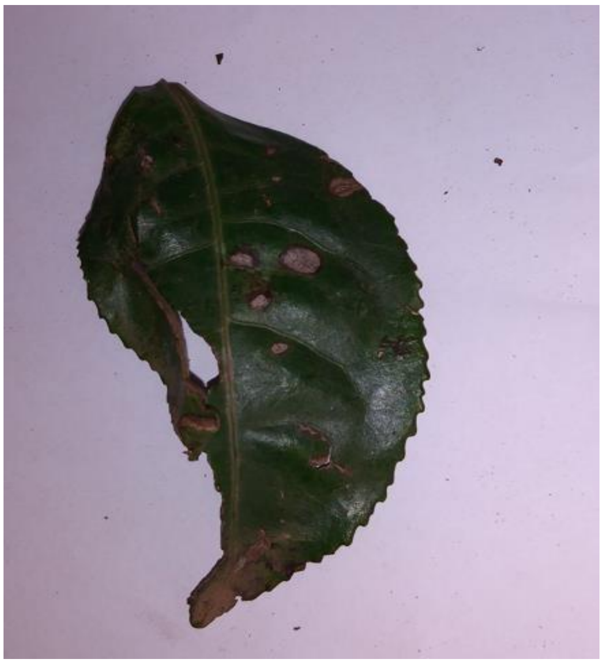
Follow the steps outlined below to execute your Flask application:

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

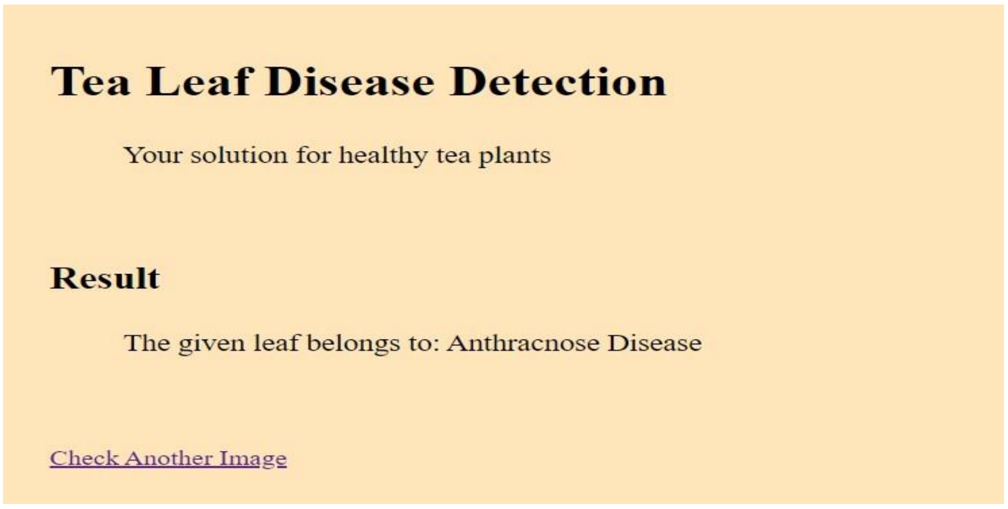
**Task – 17 : Final Output Display**

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

Thus, the project is ended.

******THANK YOU******