Tea leaf disease detection

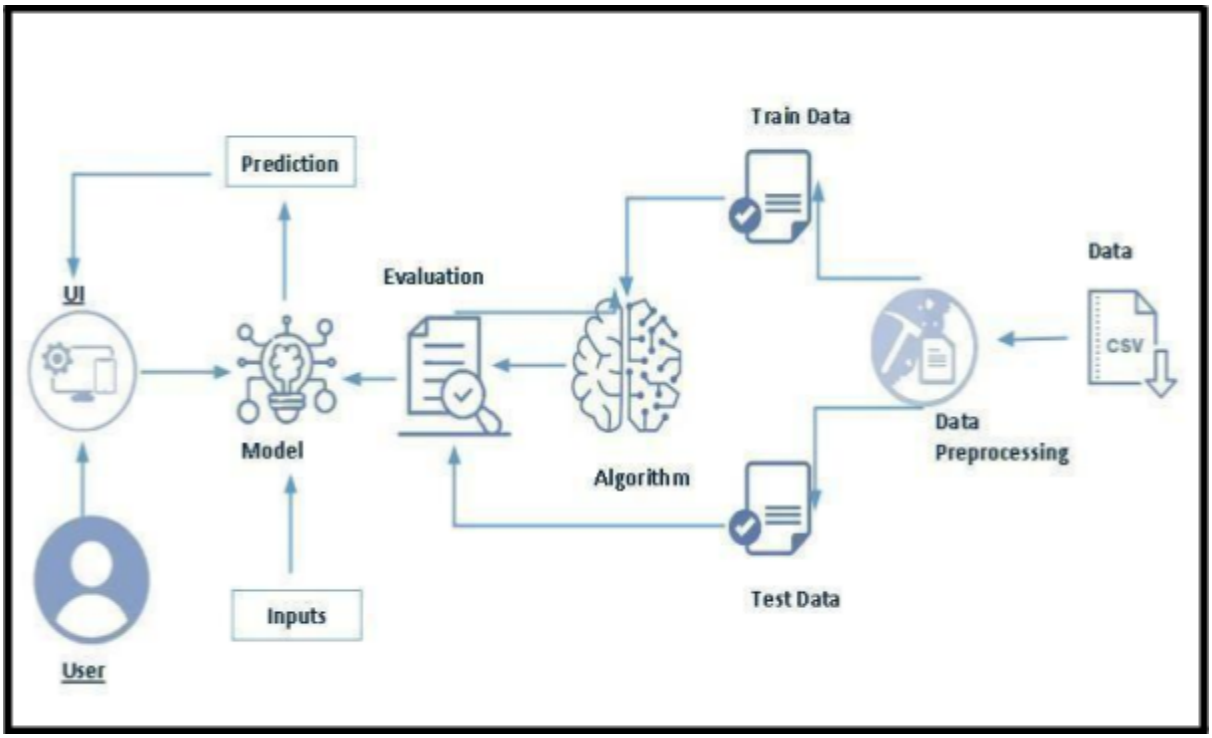
Project Description:

The goal of this project is to develop a machine learning model that can help tea farmers identify diseases in tea leaves. By analyzing the images of tea leaf disease, we aim to create a model that can help farmers identify the tea leaf diseases by clicking a picture which will work as input.

The dataset used in this project consists of different types of tea leaf disease images.

The main objective of this project is to build a robust machine learning model that can accurately classify tea leaf diseases.

Technical Architecture:



Pre requisites:

To complete this project, you must required following software's, concepts and packages

- **Google collab**
- **Python packages:**

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - NN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
    - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for Convolution Neural Networks for Image Classification.
- Hosting ML Model on website/web apps/
- Gain a broad understanding about data handling.
- Have knowledge on pre-processing the data/transformation techniques and some neural networks knowledge

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated in the Flask backend.
- Once the model analyses the input the prediction is showcased on the UI.

To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect the dataset or create the dataset
  - Splitting data into train and test

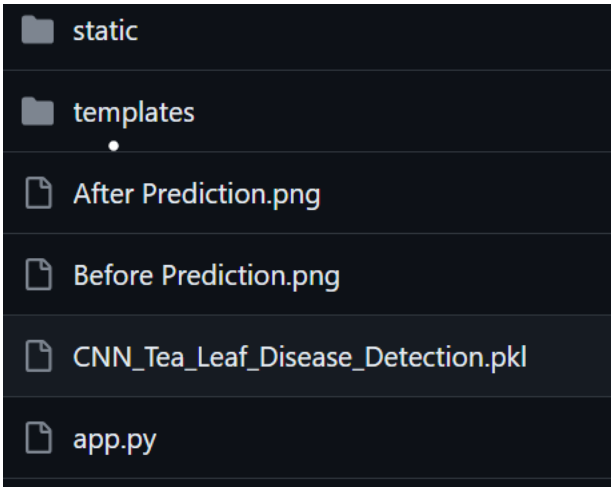
- Model building
  - Import the model building libraries
  - Initializing the model
- Training and testing the model
- Evaluating performance of model
  - Save the model
- Application Building

