INTELLIGENT DIAGNOSIS AND TREATMENT RECOMMENDATION SYSTEM FOR TEA LEAF DISEASES USING IMAGE CLASSIFICATION AND DOCUMENT-BASED QUESTION ANSWERING

Group: 41
Team Sparta

Team Sparta

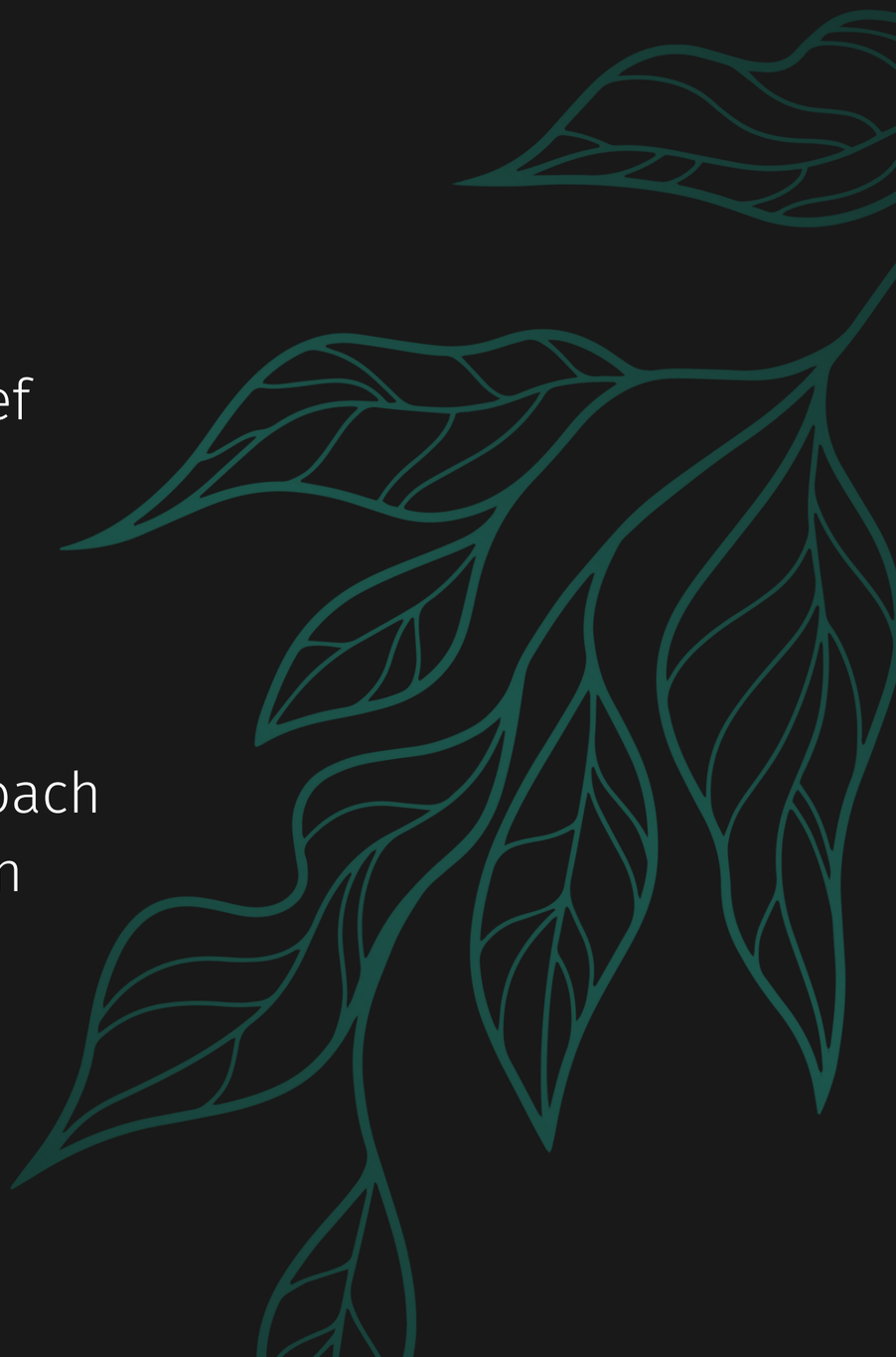
Group Members

184134U - Premachandra P.K.P.G.
184054A - Herath H.M.S.C.B.
184159B - Senanayake T.A.D.S.L.
184182M - Vindula S.G.T.

Supervisor's Name : Ms. Rajapaksha H.P.S.C.

Overview

Introduction
Problem in Brief
Others work
Research Gaps
Aim
Objectives
Design & Approach
Implementation
Further work



Introduction

- Tea is one of Sri Lanka's traditional drinks.
- James Taylor, a British planter introduced the industry to the country in 1867.
- The occurrence of tea leaf diseases has a great impact on the yield and quality of tea.
- Humidity, cool temperatures, and rainfall provide a climate conducive to the production of high-quality tea.

Problem in Brief

- If a tea plant is affected by a disease, there is a high chance of spreading it to nearby plants.
- Identification and classification of tea leaf diseases mainly rely on the expertise of tea farmers.
- Sometimes can be mistaken and it can be very costly.
- Manual classification and quality assessment of tea leaves is a difficult and subjective task, relying on expert judgment and prone to errors.
- Excessive use of pesticides without accurate disease identification can result in unnecessary expenses and environmental impact.
- Automated tea leaf quality inspection is needed to overcome subjectivity and improve efficiency.

Aim

Develop an intelligent system using image classification and question-answering techniques to diagnose tea leaf diseases and provide treatment recommendations.

Objectives

- To classify tea leaf images accurately into different disease categories.
- To differentiate healthy tea leaves from those affected by diseases.
- To enhance the quality of tea leaf images and delineate regions of interest.
- To generate customized disease remedy solutions based on user input.
- To design a user-friendly interface for easy interaction with the system.
- To be scalable and adaptable, allowing for the inclusion of additional disease categories, plant varieties, and new techniques.

Others work

Module 1: Intelligent Leaf Differentiation Module

Research Topic	Dataset	Classifier	Average accuracy rate
"Identification of Plant-Leaf Diseases Using CNN and Transfer-Learning Approach" by Sk Mahmudul Hassan, Arnab Kumar Maji, Michał Jasi, Zbigniew Leonowicz, Elzbieta Jasi	54,305	ResNet, AlexNet, Deep CNN, MobileNetV2	ResNet=92.56 AlexNet=98.64 Deep CNN=96.3 MobileNetV2=97.02
A Novel Herbal Leaf Identification and Authentication Using Deep Learning Neural Network	4050	CNN-LSTM	94.96%.
Machine learning classification of plant genotypes grown under different light conditions through the integration of multi-scale time-series data	501	ConvLSTM2D	56%
"Feature Extraction of Plant Leaf Using Deep Learning" by Muhammad Umair Ahmad, Sidra Ashiq, Gran Badshah, Ali Haider Khan, Muzammil Hussain	-	CNN	98%

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

Research Topic	Summary
A Study on Various Image Processing Techniques" by Chithra P.L and Bhavani P	Studies on image processing techniques often aim to explore different methods and algorithms used in various stages of image processing, including image acquisition, image pre-processing, clustering, segmentation, and classification. In image preprocessing histogram equalization, Laplacian and Harr filtering, unsharp masking, sharpening. Segmentation - binary, K-means and OTSU segmentation algorithm. Classification - SVM and K-Nearest Neighbour(KNN) Classifier.
'Automated Image Data Preprocessing with Deep Reinforcement Learning' Tran Ngoc Minh, Mathieu Sinn, Hoang Thanh Lam, Martin Wistuba	They investigate and evaluate various approaches to the issue of data augmentation in picture categorization. By using straightforward methods like cropping, rotating, and flipping input photos, augmentation is achieved.
"Tea Leaf Disease Recognition Based on Convolutional Neural Networks and Data Augmentation" by Wang et al. (2020)	Used a combination of rotation, flipping, shearing, zooming, and random erasing techniques for data augmentation
'Semantic Image Segmentation: Two Decades of Research' Gabriela Csurka, Riccardo Volpi, Boris Chidlovskii	This attempts to compile two decades of research in the area of image segmentation, and it will suggest a review of existing solutions, starting with early historical approaches and moving on to a discussion of more contemporary deep learning techniques, including the most recent transformer trend.

Module 3: Dual Classifier Module for Accurate Identification of Tea Leaf Diseases

Research Topic	Summary
"Image Recognition of Tea Leaf Diseases Based on Convolutional Neural Network" by Xiaoxiao SUN, Yongyu XU, Zhihao CAO, and Tingting SU	Considering CNN, SVM and BP, CNN can input the original image directly into the network without the preprocessing process of feature extraction, greatly saving time and reducing the limitations of artificial design features.
"Diseases Classification for Tea Plant Using Concatenated Convolution Neural Network" by Dikdik Krisnandi, Hilman F. Pardede, R. Sandra Yuwana, Vicky Zilvan, Ana Heryana, Fani Fauziah, and Vitria Puspitasari Rahadi	Xception is a CNN architecture based entirely on 36 depthwise separable convolution layers. The 36 convolutional layers are constructed into 14 modules. Except for the first and last modules, all modules have linear residual connections.
"Analysis of Convolutional Neural Network based Image Classification Techniques" by Milan Tripathi	The proposed method classifies the fruit by detecting the most important features of the images by applying filters or feature detectors to the input image in order to generate the feature maps or the activation maps by using the activation function. ReLU is the activation function.
The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation" by Davide Chicco, Giuseppe Jurman	Many researchers consider computing the accuracy as the standard way to go. Accuracy fails in providing a fair estimate of the classifier performance in the class unbalanced datasets.

Module 4: Smart Medication and Prevention Suggestion System / Intelligent Disease Identification and Tea Domain Knowledge System

Research Topic	Summary
Representations for Question Answering from Documents with Tables and Text	The research paper proposes a methodology for question answering using both textual documents and tables, employing deep learning techniques and natural language processing to develop a unified representation.
Answer Generation for Retrieval-based Question Answering Systems	The research paper proposes a methodology utilizing information retrieval, natural language processing, and machine learning techniques to improve the accuracy and relevance of answers in retrieval-based question answering systems, while also presenting experimental results to validate its effectiveness.
Question Answering Systems: Analysis and Survey	The research paper presents a systematic methodology for analyzing and surveying question answering systems, categorizing different approaches based on techniques, discussing their advantages and limitations, and exploring the technologies used, possibly supported by experimental results.
Ontology-based Question Answering Systems over Knowledge Bases: A Survey	The research paper provides a systematic survey of ontology-based question answering systems over knowledge bases, categorizing approaches based on techniques and ontologies' utilization, and exploring the technologies involved, with possible experimental results to support the evaluation.

Research Gaps (Cont.)

Module 1: Intelligent Leaf Differentiation Module

- The dataset is too small for most researchers and it causes to decrease the accuracy
- Adding new images to an existing data set and creating a new plant leaves data set with more images
- Struggle with unbalanced datasets where one class is significantly larger than the other. To avoid this, by using a novel augmentation function, each class has 200 images. Re-creating the data set.
- Trying two approches. First, train the data set using the transfer leaning technique and create a custom CNN+ LSTM model considering the accuracy level obtained from it, and try to bring the accuracy level of the custom model close to the accuracy level obtained from transfer leaning.

Research Gaps (Cont.)

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

- Past studies used traditional data augmentation techniques such as rotation, crop, resize, filtering, and flipping.
- A possible novelty is investigating the use of deep learning-based techniques, such as autoencoders, to learn the underlying structure and representation of tea leaf images and improve their quality.
- While autoencoders have been used for image processing in various domains, their effectiveness in tea leaf disease recognition has not been thoroughly investigated.
- Many segmentation algorithms are developed and evaluated on specific datasets, limiting their generalization to unseen data or different domains.

Research Gaps (Cont.)

Module 3: Dual Classifier Module for Accurate Identification of Tea Leaf Diseases

- Researchers used pre-built classifiers for classifications.
  -I implemented a custom CNN model

- Many researchers consider accuracy as the standard way to predict a class.
  -In this work, Accuracy and the F1-score are used to predict a class.

- Combines two classifiers to predict one result.
  -In here, there are custom built CNN model and Xception pre-trained CNN model.
  -Using a decision algorithm, I take the most optimized prediction

Research Gaps (Cont.)

Module 4: Smart Medication and Prevention Suggestion System / Intelligent Disease Identification and Tea Domain Knowledge System

- **Exploration of open-ended user descriptions for tea leaf disease identification:** Most common approach in the tea leaf disease identification domain primarily relies on collecting data through structured questionnaires or pre-defined categories.This innovative approach could potentially reveal new disease manifestations or lesser-known symptoms that might have otherwise been overlooked in conventional research methodologies

