Email ID: SI2022IBM01612@smartinternz.com

Rajesh Kumar

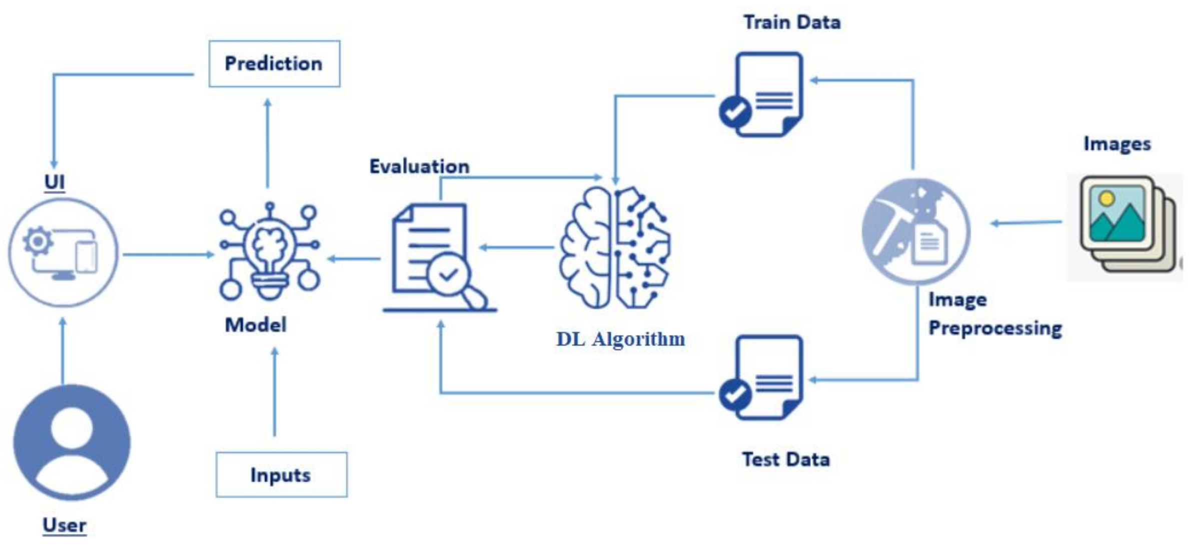
https://github.com/SI2022IBM01609/Project-a_thon

Fertilizers Recommendation System For Disease Prediction

Agriculture is the most important sector in today's life. Most plants are affected by a wide variety of bacterial and fungal diseases. Diseases on plants placed a major constraint on the production and a major threat to food security. Hence, early and accurate identification of plant diseases is essential to ensure high quantity and best quality. In recent years, the number of diseases on plants and the degree of harm caused has increased due to the variation in pathogen varieties, changes in cultivation methods, and inadequate plant protection techniques.

An automated system is introduced to identify different diseases on plants by checking the symptoms shown on the leaves of the plant. Deep learning techniques are used to identify the diseases and suggest the precautions that can be taken for those diseases.

Technical Architecture



Project Objectives

- By the end of this project you'll understand:

- Preprocess the images.
- Applying the CNN algorithm to the dataset.
- How deep neural networks detect the disease.
- find the accuracy of the model.
- build web applications using the Flask framework.

Project Flow

A web Application is built in which Farmers interact with the portal build Interacts with the user interface to upload images of diseased leaf Our model built analyses the Disease and suggests the farmer with fertilizers are to be used

To accomplish the above task below activities and tasks were performed

- Download the dataset.
- Classify the dataset into train and test sets.
- Add the neural network layers.
- Load the trained images and fit the model.
- Test the model.
- Save the model and its dependencies.
- Build a Web application using a flask that integrates with the model built.

Prerequisites

To complete this project the following software and packages are required

Anaconda Navigator :

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning-related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupiter notebook and spyder

To build Deep learning models following packages are required

Tensor flow: TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML powered applications.