- Create an HTML Landing page

- Build Flask application for backend hosting of ML Model

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for backend.
- CNN_Tea_Leaf_Disease_Detection.pkl is our saved model. Further we will use this model for flask integration.
- Static folder contains CSS and JS files with Images for UI.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specifying the business problem

The problem statement for the tea leaf disease detection machine learning project is to develop a predictive model that can accurately classify the tea leaf disease based on the training data. The aim is to create a model that can identify diseases in tea leaves accurately and quickly.
The ultimate goal of this project is to develop a machine learning model that can assist tea farmers and tea corporations to accurately identify tea leaf diseases quickly without risking losing the crop by spraying unnecessary pesticide or letting the disease spread.

Activity 2: Business requirements

Here are some potential business requirements for Share price predictor.

a. Accurate classification: The ml model should be able to accurately predict the kind of disease the leaf in the picture has.

b. Real-time data acquisition: The model must be able to acquire real-time data from tea farmers. The data acquisition must be seamless and efficient to ensure that the model is able to quickly predict the disease.

c. User-friendly interface: The predictor must have a user-friendly interface that is easy to navigate and understand. The interface should present the results of the predictor in a clear and concise manner to enable farmers to make informed decisions.

d. Report generation: Generate a report to identify the disease for the farmer so he will know which steps to take.

Activity 3: Literature Survey

Tea is a globally popular beverage, making its cultivation a significant agricultural endeavor. However, tea plants are susceptible to various diseases that can compromise crop quality and yield. The conventional method of disease detection through visual inspection by experts is time-consuming and subjective. In response, the application of deep learning models, a subset of artificial intelligence, has emerged as a promising solution for automating disease detection in tea leaves.

- Deep learning models, particularly Convolutional Neural Networks (CNNs), have demonstrated notable success in image classification tasks. This success has spurred interest in their application in agriculture, including tea leaf disease detection.
- High-quality datasets are pivotal for training accurate and reliable deep learning models. Several publicly available datasets for tea leaf disease detection have been developed, providing researchers with valuable resources for benchmarking their models.
- Preprocessing techniques like image augmentation and normalization play a crucial role in enhancing the performance of deep learning models. These techniques help standardize and augment the dataset, enabling the model to learn robust features.

Several studies have made significant strides in utilizing deep learning for tea leaf disease detection. For instance, "Tea Disease Detection using Deep Learning" introduced a CNN-based approach that achieved impressive accuracy rates in detecting common tea leaf diseases. Another notable study, "An Ensemble of Deep Learning Models for Tea Disease Classification," proposed an ensemble approach that combined multiple deep learning models, leading to improved accuracy and robustness in disease identification.

Despite the promising results, challenges remain. Large, diverse datasets are essential for training accurate models, and efforts are needed to address issues related to data collection and labeling. Additionally, ensuring model generalization across different tea varieties and environmental conditions is crucial for practical implementation. As researchers continue to address these challenges, the integration of deep learning models into tea cultivation practices holds the potential to revolutionize disease detection, offering a more efficient and reliable solution compared to traditional methods.

Activity 4: Social or Business Impact.

The application of deep learning models for tea leaf disease detection has the potential to have both significant social and business impacts.

Social Impact:

1. Improved Crop Yield and Quality: Early detection of diseases through automated systems can lead to timely intervention, preventing the spread of diseases and reducing crop losses. This, in turn, ensures a more reliable and abundant supply of tea for consumers, contributing to food security.
2. Sustainable Agriculture Practices: By minimizing the need for chemical treatments and pesticides, which can have adverse environmental effects, automated disease detection promotes more sustainable and eco-friendly agricultural practices. This benefits the environment and local communities by reducing chemical runoff and contamination.
3. Empowering Small-scale Farmers: Small-scale tea farmers often lack access to advanced technologies and resources. Implementing user-friendly deep learning models for disease detection can level the playing field, providing these farmers with tools to enhance their crop management practices and economic viability.