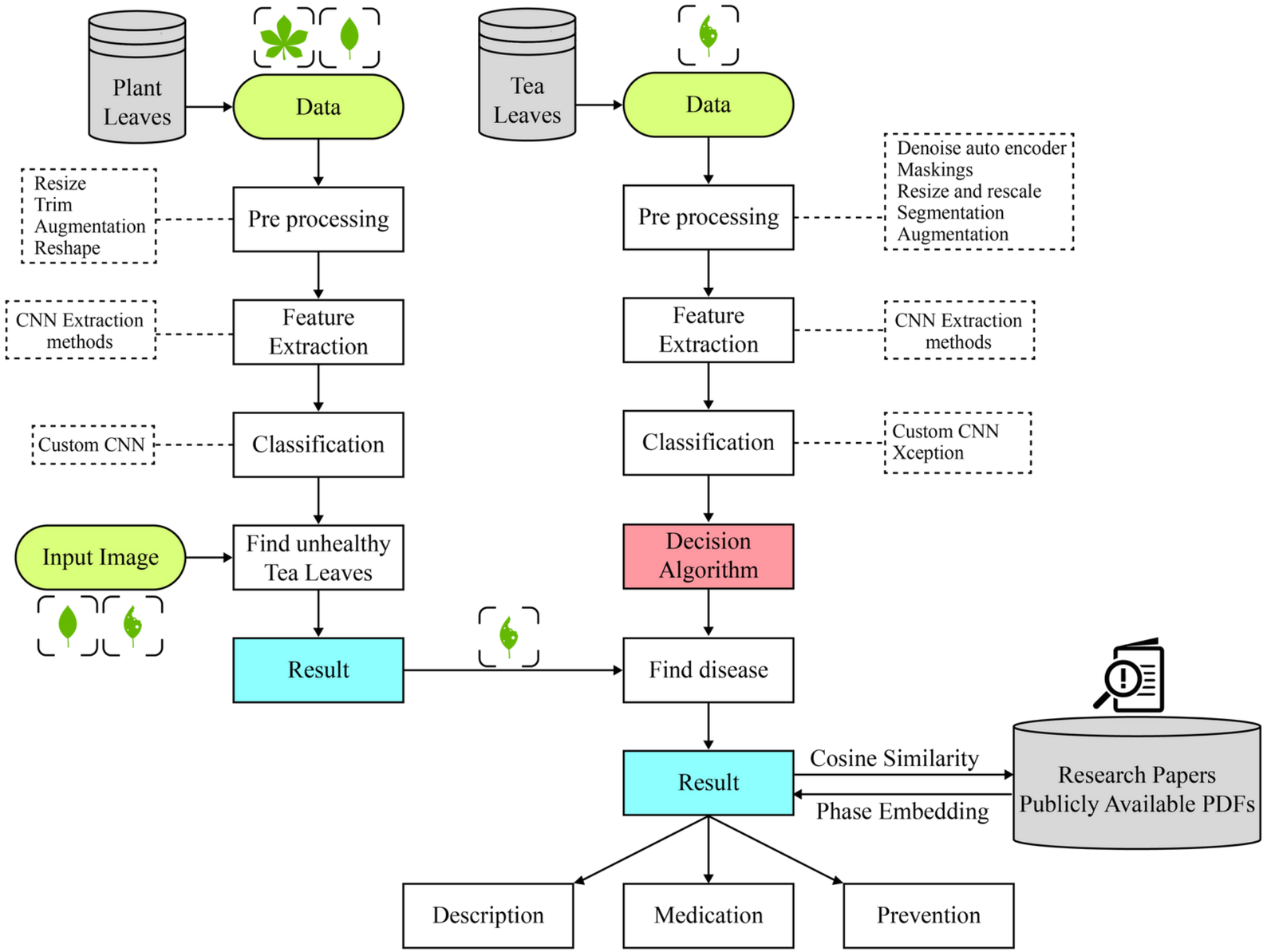
- **Novel approach using Word2Vec for improved embedding:** To tackle the accuracy issues in the Smart Medication and Prevention Suggestion System, a novel approach was proposed involving the use of Word2Vec.
  (I just take Word2Vec hidden weights, use them as our word embeddings)

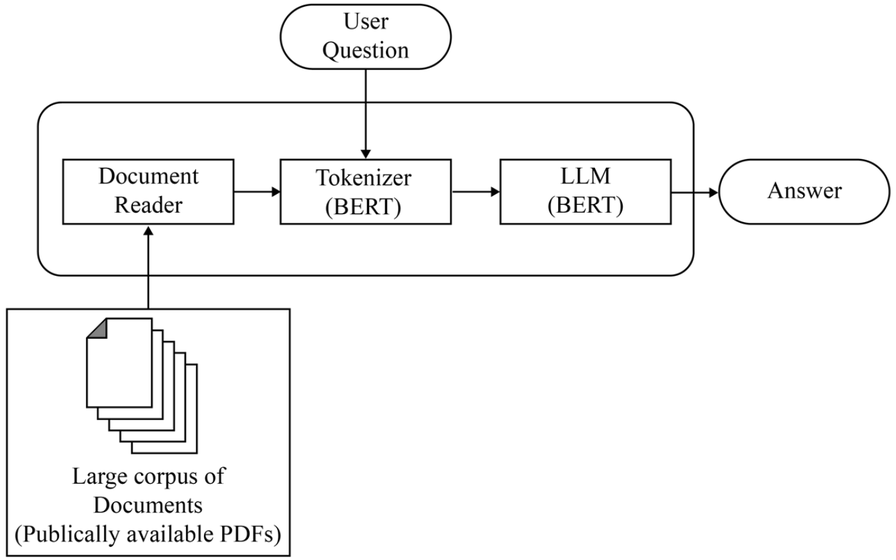
Design & Approach

Design

- The system used text and image (plant leaves and tea leaves) data as its inputs.
- Each module used different preprocessing techniques, feature extraction, and machine learning classifiers.
- Each module has its own architectural design for the process of each module.

Overall Architecture





Approach

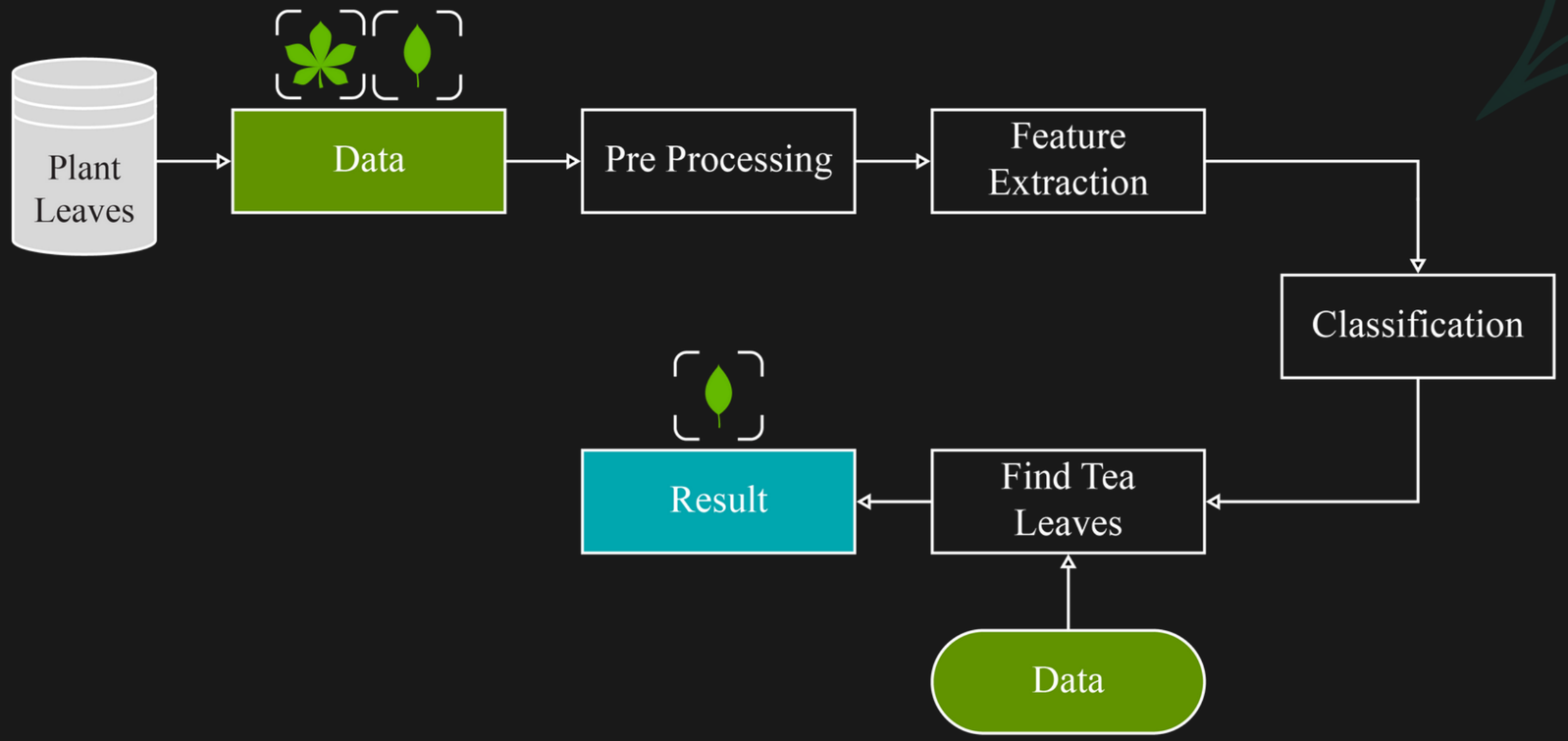
The suggested system consists of four modules:

- Module 01 – Intelligent Leaf Differentiation Module
- Module 02 – Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation.
- Module 03 - Dual Classifier Module for Accurate Identification of Tea Leaf Diseases
- Module 04 – Smart Medication and Prevention Suggestion System / Intelligent Disease Identification and Tea Domain Knowledge System

Module 1: Intelligent Leaf Differentiation Module

Input: Set of images of leaves

Process:

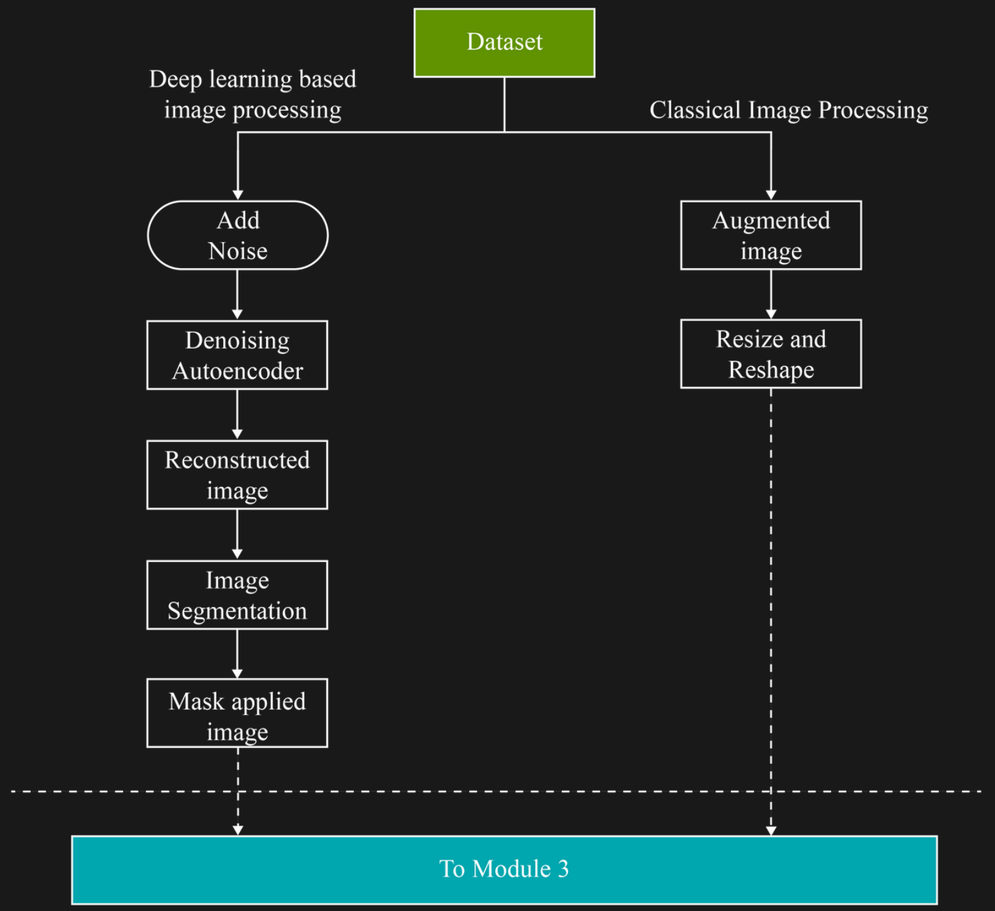


Output: Leaf class and healthy/ diseased

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

Input: Set of images of diseased tea leaves

Process:

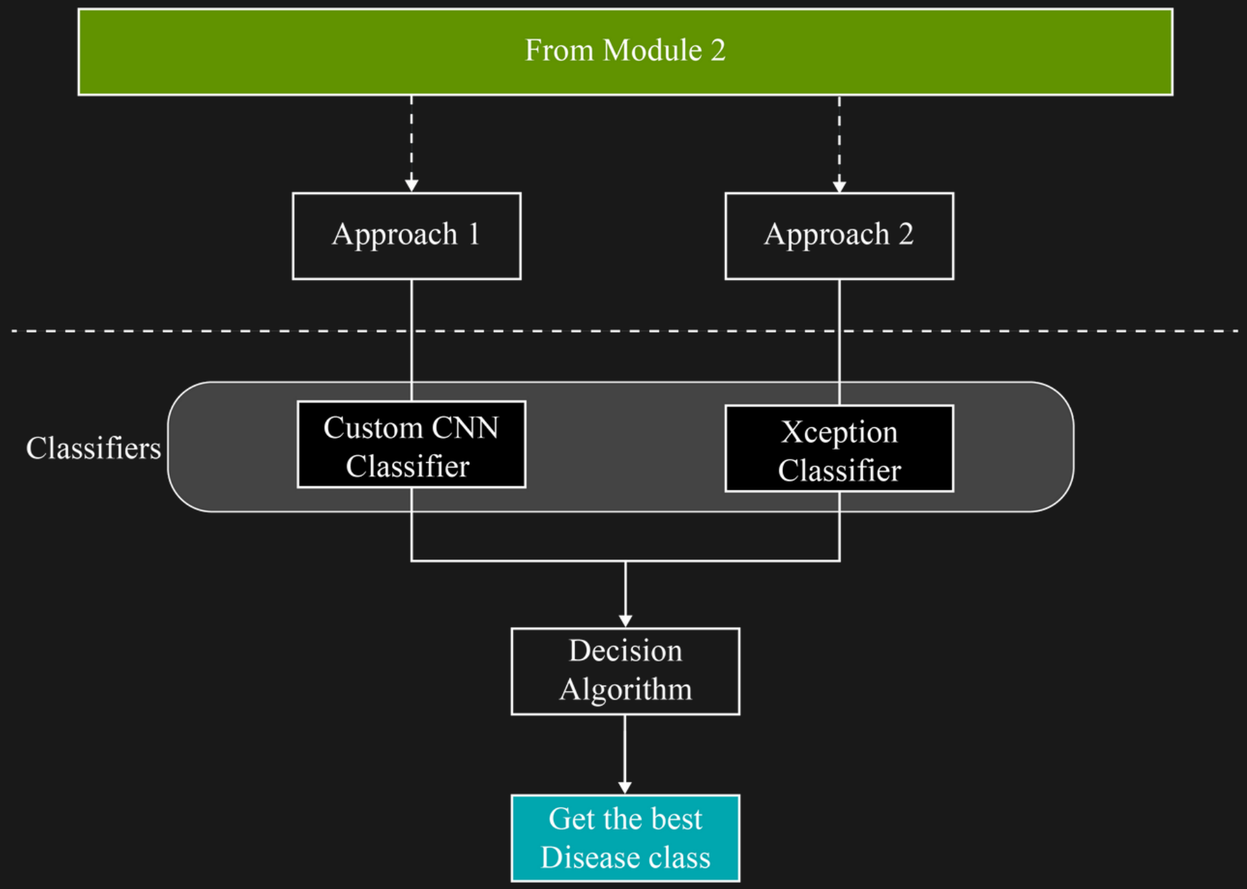


Output: Segmented image according to the disease

Module 3: Dual Classifier Module for Accurate Identification of Tea Leaf Diseases