Keras : Keras leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features:

Consistent,   GPU. Highly scalability of computation. Flask: Web framework used for building

Project Structure

Project structure is as follows

- The dataset folder contains two folders for the fruit and vegetable dataset which again contains a test and train folder, each of them have images of different diseases.
- The Flask folder has all the files necessary to build the flask application.
- the static folder has the images, style sheets, and scripts that are needed in building the web page.
- templates folder has the HTML pages.
- uploads folder has the uploads made by the user.
- app.py is the python script for server-side computing.
- .h5 files are the model files that are to be saved after model building.
- precautions excel files contain the precautions for all kinds of diseases.
- Fruit-Training.ipynb, Vegetable-Training, and Plant-Disease-Testing.ipynb are the training and testing notebooks.
- IBM folder contains IBM deployment files.

Data Collection

Download of the dataset

Train and Test folders are created with each folder having subfolders with leaf images of different plant diseases. You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository, etc. From the dataset link you can download datasets that can be used for training. Two datasets will be used, we will be creating two models one to detect vegetable leaf diseases like tomato, potato, and pepper plants and the second model would be for fruits diseases like corn, peach, and apple

Image Preprocessing

Now that we have all the data collected, let us use this data to train the model . before training the model you have to preprocess the images and then feed them on to the model for training. we make use of Keras ImageDataGenerator class for image preprocessing.
For more info about image preprocessing please click on the below link
data Augmentation

Image Pre-processing includes the following main tasks

- Import ImageDataGenerator Library.
- Configure ImageDataGenerator Class.
- Applying ImageDataGenerator functionality to the trainset and test set.

The ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.

Preprocess The Images

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

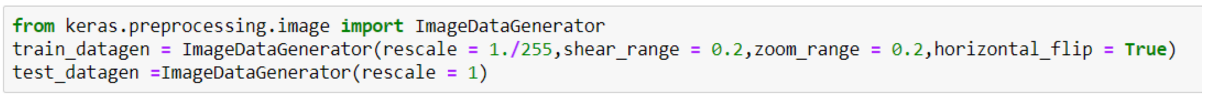
The first step is usually importing the libraries that will be needed in the program.

Import Keras library from that library import the ImageDataGenerator Library to your Python script:

import the ImageDataGenerator class from Keras

Import ImageDataGenerator Library and Configure it

ImageDataGenerator class is used to load the images with different modifications like considering the zoomed image, flipping the image and rescaling the images to range of 0 and 1.



```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,shear_range = 0.2,zoom_range = 0.2,horizontal_flip = True)
test_datagen =ImageDataGenerator(rescale = 1)
```

There are five main types of data augmentation techniques for image data; specifically:

- Image shifts via the width_shift_range and height_shift_range arguments.
- The image flips via the horizontal_flip and vertical_flip arguments.
- The image rotates via the rotation_range argument
- Image brightness via the brightness_range argument.
- The image zooms via the zoom_range argument.

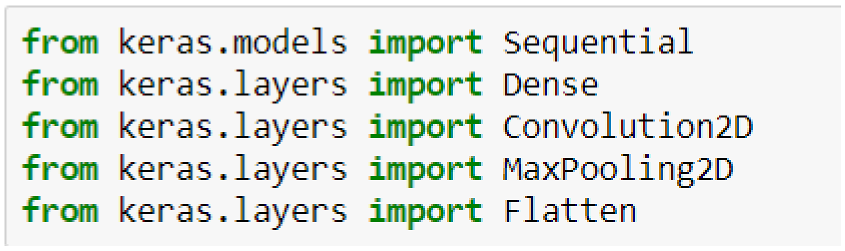
An instance of the ImageDataGenerator class can be constructed for train and test.

Apply ImageDataGenerator functionality to Train and Test set

Specify the path of both the folders in the flow_from_directory method. We are importing the images in 128*128 pixels.