Business Impact:

1. Cost Savings: Early detection and intervention can lead to significant cost savings for tea producers. By identifying diseases in their early stages, producers can minimize the extent of crop losses and reduce the need for expensive treatments or replanting.
2. Enhanced Product Quality and Reputation: Consistently producing high-quality tea is crucial for maintaining a positive brand reputation. Automated disease detection ensures that only healthy leaves are harvested, leading to a higher-quality end product that meets or exceeds consumer expectations.
3. Market Competitiveness: Implementing advanced technologies like deep learning models can give tea producers a competitive edge in the market. Companies that invest in innovative solutions for

disease detection demonstrate a commitment to quality and innovation, which can attract consumers and partners alike.

4. Research and Development Opportunities: The development and application of deep learning models for tea leaf disease detection can open up new avenues for research and innovation in the tea industry. This can lead to the creation of intellectual property, collaborations with research institutions, and potential diversification into related technologies or services.

In summary, the integration of deep learning models for tea leaf disease detection has the potential to bring about positive social impacts by promoting sustainable agricultural practices and empowering small-scale farmers. On the business front, it can lead to cost savings, improved product quality, enhanced market competitiveness, and the creation of new opportunities for research and development within the tea industry.

Milestone 2: Data Collection and Visualizing and analyzing the data

ML depends heavily on data, It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Download the dataset

There are many popular open sources for collecting the data.

E.g. kaggle.com, UCI repository, etc.

In this project, we have used **Identifying Disease in Tea leaves**

data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

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

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are n number of methods for understanding the data. But here we have used some of it.

Activity 2: Importing the libraries

Import the necessary libraries as shown in the image.

Importing Libraries

```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import os
     import PIL

     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers
     from tensorflow.keras.models import Sequential
     from sklearn.metrics import classification_report
     import pathlib
```

Activity 3: Importing the dataset from Kagel

Our dataset format might be in .csv, excel files, .txt, .json, etc. We have directly imported the data from Kaggle using Kaggle api token.

```
!pip install -q kaggle
```

```
[ ] !mkdir ~/.kaggle
```

```
[ ] !cp kaggle.json ~/.kaggle
```

Importing Kaggle Dataset

```
!kaggle datasets download -d shashwatwork/identifying-disease-in-tea-leafs
```

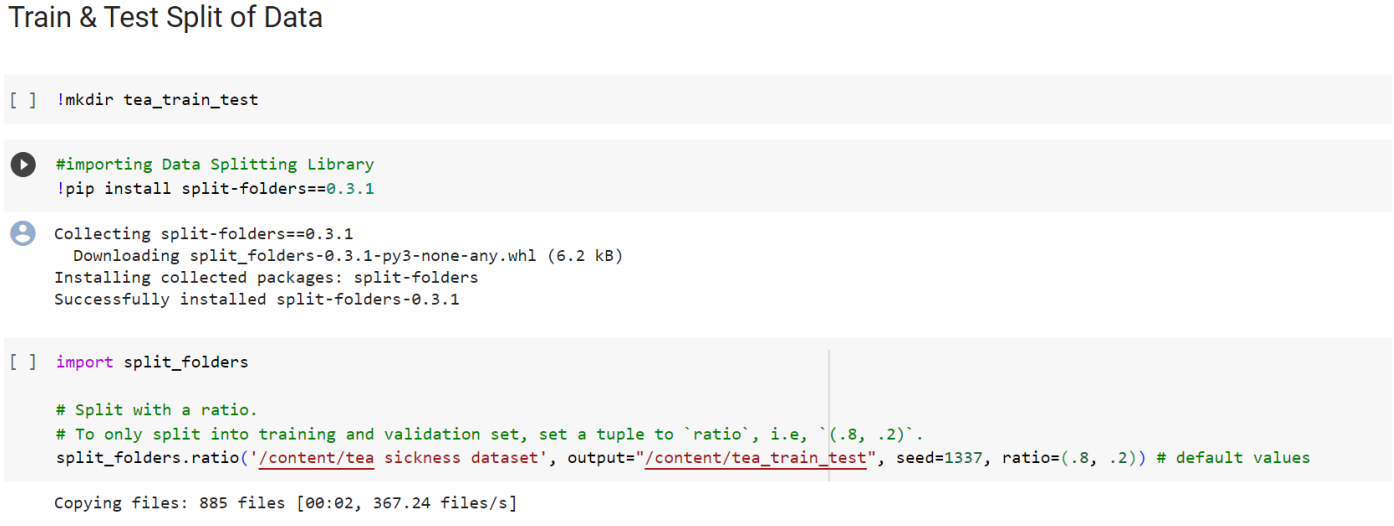
```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading identifying-disease-in-tea-leafs.zip to /content
 96% 713M/740M [00:04<00:00, 150MB/s]
100% 740M/740M [00:04<00:00, 158MB/s]
```

Milestone 3: Data Pre-processing

We have unzipped the data.