Input: Set of images of leaves

Process:

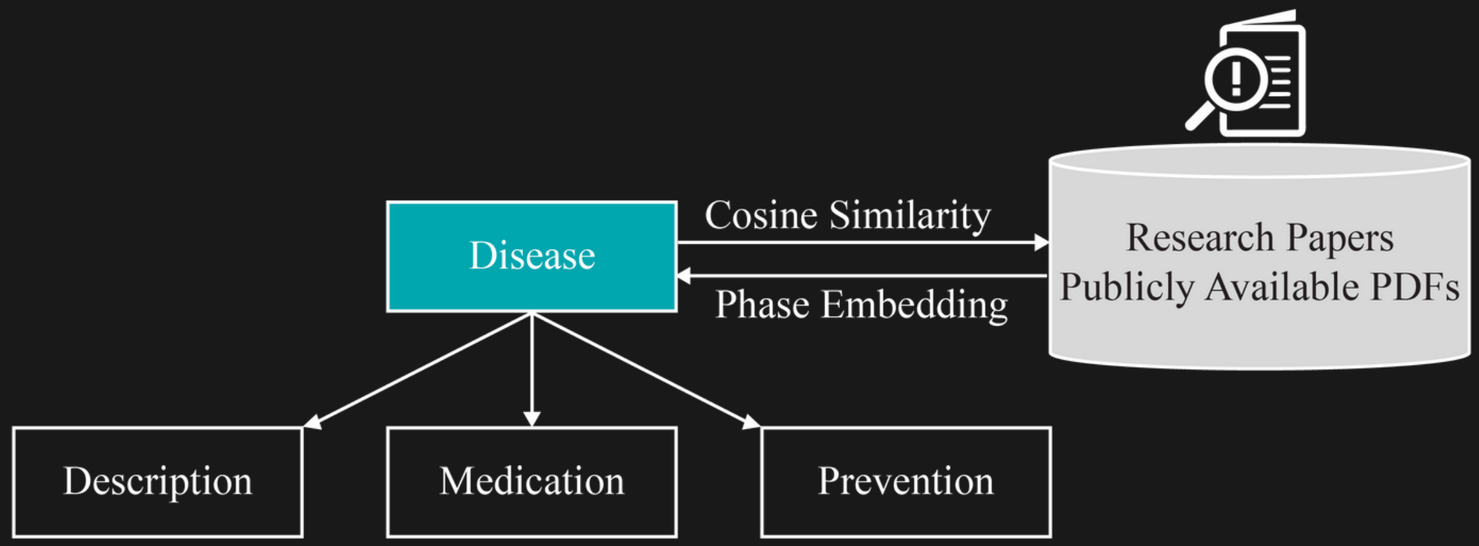


Output: Most accurate disease class

Module 4: Smart Medication and Prevention Suggestion System / Intelligent Disease Identification and Tea Domain Knowledge System

Input: Predicted Disease

Process:

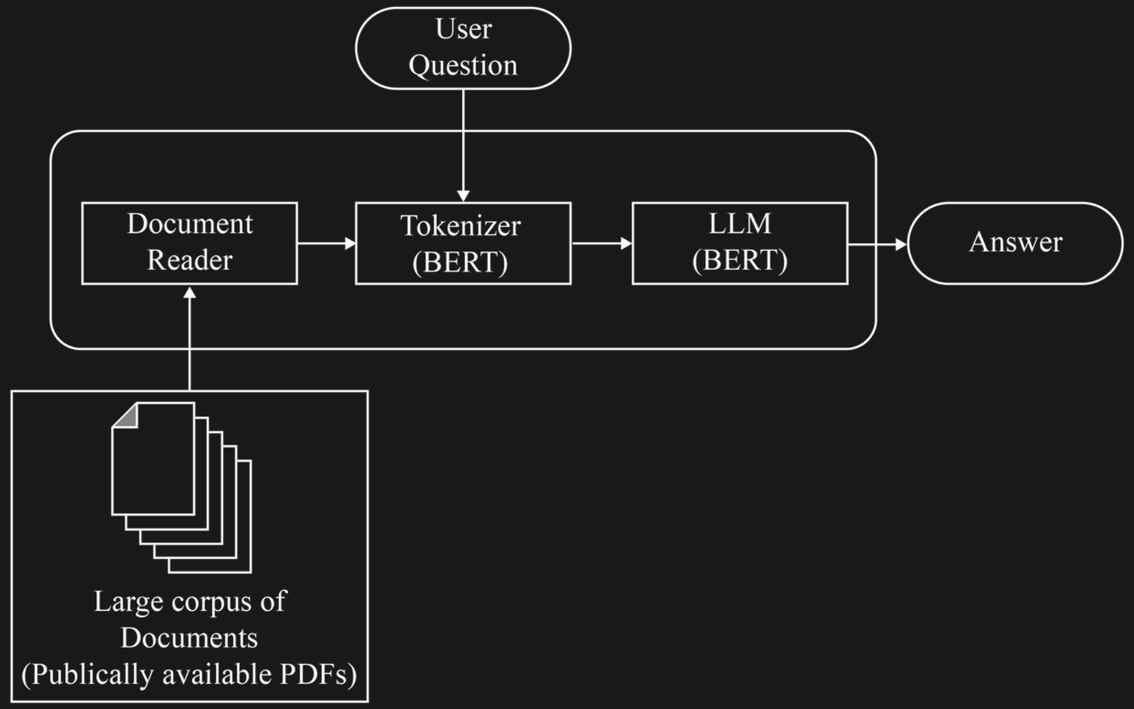


Output: Medication, Preventions and Description about that Predicted Disease

Module 4: Smart Medication and Prevention Suggestion System / Intelligent Disease Identification and Tea Domain Knowledge System

Input: Description about disease(symptoms) / Anything about tea

Process:



Output: Predicted disease / Answers for user's question

Implementation

Module 1: Intelligent Leaf Differentiation Module

- **Image Data Preprocessing:** In this stage convert to reduce data size, and remove noise and balancing the dataset.

- **Feature Extraction:** In this model extracts visual features from input images using convolutional layers.

- **Classification:**

  Approach 01 - The trained model, which incorporates LSTM and CNN layers, accepts input images, extracts visual features, and analyzes temporal patterns. It utilizes these features to classify the input into predefined classes, offering predictions based on the acquired patterns and features

  Approch 02 (Evaluation model) - Using the resnet50 model, the second model was created using the transfer learning technique.. Added some dense layers and increased the accuracy level.

Collect Tea Laves from Different Places in Srilanka

Rathnapura

Nuwara Eliya

Mathale

Kandy

Bandarawela

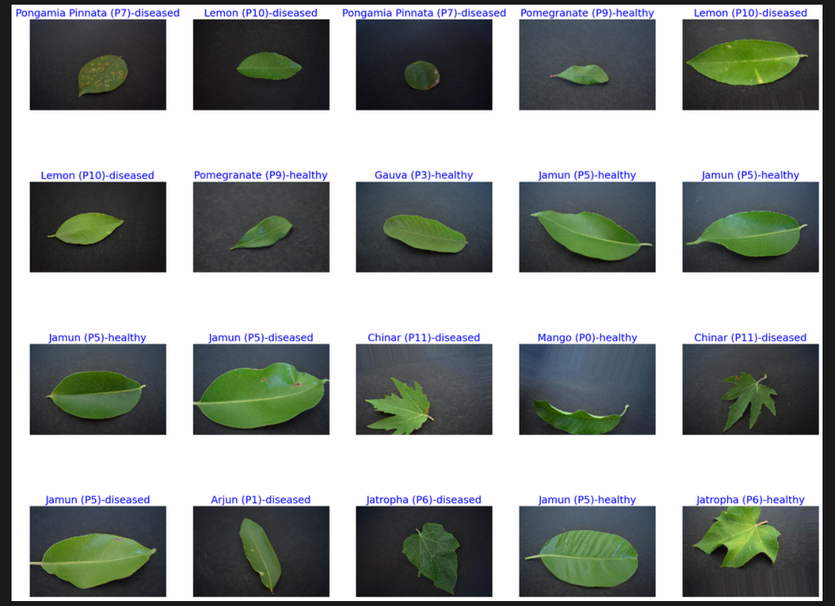
Galle

Applying augmentation and visualizing

```python
def balance(df, n, working_dir, img_size):
    df=df.copy()
    print('Initial length of dataframe is ', len(df))
    aug_dir=os.path.join(working_dir, 'aug')# directory to store augmented images
    if os.path.isdir(aug_dir):# start with an empty directory
        shutil.rmtree(aug_dir)
    os.mkdir(aug_dir)
    for label in df['labels'].unique():
        dir_path=os.path.join(aug_dir,label)
        os.mkdir(dir_path) # make class directories within aug directory
    # create and store the augmented images
    total=0
    gen=ImageDataGenerator(horizontal_flip=True,  rotation_range=20, width_shift_range=.2,
                           height_shift_range=.2, zoom_range=.2)
    groups=df.groupby('labels') # group by class
    for label in df['labels'].unique():  # for every class
        group=groups.get_group(label)  # a dataframe holding only rows with the specified label
        sample_count=len(group)   # determine how many samples there are in this class
        if sample_count< n: # if the class has less than target number of images
            aug_img_count=0
            delta=n - sample_count  # number of augmented images to create
            target_dir=os.path.join(aug_dir, label)  # define where to write the images
            msg='{0:40s} for class {1:^30s} creating {2:^5s} augmented images'.format(' ', label, str(delta))
            print(msg, '\r', end='') # prints over on the same line
            aug_gen=gen.flow_from_dataframe( group,  x_col='filepaths', y_col=None, target_size=img_size,
                                        class_mode=None, batch_size=1, shuffle=False,
                                        save_to_dir=target_dir, save_prefix='aug-', color_mode='rgb',
                                        save_format='jpg')
            while aug_img_count<delta:
                images=next(aug_gen)
                aug_img_count += len(images)
            total +=aug_img_count
    print('Total Augmented images created= ', total)
    # create aug_df and merge with train_df to create composite training set ndf
    aug_fpaths=[]
    aug_labels=[]
    classlist=os.listdir(aug_dir)
    for klass in classlist:
        classpath=os.path.join(aug_dir, klass)
        flist=os.listdir(classpath)
        for f in flist:
            fpath=os.path.join(classpath,f)
            aug_fpaths.append(fpath)
            aug_labels.append(klass)
    Fseries=pd.Series(aug_fpaths, name='filepaths')
    Lseries=pd.Series(aug_labels, name='labels')
    aug_df=pd.concat([Fseries, Lseries], axis=1)
    df=pd.concat([df,aug_df], axis=0).reset_index(drop=True)
    print('Length of augmented dataframe is now ', len(df))
    return df

n=200 # number of samples in each class
working_dir=r'./' # directory to store augmented images
img_size=(200,300) # size of augmented images
train_df=balance(train_df, n, working_dir, img_size)
```

```
Initial length of dataframe is  3331
Found 125 validated image filenames.
Found 91 validated image filenames.
Found 193 validated image filenames.
Found 190 validated image filenames.
Found 72 validated image filenames.
Found 99 validated image filenames.
Found 157 validated image filenames.
Found 111 validated image filenames.
Found 87 validated image filenames.
Found 186 validated image filenames.
Found 54 validated image filenames.
Found 119 validated image filenames.
Found 83 validated image filenames.
Found 162 validated image filenames.
Found 194 validated image filenames.
Found 195 validated image filenames.
Found 93 validated image filenames.
Found 104 validated image filenames.
Found 154 validated image filenames.
Found 178 validated image filenames.
Found 84 validated image filenames.
Total Augmented images created=  1469
Length of augmented dataframe is now  4800
```

```python
def show_image_samples(gen ):
    t_dict=gen.class_indices
    classes=list(t_dict.keys())
    images,labels=next(gen) # get a sample batch from the generator
    plt.figure(figsize=(25, 25))
    length=len(labels)
    if length<25:   #show maximum of 25 images
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5, 5, i + 1)
        image=images[i] /255
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color='blue', fontsize=18)
        plt.axis('off')
    plt.show()

show_image_samples(train_gen )
```



Final CNN + LSTM model

```python
#most accurate
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define image dimensions and number of classes
img_shape=(img_size[0], img_size[1], 3)
# Build a custom sequential CNN model
model = Sequential()

# Add Layers
model.add(Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu', input_shape=(img_size[0], img_size[1], 3)))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flatten the feature map
model.add(Flatten())

# Add fully connected layers
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.1))
# Reshape the flattened features to include time steps
model.add(Reshape((1, -1)))
# Add LSTM Layers
model.add(LSTM(128, return_sequences=True))

model.add(LSTM(64))
model.add(Dense(class_count, activation='softmax'))


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

# print the model summary
model.summary()
```