Import The Libraries

Import the libraries that are required to initialize the neural network layer, and create and add different layers to the neural network model.



```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
```

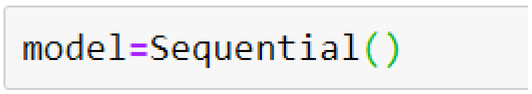
Initializing The Model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add () method.

Initialize the neural network layer by creating a reference/object to the Sequential class.



```
model=Sequential()
```

ADD CNNLayers

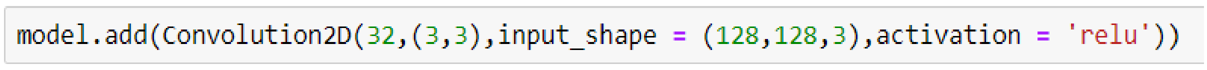
We will be adding three layers for CNN

- Convolution layer
- Pooling layer
- Flattening layer

Add Convolution Layer

The first layer of the neural network model, the convolution layer will be added. To create a convolution layer, Convolution2D class is used. It takes a number of feature detectors, feature detector size, expected input shape of the image, and activation function as arguments. This layer applies feature detectors on the input image and returns a feature map (features from the image).

Activation Function: These are the functions that help us to decide if we need to activate the node or not. These functions introduce non-linearity in the networks.

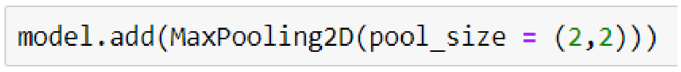


```
model.add(Convolution2D(32,(3,3),input_shape = (128,128,3),activation = 'relu'))
```

Add the pooling layer

Max Pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after the max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

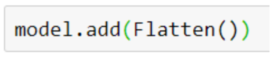
After the convolution layer, a pooling layer is added. Max pooling layer can be added using MaxPooling2D class. It takes the pool size as a parameter. Efficient size of the pooling matrix is (2,2). It returns the pooled feature maps.



```
model.add(MaxPooling2D(pool_size = (2,2)))
```

Add the flatten layer

The flatten layer is used to convert n-dimensional arrays to 1-dimensional arrays. This 1D array will be given as input to ANN layers.



```
model.add(Flatten())
```

Add Dense Layers

Dense Layers are added

Dense layers

The name suggests that layers are fully connected (dense) by the neurons in a network layer. Each neuron in a layer receives input from all the neurons present in the previous layer. Dense is used to add the layers.

Adding Hidden layers

This step is to add a dense layer (hidden layer). We flatten the feature map and convert it into a vector or single dimensional array in the Flatten layer. This vector array is fed it as an input to the neural network and applies an activation function, such as sigmoid or other, and returns the output.

- **init** is the weight initialization; function which sets all the weights and biases of a network to values suitable as a starting point for training.
- units/ output_dim, denotes the number of neurons in the hidden layer.

- The activation function basically decides to deactivate neurons or activate them to get the desired output. It also performs a nonlinear transformation on the input to get better results on a complex neural network.
- In this project we have added two hidden layers. The 1st hidden layer with 300 neurons and 2nd hidden layer with 150 neurons.

Adding the output layer

This step is to add a dense layer (output layer) where you will be specifying the number of classes your dependent variable has, activation function, and weight initializer as the arguments. We use the add () method to add dense layers. the output dimensions here is 9 for vegetable and 6 for the fruit.

Train And Save The Model

Compile the model

After adding all the required layers, the model is to be compiled. For this step, loss function, optimizer and metrics for evaluation can be passed as arguments.

```
model.compile(loss = 'categorical_crossentropy',optimizer = "adam",metrics = ["accuracy"])
```

Fit and save the model

Fit the neural network model with the train and test set, number of epochs and validation steps. Steps per epoch is determined by number of training images//batch size, for validation steps number of validation images//batch size.

```
model.fit_generator(x_train, steps_per_epoch = 168,epochs = 3,validation_data = x_test,validation_steps = 52)
```

Accuracy, Loss: Loss value implies how poorly or well a model behaves after each iteration of optimization. An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage.