Activity 4: Splitting data into train and test

Now let's split the Dataset into train and test sets. First create a directory tea_train_test, and then we used imported a data splitting library, which we used to split the data into test, training.



```
dataset_path = "/content/tea sickness dataset"
dataset_dir = pathlib.Path(dataset_path)
```

```
# list of sub directory(class)

class_names = []
for root, dirs, files in os.walk(dataset_path):
    if len(root) > len(dataset_path):
        x_class = os.path.relpath(root, dataset_path)
        class_names.append(x_class)

print(class_names)
```

```
['red leaf spot', 'algal leaf', 'gray light', 'healthy', 'brown blight', 'Anthracnose', 'bird eye spot', 'white spot']
```

```
# print total number of images in the dataset
for class_i in class_names:
    image_count = len(list(dataset_dir.glob(f'{class_i}/*.jpg')))
    print(f"Images in class {class_i}:",image_count)
```

```
# Parameter setting
train_batch = 128
val_batch = 128
img_height = 224
img_width = 224
IMG_SIZE = (img_height, img_width)
val_split = 0.2

# Load the training dataset
train_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_path,
    validation_split=val_split,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=train_batch
)
```

```
Found 885 files belonging to 8 classes.
Using 708 files for training.
```

```
# Load data for validation
val_ds = tf.keras.utils.image_dataset_from_directory(dataset_dir,
                                                     validation_split=val_split,
                                                     subset="validation",
                                                     seed=123,
                                                     image_size=(img_height, img_width),
                                                     batch_size=val_batch
                                                    )
```

```
Found 885 files belonging to 8 classes.
Using 177 files for validation.
```

```
class_names = train_ds.class_names
print(class_names)

num_classes=len(class_names)
```

```
['Anthracnose', 'algal leaf', 'bird eye spot', 'brown blight', 'gray light', 'healthy', 'red leaf spot', 'white spot']
```

```
# Review dataset sample
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

Data Preprocessing

```
[ ]  AUTOTUNE = tf.data.AUTOTUNE

     train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
     val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1:

CNN was our best model that we got, other models were Resnet50,Vgg16.

CNN Model Building

```
[ ]  #Data augmentation layers to increase data variation for training
     data_augmentation = keras.Sequential([
         layers.RandomFlip("horizontal",input_shape=(img_height,img_width,3)),
         layers.RandomFlip("vertical"),
         layers.RandomRotation(0.2),
         layers.RandomZoom(0.2),
     ])
```

```
[ ]  # Model architecture
     cnn_model = Sequential([
         data_augmentation,
         layers.Rescaling(1./255),
         layers.Conv2D(16, 3, padding='same', activation='relu'),
         layers.MaxPooling2D(),
         layers.Conv2D(16, 3, padding='same', activation='relu'),
         layers.MaxPooling2D(),
         layers.Conv2D(32, 3, padding='same', activation='relu'),
         layers.MaxPooling2D(),
         layers.Flatten(),
```

```
         layers.Dense(32, activation='relu'),
         layers.Dense(32, activation='relu'),
         layers.Dropout(0.3),
         layers.BatchNormalization(),
         layers.Dense(32, activation='relu'),
         layers.Dropout(0.3),
         layers.BatchNormalization(),
         layers.Dense(num_classes)   #num_classes=len(class_names) i.e 8
     ])
```

```
]  # compile model
   base_learning_rate = 0.0005
   cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
                 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                 metrics=['accuracy'])
```

```
   # display model
   cnn_model.summary()
```

CNN Model Training

```
[ ]  epochs=300
     history = cnn_model.fit(
         train_ds,
         validation_data=val_ds,
         epochs=epochs,
         verbose=0
     )
```

```
     # Check training result
     acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']

     loss = history.history['loss']
     val_loss = history.history['val_loss']

     epochs_range = range(epochs)

     plt.figure(figsize=(12, 6))
     plt.subplot(1, 2, 1)
```