Model training process

```
Epoch 14/40
240/240 [==============================] - 50s 207ms/step - loss: 0.4115 - accuracy: 0.8625 - val_loss: 0.5819 - val_accuracy: 0.8368
Epoch 15/40
240/240 [==============================] - 52s 218ms/step - loss: 0.4057 - accuracy: 0.8587 - val_loss: 2.9440 - val_accuracy: 0.3786
Epoch 16/40
240/240 [==============================] - 50s 206ms/step - loss: 0.4150 - accuracy: 0.8610 - val_loss: 0.5378 - val_accuracy: 0.8368
Epoch 17/40
240/240 [==============================] - 50s 207ms/step - loss: 0.3375 - accuracy: 0.8852 - val_loss: 0.7320 - val_accuracy: 0.7874
Epoch 18/40
240/240 [==============================] - 50s 208ms/step - loss: 0.4344 - accuracy: 0.8617 - val_loss: 2.3276 - val_accuracy: 0.4582
Epoch 19/40
240/240 [==============================] - 50s 206ms/step - loss: 0.3307 - accuracy: 0.8944 - val_loss: 3.2593 - val_accuracy: 0.3196
Epoch 20/40
240/240 [==============================] - 50s 210ms/step - loss: 0.2706 - accuracy: 0.9096 - val_loss: 2.0456 - val_accuracy: 0.5405
Epoch 21/40
240/240 [==============================] - 50s 207ms/step - loss: 0.2713 - accuracy: 0.9117 - val_loss: 0.5689 - val_accuracy: 0.8422
Epoch 22/40
240/240 [==============================] - 50s 206ms/step - loss: 0.2303 - accuracy: 0.9256 - val_loss: 0.6566 - val_accuracy: 0.8189
Epoch 23/40
240/240 [==============================] - 50s 208ms/step - loss: 0.2347 - accuracy: 0.9271 - val_loss: 0.4444 - val_accuracy: 0.8779
Epoch 24/40
240/240 [==============================] - 50s 209ms/step - loss: 0.2095 - accuracy: 0.9356 - val_loss: 1.4597 - val_accuracy: 0.6145
Epoch 25/40
240/240 [==============================] - 49s 205ms/step - loss: 0.2112 - accuracy: 0.9292 - val_loss: 0.4012 - val_accuracy: 0.8875
Epoch 26/40
240/240 [==============================] - 50s 207ms/step - loss: 0.2140 - accuracy: 0.9281 - val_loss: 0.4408 - val_accuracy: 0.8724
Epoch 27/40
240/240 [==============================] - 50s 208ms/step - loss: 0.2310 - accuracy: 0.9258 - val_loss: 0.4867 - val_accuracy: 0.8491
Epoch 28/40
240/240 [==============================] - 52s 219ms/step - loss: 0.2082 - accuracy: 0.9331 - val_loss: 0.5598 - val_accuracy: 0.8505
Epoch 29/40
240/240 [==============================] - 50s 209ms/step - loss: 0.2026 - accuracy: 0.9375 - val_loss: 0.8528 - val_accuracy: 0.7750
Epoch 30/40
240/240 [==============================] - 50s 209ms/step - loss: 0.1664 - accuracy: 0.9448 - val_loss: 2.1838 - val_accuracy: 0.4993
Epoch 31/40
240/240 [==============================] - 50s 207ms/step - loss: 0.1582 - accuracy: 0.9502 - val_loss: 0.9235 - val_accuracy: 0.7572
Epoch 32/40
240/240 [==============================] - 50s 208ms/step - loss: 0.1514 - accuracy: 0.9504 - val_loss: 1.2838 - val_accuracy: 0.6927
Epoch 33/40
240/240 [==============================] - 50s 209ms/step - loss: 0.1402 - accuracy: 0.9502 - val_loss: 0.8307 - val_accuracy: 0.7888
Epoch 34/40
240/240 [==============================] - 49s 206ms/step - loss: 0.1331 - accuracy: 0.9594 - val_loss: 0.8366 - val_accuracy: 0.7984
Epoch 35/40
240/240 [==============================] - 50s 208ms/step - loss: 0.1362 - accuracy: 0.9550 - val_loss: 0.7047 - val_accuracy: 0.8230
```

ResNet50 Model

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the pre-trained ResNet50 model (excluding the top fully connected layers)
resnet50 = ResNet50(weights='imagenet', include_top=False, input_shape=(img_size[0], img_size[1], 3))

# Freeze the weights of all layers except the last 8 layers
for layer in resnet50.layers[:-8]:
    layer.trainable = False

# Create a new model
model = Sequential()

# Add the ResNet50 model as a layer
model.add(resnet50)

# Flatten the output of the ResNet50 model
model.add(Flatten())

# Add your own fully connected layers
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(class_count, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

ResNet50 Model training process

```
Epoch 15/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0431 - accuracy: 0.9869 - val_loss: 0.2306 - val_accuracy: 0.9273
Epoch 16/40
240/240 [==============================] - 30s 126ms/step - loss: 0.0200 - accuracy: 0.9950 - val_loss: 0.1588 - val_accuracy: 0.9575
Epoch 17/40
240/240 [==============================] - 29s 119ms/step - loss: 0.0217 - accuracy: 0.9950 - val_loss: 0.1862 - val_accuracy: 0.9479
Epoch 18/40
240/240 [==============================] - 29s 120ms/step - loss: 0.0228 - accuracy: 0.9944 - val_loss: 0.1869 - val_accuracy: 0.9534
Epoch 19/40
240/240 [==============================] - 29s 119ms/step - loss: 0.0118 - accuracy: 0.9965 - val_loss: 0.1542 - val_accuracy: 0.9561
Epoch 20/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0152 - accuracy: 0.9962 - val_loss: 0.2028 - val_accuracy: 0.9520
Epoch 21/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0099 - accuracy: 0.9975 - val_loss: 0.1796 - val_accuracy: 0.9547
Epoch 22/40
240/240 [==============================] - 29s 119ms/step - loss: 0.0037 - accuracy: 0.9996 - val_loss: 0.2159 - val_accuracy: 0.9561
Epoch 23/40
240/240 [==============================] - 29s 120ms/step - loss: 0.0087 - accuracy: 0.9977 - val_loss: 0.2046 - val_accuracy: 0.9479
Epoch 24/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0082 - accuracy: 0.9973 - val_loss: 0.1974 - val_accuracy: 0.9588
Epoch 25/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0122 - accuracy: 0.9971 - val_loss: 0.1786 - val_accuracy: 0.9547
Epoch 26/40
240/240 [==============================] - 29s 120ms/step - loss: 0.0194 - accuracy: 0.9929 - val_loss: 0.2528 - val_accuracy: 0.9369
Epoch 27/40
240/240 [==============================] - 30s 125ms/step - loss: 0.0325 - accuracy: 0.9904 - val_loss: 0.3560 - val_accuracy: 0.9122
Epoch 28/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0170 - accuracy: 0.9956 - val_loss: 0.2223 - val_accuracy: 0.9383
Epoch 29/40
240/240 [==============================] - 29s 119ms/step - loss: 0.0131 - accuracy: 0.9975 - val_loss: 0.4113 - val_accuracy: 0.9355
Epoch 30/40
240/240 [==============================] - 28s 119ms/step - loss: 0.0166 - accuracy: 0.9958 - val_loss: 0.2944 - val_accuracy: 0.9383
Epoch 31/40
240/240 [==============================] - 30s 126ms/step - loss: 0.0171 - accuracy: 0.9952 - val_loss: 0.2664 - val_accuracy: 0.9479
Epoch 32/40
240/240 [==============================] - 29s 119ms/step - loss: 0.0217 - accuracy: 0.9946 - val_loss: 0.2337 - val_accuracy: 0.9465
Epoch 33/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0163 - accuracy: 0.9962 - val_loss: 0.2402 - val_accuracy: 0.9561
Epoch 34/40
240/240 [==============================] - 28s 118ms/step - loss: 0.0077 - accuracy: 0.9977 - val_loss: 0.2284 - val_accuracy: 0.9424
```

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

Image Preprocessing

Denoising Autoencoder

```
# Add noise to the images
noise_factor = 0.1
noisy_data = data + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=data.shape)
noisy_data = np.clip(noisy_data, 0., 1.)

# Define the encoder network
encoder_input = tf.keras.layers.Input(shape=(img_size[0], img_size[1], 3))
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_input)
x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
x = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
encoded = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)

# Define the decoder network
x = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(encoded)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
decoded = tf.keras.layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

# Define the autoencoder model
autoencoder = tf.keras.Model(encoder_input, decoded)
```

Blurred Image

fastNlMeansDenoisingColored function

```
def image_denoiser(
                    image_path,
                    apply_denoising = False
                    ):
    img = cv.imread(image_path)
    if apply_denoising:
        img = cv.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
    return img

def preprocess_input(image_path):
    img = image_denoiser(image_path)
    original_img = img
```

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

Image Segmentation

Explored using of different segmentation techniques for specific tea leaf diseases, depending on the characteristics and requirements of the disease detection. But finally chose Semantic segmentation model that can be trained on large datasets to learn complex patterns and generalize well to unseen images.

Semantic Segmentation

UNet Model

```
def unet_model():
    input = tf.keras.layers.Input(shape=input_shape, name='input')

    if input_shape[0] == 128:
        c1 = tf.keras.layers.Conv2D(16, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(input)
        c1 = tf.keras.layers.Dropout(0.1)(c1)
        c1 = tf.keras.layers.Conv2D(16, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(c1)
        p1 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c1)

        c2 = tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(p1)
        c2 = tf.keras.layers.Dropout(0.1)(c2)
        c2 = tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(c2)
        p2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c2)

        c3 = tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(p2)
        c3 = tf.keras.layers.Dropout(0.2)(c3)
        c3 = tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(c3)
        p3 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c3)

        c4 = tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(p3)
        c4 = tf.keras.layers.Dropout(0.2)(c4)
        c4 = tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(c4)
        p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

        c5 = tf.keras.layers.Conv2D(256, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(p4)
        c5 = tf.keras.layers.Dropout(0.3)(c5)
        c5 = tf.keras.layers.Conv2D(256, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal')(c5)
```

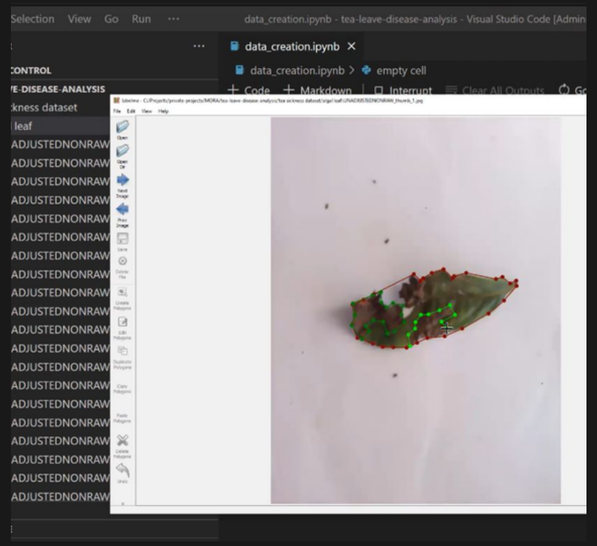
Model Summary

Model: "disease_segmentation"

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 256, 256, 3)]	0	[]
conv2d (Conv2D)	(None, 256, 256, 32)	896	['input[0][0]']
dropout (Dropout)	(None, 256, 256, 32)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 256, 256, 32)	9248	['dropout[0][0]']
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 128, 128, 64)	18496	['max_pooling2d[0][0]']
dropout_1 (Dropout)	(None, 128, 128, 64)	0	['conv2d_2[0][0]']
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36928	['dropout_1[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0	['conv2d_3[0][0]']
conv2d_4 (Conv2D)	(None, 64, 64, 128)	73856	['max_pooling2d_1[0][0]']

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

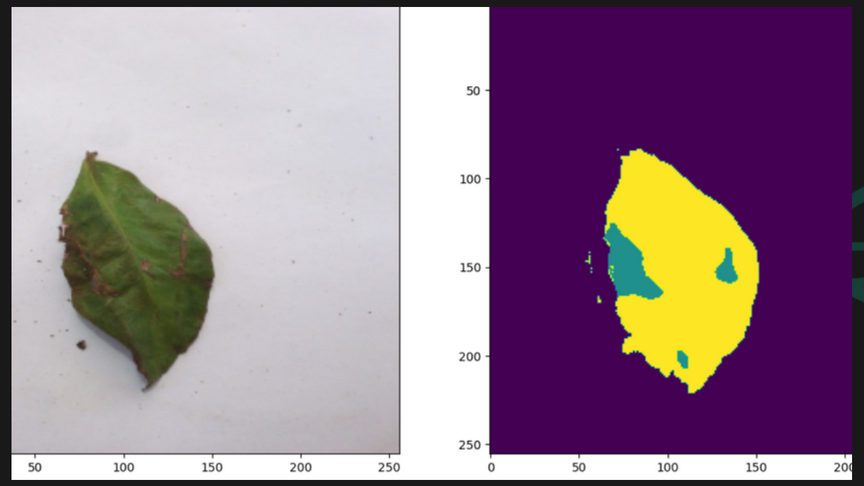
Annotated Image



Model training process

```
153/153 [==============================] - 13s 82ms/step - loss: 0.2643 - categorical_accuracy: 0.9327 - precision: 0.9394 - recall: 0.9331
Epoch 53/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2636 - categorical_accuracy: 0.9318 - precision: 0.9386 - recall: 0.9323
Epoch 54/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2628 - categorical_accuracy: 0.9317 - precision: 0.9386 - recall: 0.9323
Epoch 55/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2609 - categorical_accuracy: 0.9326 - precision: 0.9392 - recall: 0.9333
Epoch 56/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2625 - categorical_accuracy: 0.9319 - precision: 0.9384 - recall: 0.9326
Epoch 57/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2608 - categorical_accuracy: 0.9330 - precision: 0.9391 - recall: 0.9338
Epoch 58/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2604 - categorical_accuracy: 0.9312 - precision: 0.9380 - recall: 0.9322
Epoch 59/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2598 - categorical_accuracy: 0.9323 - precision: 0.9386 - recall: 0.9335
Epoch 60/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2609 - categorical_accuracy: 0.9315 - precision: 0.9378 - recall: 0.9328
Epoch 61/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2598 - categorical_accuracy: 0.9315 - precision: 0.9375 - recall: 0.9329
Epoch 62/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2593 - categorical_accuracy: 0.9319 - precision: 0.9378 - recall: 0.9335
Epoch 63/100
153/153 [==============================] - 13s 82ms/step - loss: 0.2605 - categorical_accuracy: 0.9311 - precision: 0.9369 - recall: 0.9328
```

Segmented Image



Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

Classical Image Preprocessing

Resize and rescale images

```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
```

Image Augmentation

```
[ ]  data_augmentation = tf.keras.Sequential([
       layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
       layers.experimental.preprocessing.RandomRotation(0.2),
     ])
```

Module 3: Dual Classifier Module for Accurate Identification of Tea Leaf Diseases

- **Feature Extraction:** In the CNN model, Convolution layers and max-pooling layers are used to extract the features

- **Classification:**

  Approach 01 (Evaluation model) - Xception CNN model used as Approach 1.

  Approch 02 - Classification using the custom build CNN model. Extracting features and analyzing the patterns. This model uses that features to classify the image input to the model and classify to the predefined disease classes.

  Decision Algorithm - In here, the outputs of the 2 approaches are compare using weighted F1-score and the accuracy. Then gives the output of the most accurate model for particular input.