The weights are to be saved for future use. The weights are saved in as .h5 file using save().

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

model.summary() can be used to see all parameters and shapes in each layer in our models.

Model Building For Vegetable Disease Prediction

Created an other jupyter notebook file as vegetable_training. The same steps followed for the fruit disease prediction model are to be followed to train the tomato, potato, and pepper diseases.

Train And Save The Model

ImageDataGenerator class is used to load the images with different modifications like considering the zoomed image, flipping the image and rescaling the images to range of 0 and 1.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen =ImageDataGenerator(rescale = 1)
```

Apply ImageDataGenerator functionality to Train and Test set

Specify the path of both the folders in the flow_from_directory method. Images in 128*128 pixels imported.

```
x_train = train_datagen.flow_from_directory('Veg-dataset/train_set',
                                            target_size = (128,128),
                                            batch_size = 16,
                                            class_mode = 'categorical')
```

```
Found 11386 images belonging to 9 classes.
```

```
x_test = test_datagen.flow_from_directory('Veg-dataset/test_set',
                                          target_size = (128,128),
                                          batch_size = 16,
                                          class_mode = 'categorical')
```

```
Found 3416 images belonging to 9 classes.
```

Import the required model building libraries

Import the libraries that are required to initialize the neural network layer, create and add different layers to the neural network model.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
```

Initialize the model

Initialize the neural network layer by creating a reference/object to the Sequential class.

```
model=Sequential()
```

Add the convolution layer

In the first layer of the neural network model, the convolution layer will be added. To create a convolution layer, the Convolution2D class is used. It takes a number of feature detectors, features detector size, expected input shape of the image, and activation function as arguments. This layer applies feature detectors on the input image and returns a feature map (features from the image).

Activation Function: These are the functions that help us to decide if we need to activate the node or not. These functions introduce non-linearity in the networks.

```
model.add(Convolution2D(32,(3,3),input_shape = (128,128,3),activation = 'relu'))
```

Add the pooling layer

Max Pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

After the convolution layer, a pooling layer is added. Max pooling layer can be added using MaxPooling2D class. It takes the pool size as a parameter. Efficient size of the pooling matrix is (2,2). It returns the pooled feature maps.

```
model.add(MaxPooling2D(pool_size = (2,2)))
```

Add the flatten layer

The flatten layer is used to convert n-dimensional arrays to 1-dimensional arrays. This 1D array will be given as input to ANN layers.

```
model.add(Flatten())
```

Adding the dense layers

Three dense layers are added which usually take a number of units/neurons. Specifying the activation function, kind of weight initialization is optional.

```
model.add(Dense(units = 300, init ='uniform', activation ='relu'))
```

```
model.add(Dense(units = 150, init ='uniform', activation ='relu'))
```

```
model.add(Dense(units = 75, init ='uniform', activation ='relu'))
```

```
model.add(Dense(output_dim = 9,activation = 'softmax',init ='uniform'))
```

Note: Any number of convolution, pooling, and dense layers can be added according to the data.

Compile the model

After adding all the required layers, the model is to be compiled. For this step, loss function, optimizer and metrics for evaluation can be passed as arguments.

```
model.compile(loss = 'categorical_crossentropy',optimizer = "adam",metrics = ["accuracy"])
```

Fit and save the model

Fit the neural network model with the train and test set, number of epochs and validation steps. Steps per epoch is determined by number of training images//batch size, for validation steps number of validation images//batch size.

```
model.fit_generator(x_train, steps_per_epoch = 89,
                    epochs = 20,
                    validation_data = x_test,
                    validation_steps = 27)
```

Accuracy, Loss: Loss value implies how poorly or well a model behaves after each iteration of optimization. An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage.

The weights are to be saved for future use. The weights are saved in as .h5 file using save().

```
model.save('vegetable.h5')
```

model.summary() can be used to see all parameters and shapes in each layer in our models.