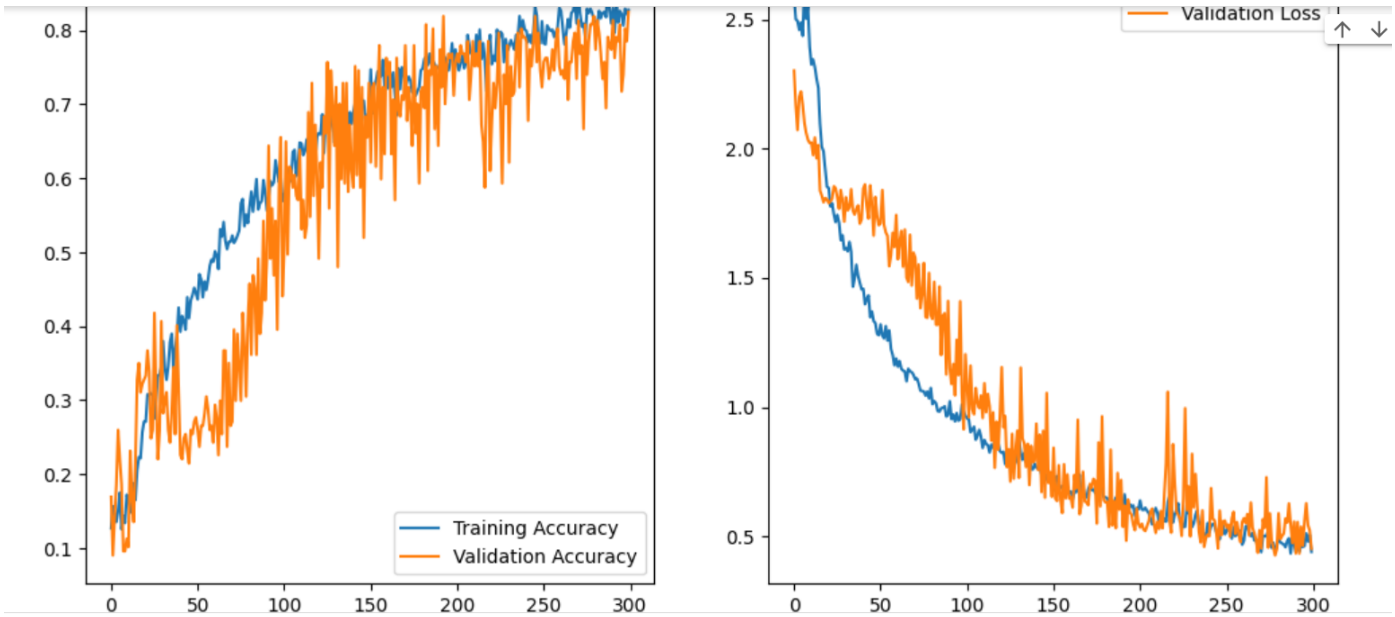
```
# Check training result
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Activity 3: Evaluating performance of the model and saving the model

Now let's see the performance of all the models and save the best model.

```
# Retrieve a batch of images from the test set
image_batch, label_batch = val_ds.as_numpy_iterator().next()
predictions = cnn_model.predict_on_batch(image_batch)

class_predictions = []
for i in predictions:
    class_prediction = np.argmax(i)
    class_predictions.append(class_prediction)

class_predictions = np.array(class_predictions)
print('Predictions:\n', class_predictions)
print('Labels:\n', label_batch)
print()
print(classification_report(label_batch, class_predictions))

plt.figure(figsize=(10, 21))
for i in range(18):
    ax = plt.subplot(6, 3, i + 1)
    plt.imshow(image_batch[i].astype("uint8"))
    plt.title("Prediction: "+class_names[class_predictions[i]]+"\nLabel: "+class_names[label_batch[i]])
    plt.axis("off")
```

Activity 4: Model Testing

Our model is performing well. So, we are saving the model by pickle.dump().

```
from tensorflow.keras.preprocessing import image

# testing 1
img = image.load_img('/content/tea sickness dataset/algal leaf/UNADJUSTEDNONRAW_thumb_19.jpg',target_size =(224,224))
```

```
[ ] img
```

```
[ ] x = image.img_to_array(img)
    x = np.expand_dims(x,axis = 0)
    pred =np.argmax(cnn_model.predict(x))
    op =['Anthracnose','algal leaf','bird eye spot','brown blight','gray light','healthy','red leaf spot','white spot'
    op[pred]
```

```
1/1 [==============================] - 0s 236ms/step
'algal leaf'
```

```
#testing-2
img2 = image.load_img('/content/tea_train_test/val/healthy/UNADJUSTEDNONRAW_thumb_21f.jpg',target_size =(224,224))
img2
```

```
[ ]  y = image.img_to_array(img2)
     y = np.expand_dims(y,axis = 0)
     pred =np.argmax(cnn_model.predict(y))
     op =['Anthracnose','algal leaf','bird eye spot','brown blight','gray light','healthy','red leaf spot','white spot']
     op[pred]
```

```
1/1 [==============================] - 0s 20ms/step
'healthy'
```

```
[ ]  #Testing-3
     img3 = image.load_img('/content/tea_train_test/val/gray light/IMG_20220503_140914.jpg',target_size =(224,224))
     img3
```