Custom CNN model

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 5

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

Model training process

```
21/21 [==============================] - 8s 328ms/step - loss: 0.3965 - accuracy: 0.8349 - val_loss: 0.2860 - val_accuracy: 0.8828
Epoch 63/150
21/21 [==============================] - 10s 373ms/step - loss: 0.2783 - accuracy: 0.8894 - val_loss: 0.2638 - val_accuracy: 0.8906
Epoch 64/150
21/21 [==============================] - 9s 386ms/step - loss: 0.3043 - accuracy: 0.8692 - val_loss: 0.3771 - val_accuracy: 0.8516
Epoch 65/150
21/21 [==============================] - 8s 335ms/step - loss: 0.2860 - accuracy: 0.8847 - val_loss: 0.3773 - val_accuracy: 0.8281
Epoch 66/150
21/21 [==============================] - 9s 334ms/step - loss: 0.2631 - accuracy: 0.8910 - val_loss: 0.2341 - val_accuracy: 0.8984
Epoch 67/150
21/21 [==============================] - 10s 418ms/step - loss: 0.2411 - accuracy: 0.8941 - val_loss: 0.3986 - val_accuracy: 0.8438
Epoch 68/150
21/21 [==============================] - 9s 368ms/step - loss: 0.3248 - accuracy: 0.8785 - val_loss: 0.3069 - val_accuracy: 0.8672
Epoch 69/150
21/21 [==============================] - 8s 337ms/step - loss: 0.2394 - accuracy: 0.8941 - val_loss: 0.5564 - val_accuracy: 0.7969
Epoch 70/150
21/21 [==============================] - 10s 371ms/step - loss: 0.5218 - accuracy: 0.7897 - val_loss: 0.4427 - val_accuracy: 0.7891
Epoch 71/150
21/21 [==============================] - 10s 429ms/step - loss: 0.2854 - accuracy: 0.8863 - val_loss: 0.3988 - val_accuracy: 0.8359
Epoch 72/150
21/21 [==============================] - 9s 354ms/step - loss: 0.3159 - accuracy: 0.8692 - val_loss: 0.4040 - val_accuracy: 0.8438
Epoch 73/150
21/21 [==============================] - 9s 333ms/step - loss: 0.3023 - accuracy: 0.8879 - val_loss: 0.8505 - val_accuracy: 0.7109
Epoch 74/150
21/21 [==============================] - 10s 405ms/step - loss: 0.3712 - accuracy: 0.8489 - val_loss: 0.3375 - val_accuracy: 0.8359
Epoch 75/150
21/21 [==============================] - 9s 379ms/step - loss: 0.2965 - accuracy: 0.8832 - val_loss: 0.2899 - val_accuracy: 0.8828
Epoch 76/150
21/21 [==============================] - 8s 319ms/step - loss: 0.2182 - accuracy: 0.9003 - val_loss: 0.3462 - val_accuracy: 0.8516
Epoch 77/150
21/21 [==============================] - 10s 411ms/step - loss: 0.3022 - accuracy: 0.8816 - val_loss: 0.4314 - val_accuracy: 0.8125
Epoch 78/150
21/21 [==============================] - 9s 355ms/step - loss: 0.4765 - accuracy: 0.8131 - val_loss: 0.6657 - val_accuracy: 0.6875
Epoch 79/150
21/21 [==============================] - 8s 337ms/step - loss: 0.3407 - accuracy: 0.8567 - val_loss: 0.3276 - val_accuracy: 0.8438
Epoch 80/150
21/21 [==============================] - 9s 371ms/step - loss: 0.2276 - accuracy: 0.9065 - val_loss: 0.4410 - val_accuracy: 0.8203
Epoch 81/150
21/21 [==============================] - 9s 390ms/step - loss: 0.5079 - accuracy: 0.8053 - val_loss: 0.3458 - val_accuracy: 0.8984
Epoch 82/150
21/21 [==============================] - 8s 334ms/step - loss: 0.3491 - accuracy: 0.8723 - val_loss: 0.3251 - val_accuracy: 0.8438
Epoch 83/150
21/21 [==============================] - 10s 366ms/step - loss: 0.2421 - accuracy: 0.8816 - val_loss: 0.4524 - val_accuracy: 0.7812
Epoch 84/150
21/21 [==============================] - 10s 398ms/step - loss: 0.2970 - accuracy: 0.8754 - val_loss: 0.6560 - val_accuracy: 0.7578
Epoch 85/150
21/21 [==============================] - 8s 326ms/step - loss: 0.5133 - accuracy: 0.8053 - val_loss: 0.7945 - val_accuracy: 0.6875
Epoch 86/150
21/21 [==============================] - 10s 368ms/step - loss: 0.3739 - accuracy: 0.8489 - val_loss: 0.5438 - val_accuracy: 0.7578
```

Exception model

```
def encoder_model():
    input = tf.keras.layers.Input(shape=input_shape, name='input')

    xception_model = tf.keras.applications.Xception(
                                                    include_top=False,
                                                    weights='imagenet',
                                                    input_shape=input_shape
                                                    )
    xception_model.trainable = False
    xception_output = xception_model(input)

    x = tf.keras.layers.GlobalAveragePooling2D()(xception_output)
    x = tf.keras.layers.Dropout(0.2)(x)

    x = tf.keras.layers.Dense(128, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)

    x = tf.keras.layers.Dense(64, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)

    output = tf.keras.layers.Dense(5, activation='softmax')(x)


    model = tf.keras.models.Model(
                                inputs=input,
                                outputs=output,
                                name='disease_classification'
                                )
    model.summary()

    return model
```

Model building

```
def train_classification(model):
    early_stopping = tf.keras.callbacks.EarlyStopping(
                                                        monitor='loss',
                                                        patience=25,
                                                        restore_best_weights=True
                                                        )


    model.compile(
                loss='categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=[
                        tf.keras.metrics.CategoricalAccuracy(),
                        tf.keras.metrics.Precision(),
                        tf.keras.metrics.Recall()
                        ]
                )
    model.fit(
            X,
            tf.keras.utils.to_categorical(Y2, num_classes=5),
            epochs=100,
            batch_size=4,
            callbacks=[early_stopping]
            )

    model.save(classification_weights)
    return model
```

Xception Model summery

```
Model: "disease_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 256, 256, 3)]     0

 xception (Functional)       (None, 8, 8, 2048)        20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout_7 (Dropout)         (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 dropout_8 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_9 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 5)                 325

=================================================================
Total params: 21,132,333
Trainable params: 270,853
Non-trainable params: 20,861,480
```

Xception Model training process

```
Epoch 77/100
153/153 [==============================] - 3s 23ms/step - loss: 0.0430 - categorical_accuracy: 0.9886 - precision_1: 0.9902 - recall_1: 0.9886
Epoch 78/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0614 - categorical_accuracy: 0.9771 - precision_1: 0.9787 - recall_1: 0.9771
Epoch 79/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0241 - categorical_accuracy: 0.9918 - precision_1: 0.9918 - recall_1: 0.9918
Epoch 80/100
153/153 [==============================] - 3s 23ms/step - loss: 0.0259 - categorical_accuracy: 0.9967 - precision_1: 0.9967 - recall_1: 0.9951
Epoch 81/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0597 - categorical_accuracy: 0.9788 - precision_1: 0.9787 - recall_1: 0.9755
Epoch 82/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0370 - categorical_accuracy: 0.9853 - precision_1: 0.9918 - recall_1: 0.9853
Epoch 83/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0237 - categorical_accuracy: 0.9935 - precision_1: 0.9951 - recall_1: 0.9935
Epoch 84/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0203 - categorical_accuracy: 0.9918 - precision_1: 0.9918 - recall_1: 0.9902
Epoch 85/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0593 - categorical_accuracy: 0.9837 - precision_1: 0.9836 - recall_1: 0.9820
Epoch 86/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0499 - categorical_accuracy: 0.9804 - precision_1: 0.9804 - recall_1: 0.9804
Epoch 87/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0695 - categorical_accuracy: 0.9771 - precision_1: 0.9771 - recall_1: 0.9771
Epoch 88/100
153/153 [==============================] - 3s 23ms/step - loss: 0.0468 - categorical_accuracy: 0.9886 - precision_1: 0.9886 - recall_1: 0.9886
Epoch 89/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0587 - categorical_accuracy: 0.9837 - precision_1: 0.9836 - recall_1: 0.9804
Epoch 90/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0185 - categorical_accuracy: 0.9918 - precision_1: 0.9918 - recall_1: 0.9918
Epoch 91/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0526 - categorical_accuracy: 0.9837 - precision_1: 0.9836 - recall_1: 0.9820
Epoch 92/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0325 - categorical_accuracy: 0.9918 - precision_1: 0.9935 - recall_1: 0.9918
Epoch 93/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0340 - categorical_accuracy: 0.9918 - precision_1: 0.9918 - recall_1: 0.9918
Epoch 94/100
153/153 [==============================] - 3s 22ms/step - loss: 0.0643 - categorical_accuracy: 0.9820 - precision_1: 0.9820 - recall_1: 0.9820
Epoch 95/100
153/153 [==============================] - 3s 23ms/step - loss: 0.0444 - categorical_accuracy: 0.9820 - precision_1: 0.9852 - recall_1: 0.9820
Epoch 96/100
153/153 [==============================] - 4s 23ms/step - loss: 0.0496 - categorical_accuracy: 0.9820 - precision_1: 0.9820 - recall_1: 0.9820
```

Decision algorithm

```python
def gr_func(img_path):
    global disease
    # Approach 01 result
    app01_pred = inference_tea_leaves(img_path)

    # Approach 02 result
    app02_pred = predict(img_path)

    # Assign weights to F1 scores and accuracies
    f1_weight = 0.6
    accuracy_weight = 0.4

    # Calculate weighted averages
    app01_weighted_avg = (f1_weight * xception_report['weighted avg']['f1-score']) + (accuracy_weight * xception_report['accuracy'])
    app02_weighted_avg = (f1_weight * custom_cnn_report['weighted avg']['f1-score']) + (accuracy_weight * custom_cnn_report['accuracy'])

    # Compare weighted averages
    if app01_weighted_avg > app02_weighted_avg:
        best_pred = app01_pred
    elif (app02_weighted_avg > app01_weighted_avg):
        best_pred = app02_pred
    else:
        # Handle tiebreaker
        if xception_report['weighted avg']['f1-score'] > custom_cnn_report['weighted avg']['f1-score']:
            best_pred = app01_pred
        else:
            best_pred = app02_pred

    disease = best_pred

    return [app01_pred, app02_pred, best_pred]
```

Module 4: Smart Medication and Prevention Suggestion System

Extract text from the PDFs

```
[18] import pdfplumber
     import glob

     symptoms_txt = ''
     knowledge_txt = ''
     medication_txt= ''

     symptoms_dir_path = r'/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Disease Identification/*.*'
     knowledge_dir_path = r'/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Tea Knowledge Base/*.*'
     medication_dir_path = r'/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Medications & Preventions/*.*'

     for file in glob.glob(symptoms_dir_path, recursive=True):
         print(file)
         with pdfplumber.open(file) as pdf:
             for pdf_page in pdf.pages:
                 single_page_text = pdf_page.extract_text()
                 symptoms_txt = symptoms_txt + '\n' + single_page_text
         pdf.close()

     for file in glob.glob(knowledge_dir_path, recursive=True):
         print(file)
         with pdfplumber.open(file) as pdf:
             for pdf_page in pdf.pages:
                 single_page_text = pdf_page.extract_text()
                 knowledge_txt = knowledge_txt + '\n' + single_page_text
         pdf.close()

     for file in glob.glob(medication_dir_path, recursive=True):
         print(file)
         with pdfplumber.open(file) as pdf:
             for pdf_page in pdf.pages:
                 single_page_text = pdf_page.extract_text()
                 medication_txt = medication_txt + '\n' + single_page_text
         pdf.close()
```

```
/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Disease Identification/Tea Leaf Disease and Symptoms.pdf
/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Tea Knowledge Base/Tea.pdf
/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Tea Knowledge Base/Tea - Wikipedia.pdf
/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Medications & Preventions/Tea Leaf Disease Medications2.pdf
/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Medications & Preventions/Tea Leaf Disease Medications3.pdf
/content/gdrive/MyDrive/Colab Notebooks/research/data/Tea/all-about-tea/Medications & Preventions/Tea Leaf Disease Medications.pdf
```

Preprocessing the text data

```
import re
import gensim
from gensim.parsing.preprocessing import remove_stopwords

def clean_sentence(sentence, stopwords=False):
  sentence = sentence.lower().strip()
  sentence = re.sub(r'[^a-z0-9\s]', '', sentence)
  if stopwords:
    sentence = remove_stopwords(sentence)
  return sentence

def get_cleaned_sentences(tokens, stopwords=False):
  cleaned_sentences = []
  for row in tokens:
    cleaned = clean_sentence(row, stopwords)
    cleaned_sentences.append(cleaned)
  return cleaned_sentences
```

```
[ ] cleaned_sentences = get_cleaned_sentences(tokens, stopwords=True)
    cleaned_sentences_with_stopwords = get_cleaned_sentences(tokens, stopwords=False)
    print(cleaned_sentences)
    print(cleaned_sentences_with_stopwords)
```

```
['tea leaf rust description tea leaf rust tea leaf rust fungal disease caused pathogen hemileia vasta
['tea leaf rust\ndescription for tea leaf rust tea leaf rust is a fungal disease caused by the pathog
```

Module 4: Smart Medication and Prevention Suggestion System

Create a Dictionary of words

```
import numpy
sentences = cleaned_sentences_with_stopwords
sentence_words = [[word for word in document.split()]
                  for document in sentences]

from gensim import corpora
dictionary = corpora.Dictionary(sentence_words)
for key, value in dictionary.items():
  print(key, ' : ', value)
```

```
0  :  a
1  :  and
2  :  appears
3  :  as
4  :  by
5  :  caused
6  :  defoliation
7  :  description
8  :  disease
9  :  for
10  :  fungal
11  :  hemileia
12  :  is
13  :  it
14  :  leading
15  :  leaf
```