Test Both The Models

Now that we have trained both the models' let's test both the models by loading the saved models. let's create another notebook for testing

Test The Model

The model is to be tested with different images to know if it is working correctly.

Import the packages and load the saved model

Import the required libraries

```
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
```

Initially, we will be loading the fruit model. You can test it with the vegetable model in a similar way.

```
model = load_model("fruit.h5")
```

Load the test image, pre-process it and predict

Pre-processing the image includes converting the image to array and resizing according to the model. Give the pre-processed image to the model to know to which class your model belongs to.

```
img = image.load_img('apple_healthy.JPG',target_size = (128,128))
```

```
x  = image.img_to_array(img)
x = np.expand_dims(x,axis = 0)
```

```
pred = model.predict_classes(x)
```

```
pred
```

```
[1]
```

The predicted class is 1

Project creation and Train The Model On IBM

login to cloud.ibm.com

resource list -> services & Software -> Watson-studio

Launch next window "Launch in IBM cloud pack for data"

Click nevigation menu left top -- three lines.

Project -> create project.

create an empty project

Click assets -> New Assets -> Code editors - Jupyter notebook

upload the jupyter notbook

click on the 01 in left top and upload the data.zip file

insert the streaming bode code at the top.

run the code and unzip the dataset file with set of code

install

!pip install watson-machine-learning-client [at the ibm cloud]

pip install ibm_watson_machine_learning [at the local system] make connection with cloud.ibm.com with the APIkeys.

```
from ibm_watson_machine_learning import APIClient

wml_credentials={

    "url":"https://us-south.ml.cloud.ibm.com",

    "apikey":"I3Fu79Y8bUV7V4Qe4us0JRhA3h9RA5zDKQhbMOybQYOL"

}

client = APIClient(wml_credentials)
```

Deployment of the Model On IBM

Deployment of model -> create new deployment space

Name(veg-fruit-deploy), Storage of service, Machine learning service

define and call function guid_space_name

match the space uid for the deployment.

```
space_uid = guid_space_name(client,'animal_model_dep')

print(space_uid)
```

set the default space uid

```
client.set.default_space(space_uid)
```

it has many packages installed -

```
client.software_specifications.list()
```

verify the ID of software package in use.

```
software_space_uid=client.software_specifications.get_uid_by_name('tensorflow_rt22.1-py3.9')
```

convert the model to .tar file

```
!tar -zcvf veg-pathon-model.tgz veg-pathon.h5
```

store the model to download and use it in the laptop

```
model_details=client.repository.store_model(model='veg-pathon-model.tgz',

meta_props={

client.repository.ModelMetaNames.NAME:"Veg-pathon",

client.repository.ModelMetaNames.TYPE:'tensorflow_2.7',

client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_space_uid

})
```

Verify the model id of stored model

```
model_id=client.repository.get_model_id(model_details)

model_id
```

download the model

```
client.repository.download(model_id, 'veg-pathon-model.tar.gz')
```

Application Building

After the model is built, we will be integrating it into a web application so that normal users can also use it. The new users need to initially register in the portal. After registration users can log in to browse the images to detect the disease.

- HTML pages - front end
- Python script - Server-side script

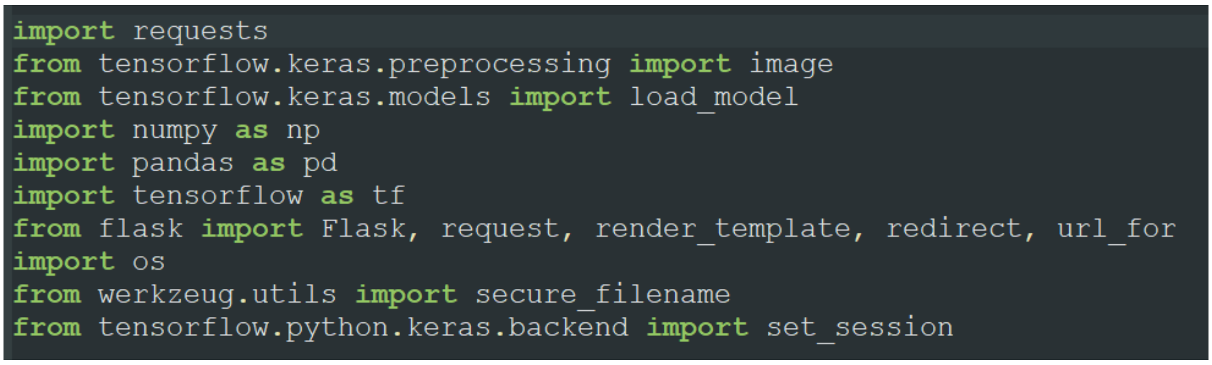
Python script is created using spyder IDE