```
[ ]  x = image.img_to_array(img3)
     x = np.expand_dims(x,axis = 0)
     pred =np.argmax(cnn_model.predict(x))
     op =['Anthracnose','algal leaf','bird eye spot','brown blight','gray light','healthy','red leaf spot','white spot']
     op[pred]
```

```
1/1 [==============================] - 0s 18ms/step
'Anthracnose'
```

Saving Model Histroy

```
history_df = pd.DataFrame(history.history)
history_df.to_csv("cnn_history.csv")
history_df.tail()
```

	loss	accuracy	val_loss	val_accuracy
295	0.367669	0.865819	0.611099	0.774011
296	0.375968	0.850282	0.609117	0.790960
297	0.385573	0.851695	0.777662	0.745763
298	0.386529	0.847458	0.512042	0.824859
299	0.365979	0.878531	0.614045	0.796610

```
cnn_model.save('CNN_Tea_Leaf_Disease_Detection.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `mo
  saving_api.save_model(
```

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script

Activity1: Building Html Pages:

For this project we created one html page, namely index.html as well as a css styles sheet and javascript document.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Falling Leaves Animation</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
        }

    table {
        margin: 0 auto;
        }

        #grey-rectangle {
            width: 1000px;
            height: 600px;
            background-color: rgba(204, 204, 204, 0.5);
            position: absolute;
            top: 50%;
            left: 50%;
            transform: translate(-50%, -50%);
            border-radius: 10px;
            padding: 20px;
        }
```

```css
    #writing-area {
        background-color: rgba(255, 255, 255, 0.7);
        padding: 10px;
        border-radius: 10px;
        text-align: center;
    }

    #upload-container {
        width: 10px;
        height: 50px;
        background-color: rgba(204, 204, 204, 0);
        position: absolute;
        top: 85%;
        left: 5%;
        transform: translate(-50%, -50%);
        border-radius: 10px;
        display: flex;
        align-items: center;
    }

    #upload-box {
        margin-right: 10px;
    }

    input[type="file"] {
        margin-right: 10px;
    }
```

```
        .prediction {
            margin-top: 10px;
        }
    </style>
</head>
<body>
    <div id="grey-rectangle">
        <div id="writing-area">
            <!-- Add your content here --><h3><u>Convolutional Neural Network Based Tea Leaf Disease Detection</u></h3>
            <div><br>This Web App helps the user identify the condition of their Tea Plants.
            <br>The following 8 Leaf Health conditions can be classified using the CNN Model with an accuracy of <b>82%</b>.
            <br><br>
            <table>
    <tr>
        <td style="text-align: center;">
            <img src="{{ url_for('static', filename='Healthy.jpg') }}" alt="Healthy Leaf" width="100" height="100"><br>
            Healthy Leaf
        </td>
        <td style="text-align: center;">
            <img src="{{ url_for('static', filename='Algal.jpg') }}" alt="Algal Leaf Disease" width="100" height="100"><br>
            Algal Leaf Disease
        </td>
        <td style="text-align: center;">
            <img src="{{ url_for('static', filename='Anthracnose.jpg') }}" alt="Anthracnose" width="100" height="100"><br>
            Anthracnose
        </td>
        <td style="text-align: center;">
            <img src="{{ url_for('static', filename='Bird Eye Spot.jpg') }}" alt="Bird Eye Spot" width="100" height="100"><br>
            Bird Eye Spot
```

```
        </td>
    </tr>
    <tr>
        <td style="text-align: center;">
            <img src="{{ url_for('static', filename='Gray Blight.jpg') }}" alt="Gray Blight" width="100" height="100"><br>
            Gray Blight
        </td>
        <td rowspan="2" style="text-align: center;">
            <img src="{{ url_for('static', filename='Red Leaf Spot.jpg') }}" alt="Red Leaf Spot" width="100" height="100"><br>
            Red Leaf Spot
        </td>
        <td>
            <img src="{{ url_for('static', filename='White Spot.jpg') }}" alt="White Spot" width="100" height="100"><br>
            White Spot
        </td>
        <td style="text-align: center;">
            <img src="{{ url_for('static', filename='Brown Blight.jpg') }}" alt="Brown Blight" width="100" height="100"><br>
            Brown Blight
        </td>
    </tr>

</table>
</div>
        </div>
        <div id="upload-container">
            <div id="upload-box">
                <form action="/predict" method="post" enctype="multipart/form-data">
                    <input type="file" name="image" id="fileInput" accept=".jpg, .jpeg, .png">
                    <input type="submit" value="Predict">
```

```
                </form>
            </div>
            <div class="prediction">Predicted class: {{ prediction }}</div>
        </div>
    </div>

    <div id="canvas-container">
        <canvas id="canvas"></canvas>
    </div>

    <script src="{{ url_for('static', filename='script.js') }}"></script>
</body>
</html>
```