Embedding Question and Context

```
[ ] import pprint
    bow_corpus = [dictionary.doc2bow(text) for text in sentence_words]
    for sent, embedding in zip(sentences, bow_corpus):
      print(sent)
      print(embedding)
```

```
tea leaf rust
description for tea leaf rust tea leaf rust is a fungal disease caused by the pathogen hemileia
vastatrix it appears as orange to rustcolored powdery spores on the undersides of tea leaves leading
to yellowing and defoliation
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1),
medication for tea leaf rust fungicides like copperbased sprays copper oxychloride or systemic
fungicides propiconazole can be used to control tea leaf rust these should be applied according to
the manufacturers instructions and in consultation with agricultural experts
[(1, 1), (9, 1), (15, 2), (22, 2), (25, 2), (26, 1), (27, 2), (31, 1), (32, 1), (33, 1), (34, 2), (35, 1),
prevention for tea leaf rust to prevent tea leaf rust it is important to maintain good sanitation
practices in tea gardens such as removing fallen leaves and plant debris regular monitoring and early
detection of rustinfected plants can help in containing the spread additionally planting resistant tea
cultivars can provide some level of protection
[(1, 2), (3, 1), (9, 1), (12, 1), (13, 1), (15, 2), (16, 1), (17, 2), (22, 2), (25, 4), (26, 1), (27, 2), (
tea blister blight
description for tea blister blight tea blister blight is a fungal disease caused by exobasidium vexans it i
characterized by raised blisterlike lesions on tea leaves which turn white or pinkishgray as the disease
progresses
[(0, 1), (3, 1), (4, 2), (5, 1), (7, 1), (8, 2), (9, 1), (10, 1), (12, 2), (13, 1), (16, 1), (18, 1), (25,
medication for tea blister blight fungicides containing active ingredients like mancozeb chlorothalonil
or propiconazole can be used to control tea blister blight application should be done as per the
recommended dosage and schedule
[(1, 1), (3, 1), (9, 1), (25, 2), (26, 1), (27, 1), (34, 2), (35, 1), (37, 1), (41, 1), (44, 1), (46, 1), (
```

```
[ ] question = "Prevention for Tea Leaf Rust:?"
```

```
[ ] question = clean_sentence(question, stopwords=False)
    question_embedding = dictionary.doc2bow(question.split())
    print(question, "\n", question_embedding)
```

```
prevention for tea leaf rust
 [(9, 1), (15, 1), (22, 1), (25, 1), (75, 1)]
```

Module 4: Smart Medication and Prevention Suggestion System

Getting the cosine similarity

```
import sklearn
from sklearn.metrics.pairwise import cosine_similarity

def retrieveAndPrintFAQAnswer(question_embedding, sentence_embeddings, sentences):
  max_sim = -1
  index_sim = -1
  for index, embedding in enumerate(sentence_embeddings):
    sim = cosine_similarity(embedding, question_embedding)[0][0]
    print(index, sim, sentences[index])
    if sim > max_sim:
      max_sim = sim
      index_sim = index

  return index_sim
index = retrieveAndPrintFAQAnswer(question_embedding, bow_corpus, sentences)
```

```
0 0.11043152607484653 tea leaf rust
description for tea leaf rust tea leaf rust is a fungal disease caused by the pathogen hemileia
vastatrix it appears as orange to rustcolored powdery spores on the undersides of tea leaves leading
to yellowing and defoliation
1 0.7808688094430302 medication for tea leaf rust fungicides like copperbased sprays copper oxychloride or
fungicides propiconazole can be used to control tea leaf rust these should be applied according to
the manufacturers instructions and in consultation with agricultural experts
2 0.5432512781572743 prevention for tea leaf rust to prevent tea leaf rust it is important to maintain gooc
practices in tea gardens such as removing fallen leaves and plant debris regular monitoring and early
detection of rustinfected plants can help in containing the spread additionally planting resistant tea
cultivars can provide some level of protection
3 0.11043152607484653 tea blister blight
description for tea blister blight tea blister blight is a fungal disease caused by exobasidium vexans it i
characterized by raised blisterlike lesions on tea leaves which turn white or pinkishgray as the disease
progresses
4 0.7808688094430302 medication for tea blister blight fungicides containing active ingredients like mancoz
or propiconazole can be used to control tea blister blight application should be done as per the
recommended dosage and schedule
```

Inferencing

```
[ ] print("Question: ", question)
    print("Answer: ", sentences[index])
```

```
Question:  prevention for tea leaf rust
Answer:  medication for tea shot hole fungicides containing active ingredients li
tebuconazole can be used to manage tea shot hole
```

Note: answer is not accurate

Module 4: Smart Medication and Prevention Suggestion System

Load Word2Vec model

Google Pre-trained Model - GoogleNews-vectors-negative300

```
[ ] from gensim.models import KeyedVectors
    import gensim.downloader as api
    from gensim import models

    try:
      v2w_model = models.KeyedVectors.load_word2vec_format('./GoogleNews-vectors-negative3
      print("w2v Model Successfully loaded")
    except:
      v2w_model = api.load('word2vec-google-news-300')
      v2w_model.save("./w2vecmodel.mod")
      print("w2v Model Saved")
```

```
w2v Model Successfully loaded
```

Inferencing

```
[ ] print("Question: ", question)
    print("Answer: ", cleaned_sentences_with_stopwords[index])
```

```
Question:  prevention for tea leaf rust
Answer:  prevention for tea leaf rust to prevent tea leaf rust it is important to maintain good sanitation
practices in tea gardens such as removing fallen leaves and plant debris regular monitoring and early
detection of rustinfected plants can help in containing the spread additionally planting resistant tea
cultivars can provide some level of protection
```

Use Word2Vec hidden weights as word embedding and getting the answers cosine similarity values

```
[ ] sent_embeddings = []
    for sent in cleaned_sentences:
      sent_embeddings.append(getPhraseEmbedding(sent, v2w_model))

    question_embedding = getPhraseEmbedding(question, v2w_model)
    index = retrieveAndPrintFAQAnswer(question_embedding, sent_embeddings, cleaned_sentences_with_stopwords)
```

```
0 0.8344631757149804 tea leaf rust
description for tea leaf rust tea leaf rust is a fungal disease caused by the pathogen hemileia
vastatrix it appears as orange to rustcolored powdery spores on the undersides of tea leaves leading
to yellowing and defoliation
1 0.8025618335802304 medication for tea leaf rust fungicides like copperbased sprays copper oxychloride or systemic
fungicides propiconazole can be used to control tea leaf rust these should be applied according to
the manufacturers instructions and in consultation with agricultural experts
2 0.844491750413192 prevention for tea leaf rust to prevent tea leaf rust it is important to maintain good sanitation
practices in tea gardens such as removing fallen leaves and plant debris regular monitoring and early
detection of rustinfected plants can help in containing the spread additionally planting resistant tea
cultivars can provide some level of protection
3 0.6643287504774921 tea blister blight
description for tea blister blight tea blister blight is a fungal disease caused by exobasidium vexans it is
characterized by raised blisterlike lesions on tea leaves which turn white or pinkishgray as the disease
progresses
4 0.6401390551091775 medication for tea blister blight fungicides containing active ingredients like mancozeb chlorothalonil
or propiconazole can be used to control tea blister blight application should be done as per the
recommended dosage and schedule
5 0.7399693264846601 prevention for tea blister blight proper sanitation measures including the removal and destruction of
infected leaves are essential in preventing the spread of tea blister blight pruning of affected branches
and maintaining good air circulation in tea gardens can help reduce disease incidence choosing
resistant tea cultivars is also an effective preventive measure
6 0.6167938164775959 tea orange blotch
```

Module 4: Intelligent Disease Identification and Tea Domain Knowledge System

Load Bert model and tokenizer

```
[ ] import torch
    from transformers import BertForQuestionAnswering

    model = BertForQuestionAnswering.from_pretrained(r'gdrive/My Drive/Colab Notel

    from transformers import BertTokenizer

    tokenizer = BertTokenizer.from_pretrained(r'gdrive/My Drive/Colab Notebooks/r
```

Inferencing

```
def answer_question(question, answer_text):

    input_ids = tokenizer.encode(question, answer_text, max_length=512, truncation=True)

    print('Query has {:,} tokens.\n'.format(len(input_ids)))

    sep_index = input_ids.index(tokenizer.sep_token_id)

    num_seg_a = sep_index + 1

    num_seg_b = len(input_ids) - num_seg_a

    segment_ids = [0]*num_seg_a + [1]*num_seg_b

    assert len(segment_ids) == len(input_ids)

    start_scores, end_scores = model(torch.tensor([input_ids]), token_type_ids=torch.tensor([segment_ids]), return_dict=False)

    all_tokens = tokenizer.convert_ids_to_tokens(input_ids)

    print(' '.join(all_tokens[torch.argmax(start_scores) : torch.argmax(end_scores)+1]))
    print(f'score: {torch.max(start_scores)}')
    score = float(torch.max(start_scores))
    answer_start = torch.argmax(start_scores)
    answer_end = torch.argmax(end_scores)
    tokens = tokenizer.convert_ids_to_tokens(input_ids)
    answer = tokens[answer_start]

    for i in range(answer_start + 1, answer_end + 1):

        if tokens[i][0:2] == ' ':
            answer += tokens[i][2:]

        else:
            answer += ' ' + tokens[i]
    return answer, score
```

```
[ ] user_question = " I have observed a concerning disease on my farm that affects the upper leaf surface of the tea plants. It begins with the appearance of small, yellowish spots
```

```
[ ] answer_question(user_question, symptoms_txt)

    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncation strategy. So the returned list will alway
    Query has 512 tokens.

    tea leaf rust
    score: 4.301675796508789
    ('tea leaf rust', 4.301675796508789)
```

Evaluation

Evaluation

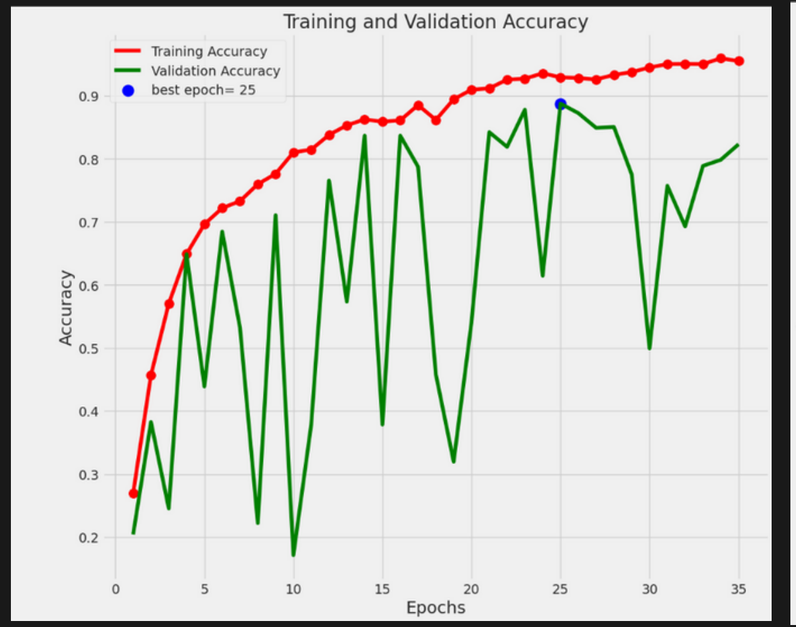
Module 1: Intelligent Leaf Differentiation Module

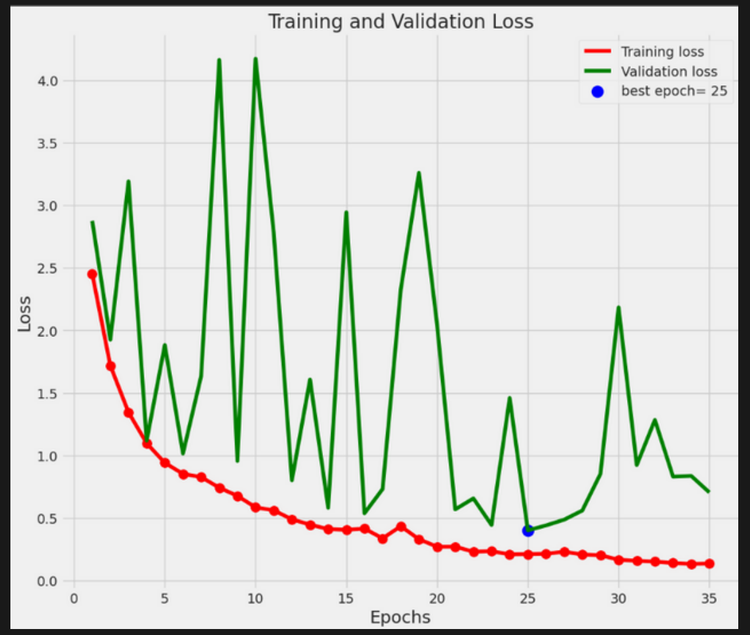
- To evaluate the custom CNN model, a model was created and trained using the transfer learning technique.

- For that, I took the resnet50 model and trained the last 8 layers to my dataset and added some dense layers to increase the accuracy level. Evaluation was done by comparing with this transfer learning model

- The training accuracy of the model is 95.5%. The testing accuracy is 82.5%.