Css style sheet

```
body, html {
    margin: 0;
    padding: 0;
    height: 100%;
    overflow: hidden; /* Add this line to prevent scrolling */

}

#grey-rectangle {
    width: 300px;
    height: 200px;
    background-color: #ccc;
    position: absolute;
    top: 20px;
    left: 20px;
    border-radius: 10px;
}



#canvas-container {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    z-index: -1; /* Ensure canvas is behind other elements */
}

```

```css
canvas {
    display: block;
}


label {
    cursor: pointer;
}
```

Javascript

```javascript
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');

let leaves = [];

canvas.width = window.innerWidth;
canvas.height = window.innerHeight;

class Leaf {
    constructor() {
        this.image = new Image();
        this.image.src = 'static/leaf.png'; // Make sure you have an image named 'leaf.png' in the same directory
        this.x = Math.random() * canvas.width;
        this.y = -50;
        this.speed = Math.random() * 2 + 1;
    }

    fall() {
        this.y += this.speed;
        if (this.y > canvas.height) {
            this.y = -50;
            this.x = Math.random() * canvas.width;
        }
    }

    draw() {
        ctx.drawImage(this.image, this.x, this.y, 200, 150); // Adjust size as needed
    }
}
```

```
function animate() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    leaves.forEach(leaf => {
        leaf.fall();
        leaf.draw();
    });

    requestAnimationFrame(animate);
}

for (let i = 0; i < 20; i++) {
    leaves.push(new Leaf());
}

animate();
```

Activity 2: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request
import pickle

import numpy as np
import pandas as pd
import PIL
```

Load the saved model. Importing the flask module in the project is mandatory.

```
app = Flask(__name__, static_folder='static', static_url_path='/static')

model = pickle.load(open('CNN_Tea_Leaf_Disease_Detection.pkl','rb'))
```

Render HTML page:

```python
@app.route('/')
def index():
    return render_template('index.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you upload the images from the html page the values can be retrieved using POST Method.

Collect Image Data:

```
</table>
</div>
        </div>
        <div id="upload-container">
            <div id="upload-box">
                <form action="/predict" method="post" enctype="multipart/form-data">
                    <input type="file" name="image" id="fileInput" accept=".jpg, .jpeg, .png">
                    <input type="submit" value="Predict">
                </form>
            </div>
            <div class="prediction">Predicted class: {{ prediction }}</div>
        </div>
    </div>

    <div id="canvas-container">
        <canvas id="canvas"></canvas>
    </div>
```