Accuracy & Loss in final model





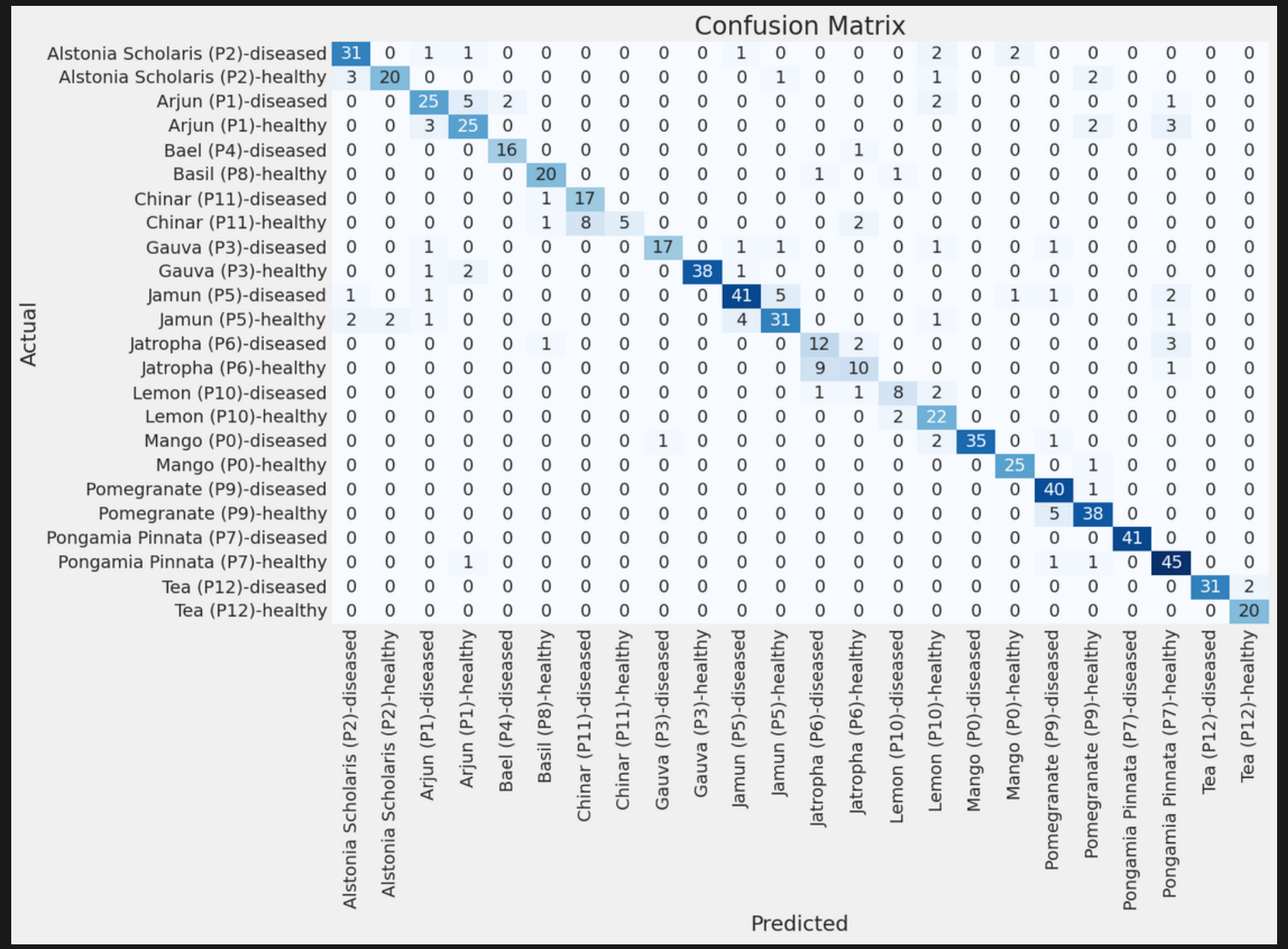
Implemented UI

Classification report (CNN)

Classification Report:

	precision	recall	f1-score	support
Alstonia Scholaris (P2)-diseased	0.8378	0.8158	0.8267	38
Alstonia Scholaris (P2)-healthy	0.9091	0.7407	0.8163	27
Arjun (P1)-diseased	0.7576	0.7143	0.7353	35
Arjun (P1)-healthy	0.7353	0.7576	0.7463	33
Bael (P4)-diseased	0.8889	0.9412	0.9143	17
Basil (P8)-healthy	0.8696	0.9091	0.8889	22
Chinar (P11)-diseased	0.6800	0.9444	0.7907	18
Chinar (P11)-healthy	1.0000	0.3125	0.4762	16
Gauva (P3)-diseased	0.9444	0.7727	0.8500	22
Gauva (P3)-healthy	1.0000	0.9048	0.9500	42
Jamun (P5)-diseased	0.8542	0.7885	0.8200	52
Jamun (P5)-healthy	0.8158	0.7381	0.7750	42
Jatropha (P6)-diseased	0.5217	0.6667	0.5854	18
Jatropha (P6)-healthy	0.6250	0.5000	0.5556	20
Lemon (P10)-diseased	0.7273	0.6667	0.6957	12
Lemon (P10)-healthy	0.6667	0.9167	0.7719	24
Mango (P0)-diseased	1.0000	0.8974	0.9459	39
Mango (P0)-healthy	0.8929	0.9615	0.9259	26
Pomegranate (P9)-diseased	0.8163	0.9756	0.8889	41
Pomegranate (P9)-healthy	0.8444	0.8837	0.8636	43
Pongamia Pinnata (P7)-diseased	1.0000	1.0000	1.0000	41
Pongamia Pinnata (P7)-healthy	0.8036	0.9375	0.8654	48
Tea (P12)-diseased	1.0000	0.9394	0.9688	33
Tea (P12)-healthy	0.9091	1.0000	0.9524	20
accuracy			0.8409	729
macro avg	0.8375	0.8202	0.8170	729
weighted avg	0.8511	0.8409	0.8384	729

Confusion matrix (CNN)

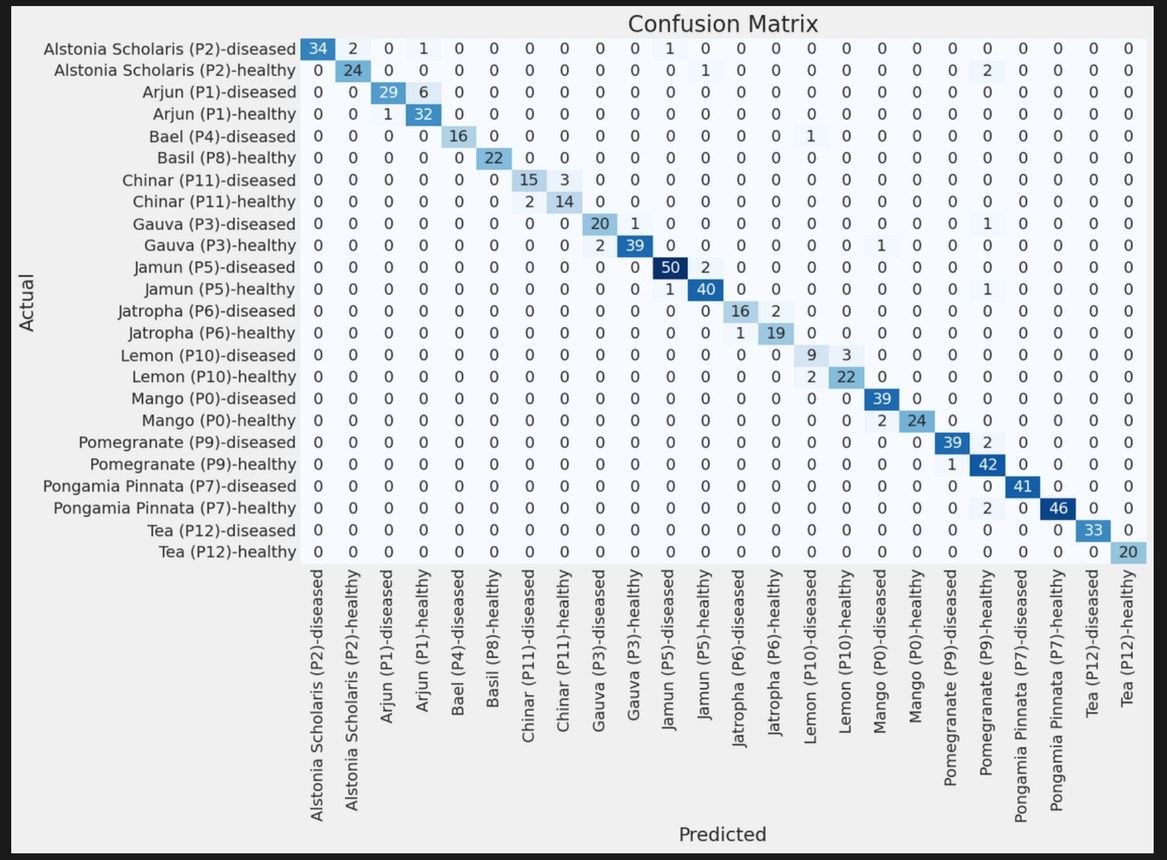


Classification report (ResNet50)

Classification Report:

	precision	recall	f1-score	support
Alstonia Scholaris (P2)-diseased	1.0000	0.8947	0.9444	38
Alstonia Scholaris (P2)-healthy	0.9231	0.8889	0.9057	27
Arjun (P1)-diseased	0.9667	0.8286	0.8923	35
Arjun (P1)-healthy	0.8205	0.9697	0.8889	33
Bael (P4)-diseased	1.0000	0.9412	0.9697	17
Basil (P8)-healthy	1.0000	1.0000	1.0000	22
Chinar (P11)-diseased	0.8824	0.8333	0.8571	18
Chinar (P11)-healthy	0.8235	0.8750	0.8485	16
Gauva (P3)-diseased	0.9091	0.9091	0.9091	22
Gauva (P3)-healthy	0.9750	0.9286	0.9512	42
Jamun (P5)-diseased	0.9615	0.9615	0.9615	52
Jamun (P5)-healthy	0.9302	0.9524	0.9412	42
Jatropha (P6)-diseased	0.9412	0.8889	0.9143	18
Jatropha (P6)-healthy	0.9048	0.9500	0.9268	20
Lemon (P10)-diseased	0.7500	0.7500	0.7500	12
Lemon (P10)-healthy	0.8800	0.9167	0.8980	24
Mango (P0)-diseased	0.9286	1.0000	0.9630	39
Mango (P0)-healthy	1.0000	0.9231	0.9600	26
Pomegranate (P9)-diseased	0.9750	0.9512	0.9630	41
Pomegranate (P9)-healthy	0.8400	0.9767	0.9032	43
Pongamia Pinnata (P7)-diseased	1.0000	1.0000	1.0000	41
Pongamia Pinnata (P7)-healthy	1.0000	0.9583	0.9787	48
Tea (P12)-diseased	1.0000	1.0000	1.0000	33
Tea (P12)-healthy	1.0000	1.0000	1.0000	20
accuracy			0.9396	729
macro avg	0.9338	0.9291	0.9303	729
weighted avg	0.9429	0.9396	0.9400	729

Confusion matrix (ResNet50)



Evaluation

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

Evaluation Result

```
{'recall': 0.92950696,
 'precision': 0.9492791,
 'specificity': 0.97556126,
 'negative predictive value': 0.9656639,
 'f1 score': 0.9392889,
 'fbeta': 0.68512297,
 'matthews correlation coefficient': 0.9099921,
 'equal error_rate': 0.042208664}
```

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

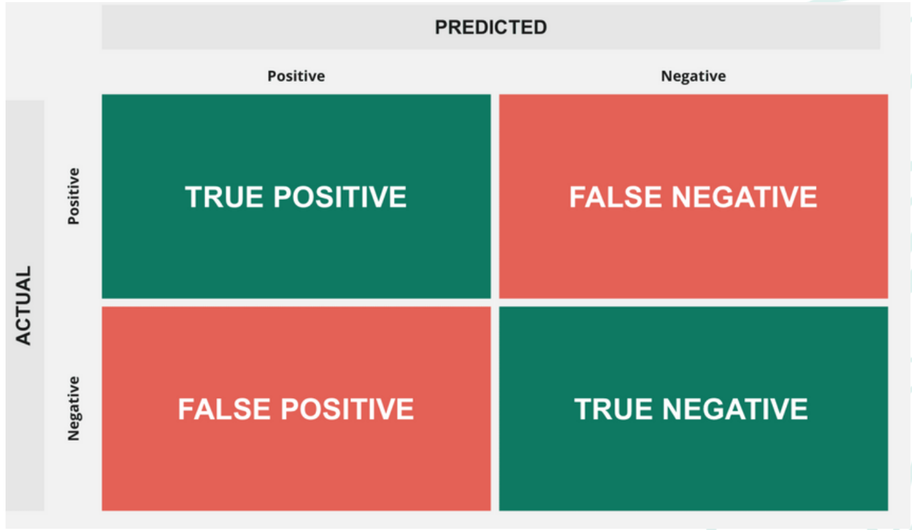
TP = True positive

TN = True negative

FP = False positive

FN = False negative

Evaluation Metrics Table



$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Evaluation (Cont.)

Module 2: Tea leaf preprocessing and segmentation using denoising autoencoder and semantic segmentation

Evaluation Result Summmary

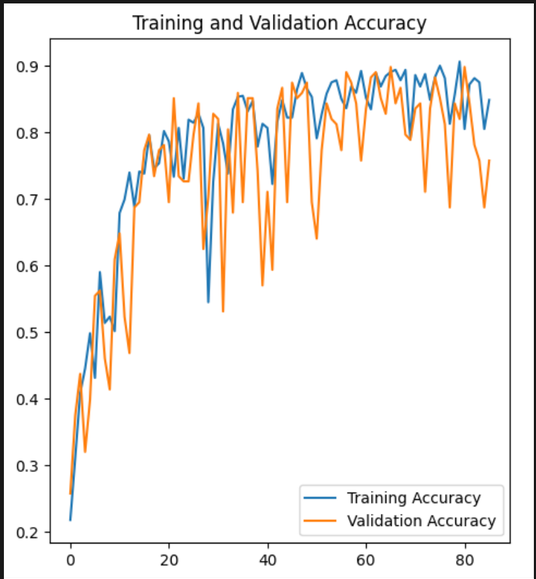
Recall	**0.9295**
Precision	**0.9492**
F1 Score	**0.9392**
Accuracy	**0.9330**

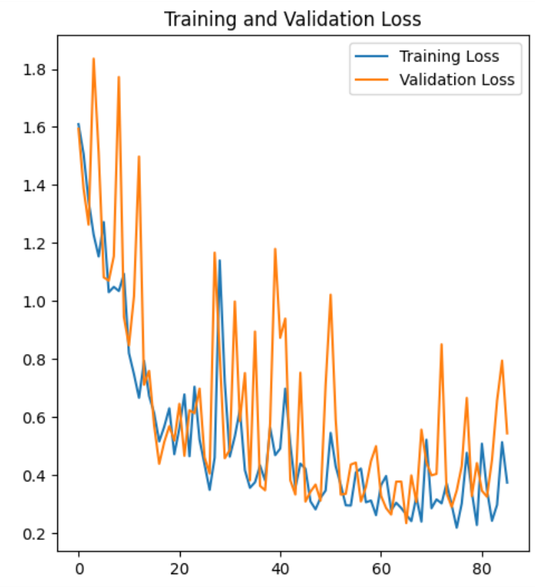
Evaluation (Cont.)

Module 3: Dual Classifier Module for Accurate Identification of Tea Leaf Diseases

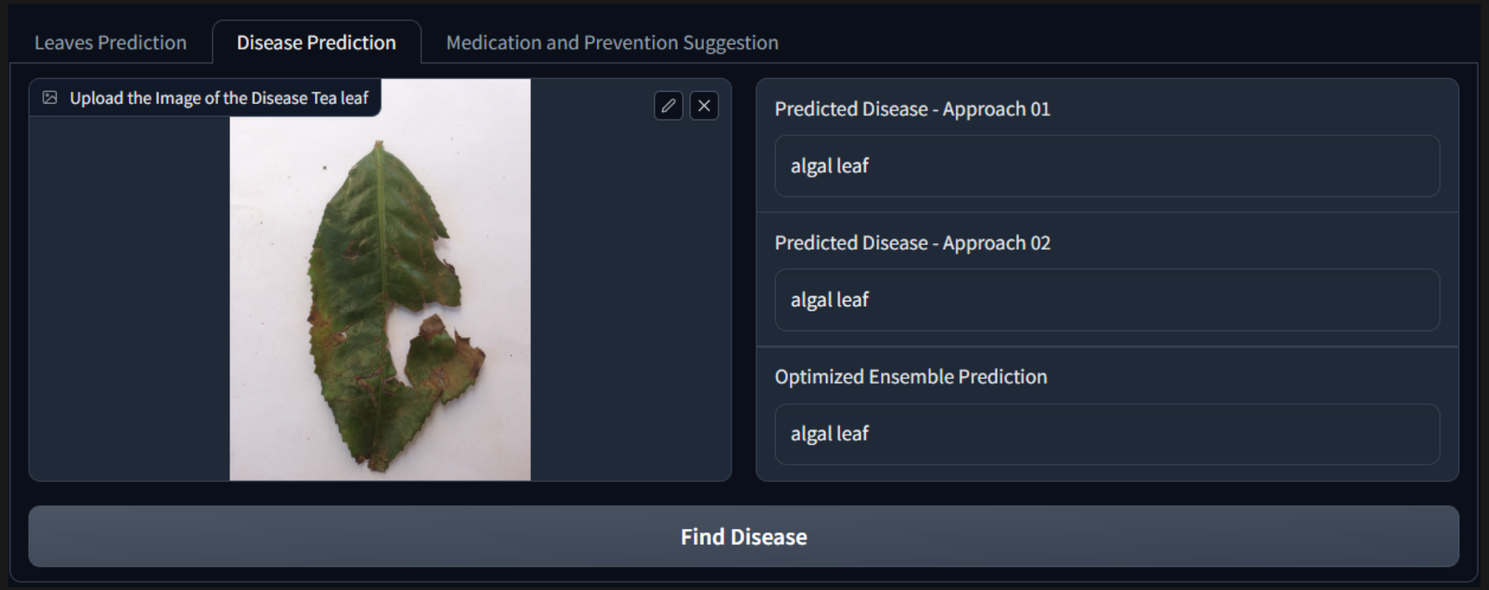
- I used xception model for evaluating custom built CNN model.
- The accuracy of the model is 98%..

Accuracy & Loss in custom CNN





Implemented UI



Classification report (Custom CNN)

	precision	recall	f1-score	support
algal leaf	0.50	0.50	0.50	26
brown blight	0.53	0.70	0.60	27
greyblight_new	0.55	0.55	0.55	22
red leaf spot	0.77	0.74	0.76	23
white spot	0.45	0.33	0.38	30
accuracy			0.55	128
macro avg	0.56	0.56	0.56	128
weighted avg	0.55	0.55	0.55	128

Confusion matrix (Custom CNN)

```
print(confusion_matrix(y_val_class, y_pred_class))
```

```
[[13  4  2  3  4]
 [ 2 19  1  1  4]
 [ 3  2 12  1  4]
 [ 3  2  1 17  0]
 [ 5  9  6  0 10]]
```

Decision algorithm evaluation

- To evaluate desicion algorithm, I used 5 sample images from each class. Then predicted results and actual results are in below list.

Data	Actual Result	Xception model result	Custom CNN result	Approach 1	Approach 2	Approach 3
				F1-score : Accuracy	F1-score : Accuracy	F1-score : Accuracy
				0.6 : 0.4	0.7 : 0.3	0.8 : 0.2
UNADJUSTEDNONRAW_thumb_1a	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf
UNADJUSTEDNONRAW_thumb_1b	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf
UNADJUSTEDNONRAW_thumb_2c	Algal Leaf	Algal Leaf	White spot	Algal Leaf	White spot	White spot
UNADJUSTEDNONRAW_thumb_1d	Algal Leaf	Brown blight	Algal Leaf	Brown blight	Brown blight	Brown blight
UNADJUSTEDNONRAW_thumb_1e	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf	Algal Leaf
UNADJUSTEDNONRAW_thumb_15e	Brown blight	Brown blight	White spot	Brown blight	White spot	Brown blight
UNADJUSTEDNONRAW_thumb_10b	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight
UNADJUSTEDNONRAW_thumb_11e	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight
UNADJUSTEDNONRAW_thumb_12a	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight
UNADJUSTEDNONRAW_thumb_13e	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight	Brown blight
20211225_233634	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight
20211228_134955	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight
20211228_135022	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight
20211228_135747	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight
20211228_135846	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight	Grey Blight
UNADJUSTEDNONRAW_thumb_1a0	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot
UNADJUSTEDNONRAW_thumb_1e1	Red leaf spot	Red leaf spot	White spot	Red leaf spot	Red leaf spot	White spot
UNADJUSTEDNONRAW_thumb_1b2	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot
UNADJUSTEDNONRAW_thumb_205	Red leaf spot	Red leaf spot	Algal Leaf	Red leaf spot	Red leaf spot	Algal Leaf
UNADJUSTEDNONRAW_thumb_1e6	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot
UNADJUSTEDNONRAW_thumb_204	White spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot	Red leaf spot
UNADJUSTEDNONRAW_thumb_7b	White spot	White spot	White spot	White spot	White spot	White spot
UNADJUSTEDNONRAW_thumb_74	White spot	White spot	White spot	White spot	White spot	White spot
UNADJUSTEDNONRAW_thumb_8f	White spot	White spot	Brown blight	White spot	Brown blight	Brown blight
UNADJUSTEDNONRAW_thumb_b6	White spot	White spot	White spot	White spot	White spot	White spot
Total	25	23	19	23	20	19
Accuracy		92%	76%	92%	80%	76%

- I used approach 1 as the best approach, which considers the weights as 0.6 for F1-score and 0.4 for accuracy

Evaluation (Cont.)

Module 4: Smart Medication and Prevention Suggestion System / Intelligent Disease Identification and Tea Domain Knowledge System

- To Identifying the best-performing model for this module I custom trained BERT and RoBERTa on the Stanford Question Answering Dataset (SQuAD) for question answering, and the evaluation results are as below.

BERT

```
{'EM': 95.32433930881538,
 'f1': 98.30690739601266,
 'top_n_accuracy': 99.55645906486787,
 'top_n': 4,
 'EM_text_answer': 95.32433930881538,
 'f1_text_answer': 98.30690739601266,
 'top_n_accuracy_text_answer': 99.55645906486787,
 'top_n_EM_text_answer': 95.82332286083903,
 'top_n_f1_text_answer': 98.49288499379847,
 'Total_text_answer': 5411,
 'EM_no_answer': 0,
 'f1_no_answer': nan,
 'top_n_accuracy_no_answer': nan,
 'Total_no_answer': 0}
```

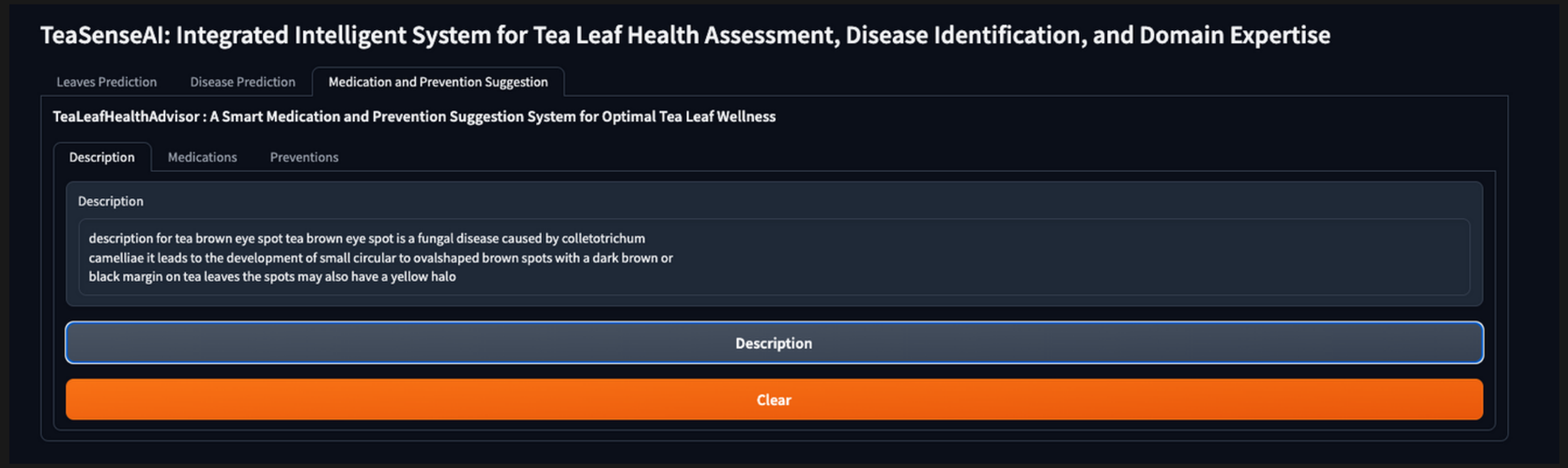
RoBERTa

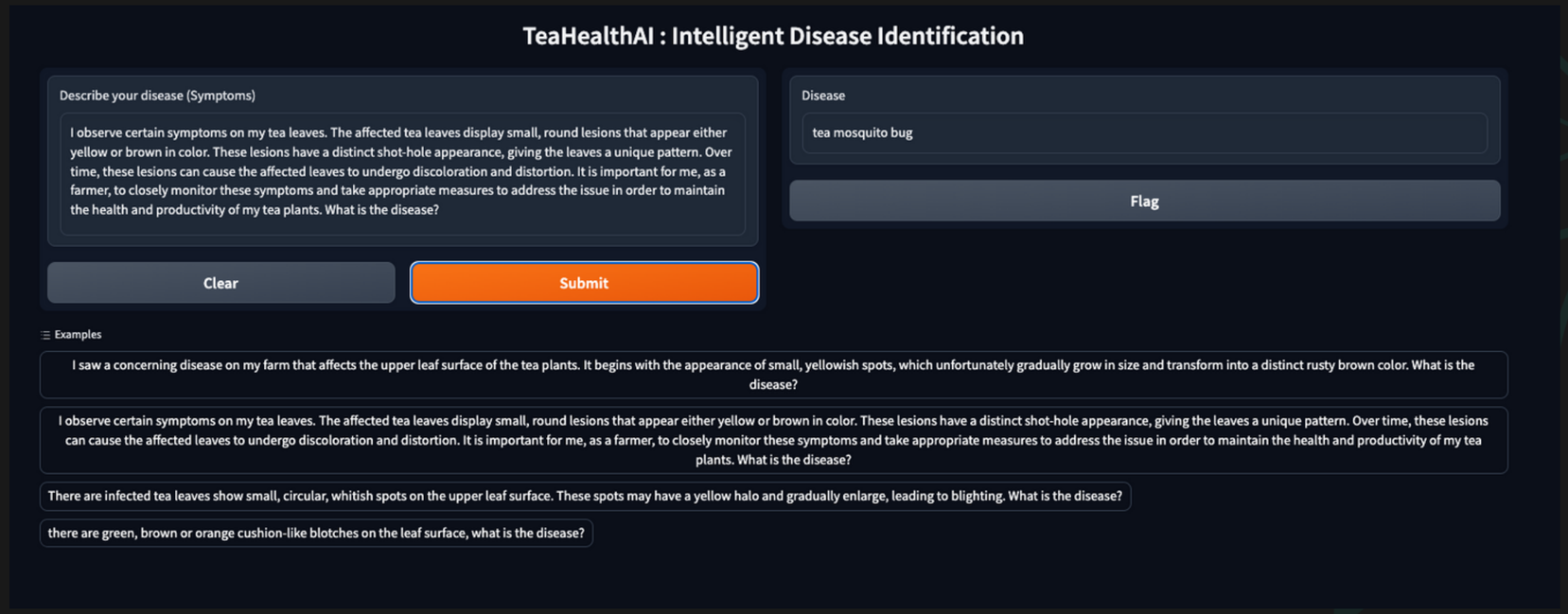
```
{'EM': 81.89730200174064,
 'f1': 84.24647292136972,
 'top_n_accuracy': 99.40818102697997,
 'top_n': 4,
 'EM_text_answer': 84.07773610637572,
 'f1_text_answer': 88.67916363220903,
 'top_n_accuracy_text_answer': 98.84077736106376,
 'top_n_EM_text_answer': 92.90828503239005,
 'top_n_f1_text_answer': 97.18607875349937,
 'Total_text_answer': 2933,
 'EM_no_answer': 79.62304409672831,
 'f1_no_answer': 79.62304409672831,
 'top_n_accuracy_no_answer': 100.0,
 'Total_no_answer': 2812}
```

- Also I wrote a custom evaluation function for evaluating these models by giving them a context, questions and answer arrays. And here are some evaluations sets and results

```
[ ] %%time

    context = """ Queen are a British rock band formed in London in 1970. Their classic line-up was
                  Freddie Mercury (lead vocals, piano), Brian May (guitar, vocals), Roger Taylor
                  (drums, vocals) and John Deacon (bass). Their earliest works were influenced by
                  progressive rock, hard rock and heavy metal, but the band gradually ventured into
                  more conventional and radio-friendly works by incorporating further styles, such as
                  arena rock and pop rock. """

    queries = ["When was Queen found?",
               "Who were the classic members of Queen band?",
               "What kind of band they are?"]

    answers = ["1970",
               "Freddie Mercury, Brian May, Roger Taylor and John Deacon",
               "rock"]


    for q,a in zip(queries, answers):
        give_an_answer(context, q, a)

    Question : When was Queen found?
    Prediction : 1970
    True Answer : 1970
    EM : 1
    F1 : 1.0


    Question : Who were the classic members of Queen band?
    Prediction : freddie mercury ( lead vocals , piano ) , brian may ( guitar , vocals ) , roger taylor ( drums , vocals ) and john deacon ( bass )
    True Answer : Freddie Mercury, Brian May, Roger Taylor and John Deacon
    EM : 0
    F1 : 0.6923076923076924


    Question : What kind of band they are?
    Prediction : british rock
    True Answer : rock
    EM : 0
    F1 : 0.6666666666666666


    CPU times: user 5 s, sys: 25.3 ms, total: 5.03 s
    Wall time: 7.19 s
```

Implemented UI





Further work

- Further enhancing the accuracy and robustness of the classification models by training the models for diverse datasets.
- Exploring ensemble methods, such as model averaging or stacking, to improve the overall accuracy and generalization capability.
- Create an intelligent system that generates customized disease remedies based on user input.
- Integrating additional sources of domain knowledge.
- Optimize the F1 weight and Accuracy weight on decision algorithms.

Demonstration

Thank you!