Retrieves the value from UI:

```
@app.route('/predict', methods=['POST'])
def upload():
    uploaded_file = request.files['image']
    if uploaded_file.filename != '':
        uploaded_file.save(uploaded_file.filename)  # Save the uploaded file
        #return "File uploaded successfully!"
        return predict()
    return "No file uploaded."

def predict():
    # Get the image from the request
    image = request.files["image"]
    image = PIL.Image.open(image).resize((224, 224))
    x = np.array(image)
    x = np.expand_dims(x, axis=0)
    pred = np.argmax(model.predict(x))

    op = ['Anthracnose', 'Algal Leaf', 'Bird Eye Spot', 'Brown Blight', 'Gray Blight', 'Healthy', 'Red Leaf Spot', 'White Spot']

    # Return the prediction
    return render_template("index.html", prediction=op[pred])


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

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the index.html page earlier.

Main Function:

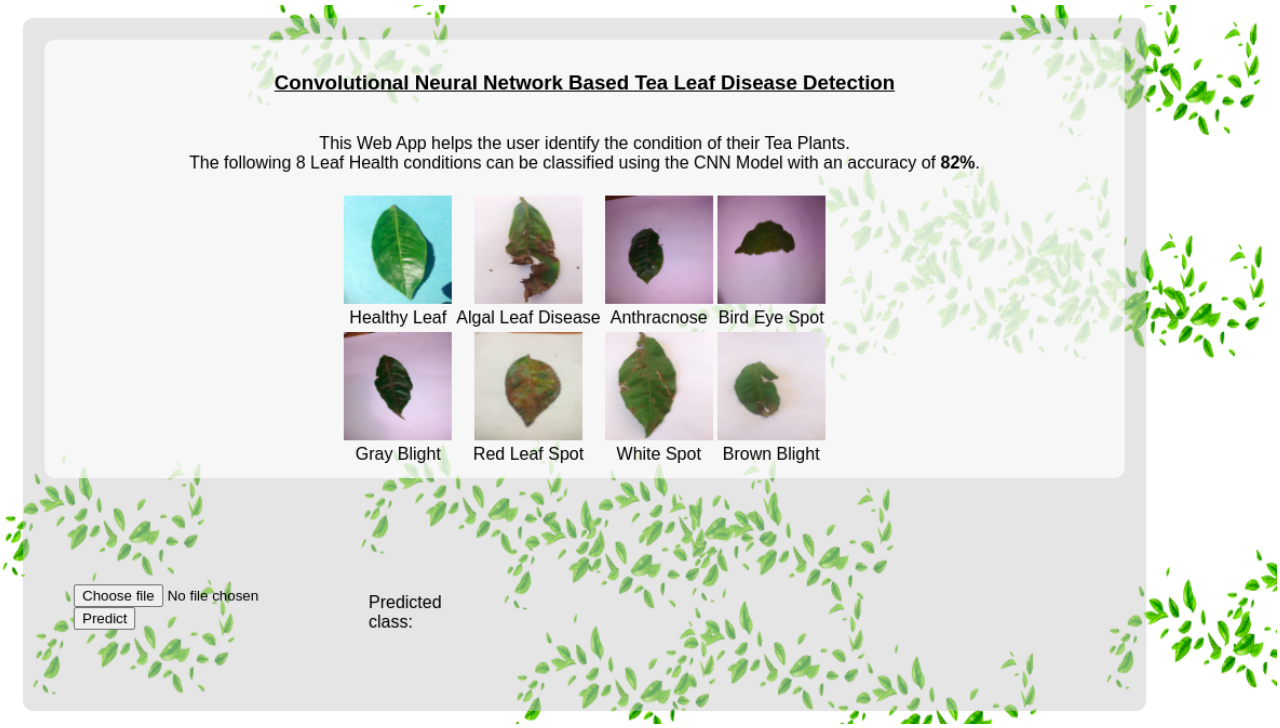
```
if __name__ == '__main__':
    app.run(debug=True)
```

Activity 3: Run the application

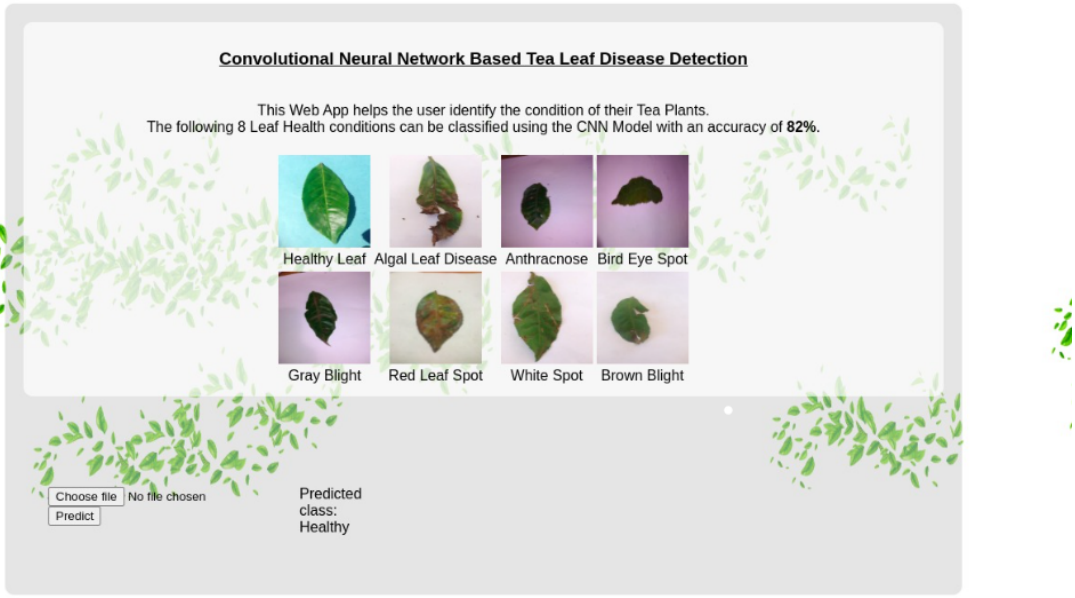
Open Visual studio code and Import all the project folders.

When you run the app.py file and click on the server url in the terminal, you will be redirected to the home page.

The home page will looks like following **Before prediction:**



After Prediction:



Conclusion:

This machine learning project aims to develop a predictive model that can assist tea farmers and tea corporations as well as organizations. By accurately predicting the tea leaf disease, the model can support individual tea farmers, optimize resource allocation, and contribute to a healthier tea farming.

The above description outlines a general approach to a tea leaf disease detection. The specific implementation and details may vary based on the available dataset, project scope, and requirements.