Build Python Code

After the model is built, we will be integrating it into a web application so that normal users can also use it. The user needs to browse the images to detect the disease.

Activity 1: Build a flask application

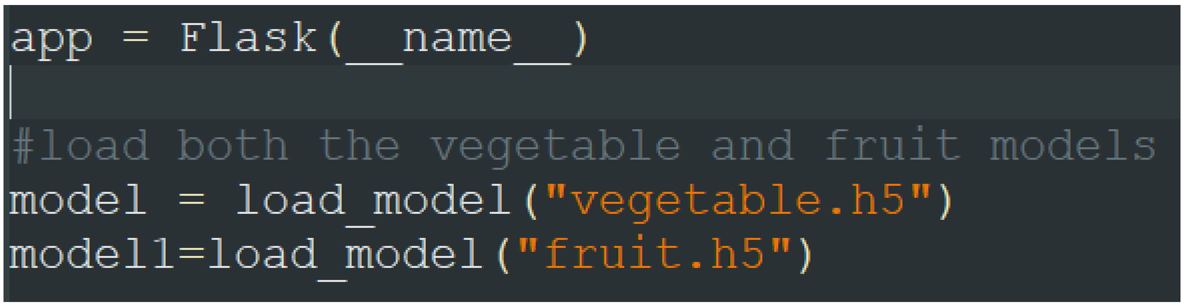
Step 1: **Load the required packages**



```python
import requests
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import numpy as np
import pandas as pd
import tensorflow as tf
from flask import Flask, request, render_template, redirect, url_for
import os
from werkzeug.utils import secure_filename
from tensorflow.python.keras.backend import set_session
```

Step 2: **Initialize the flask app and load the model**

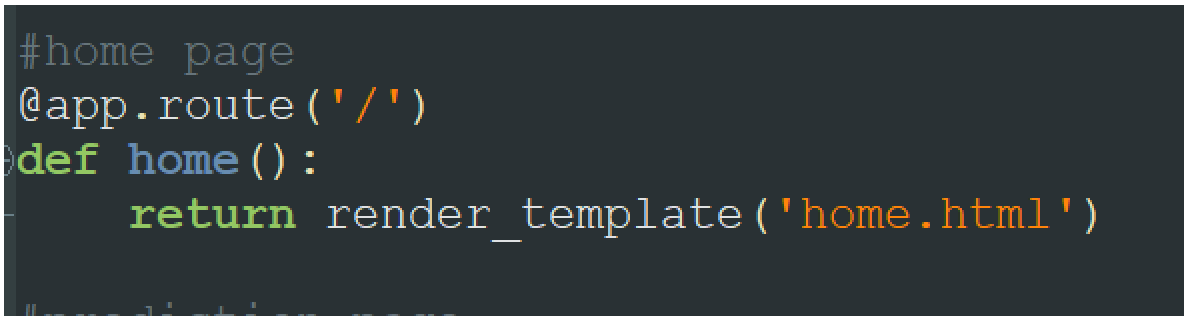
An instance of Flask is created and the model is loaded using load_model from Keras.



```python
app = Flask(__name__)

#load both the vegetable and fruit models
model = load_model("vegetable.h5")
model1=load_model("fruit.h5")
```

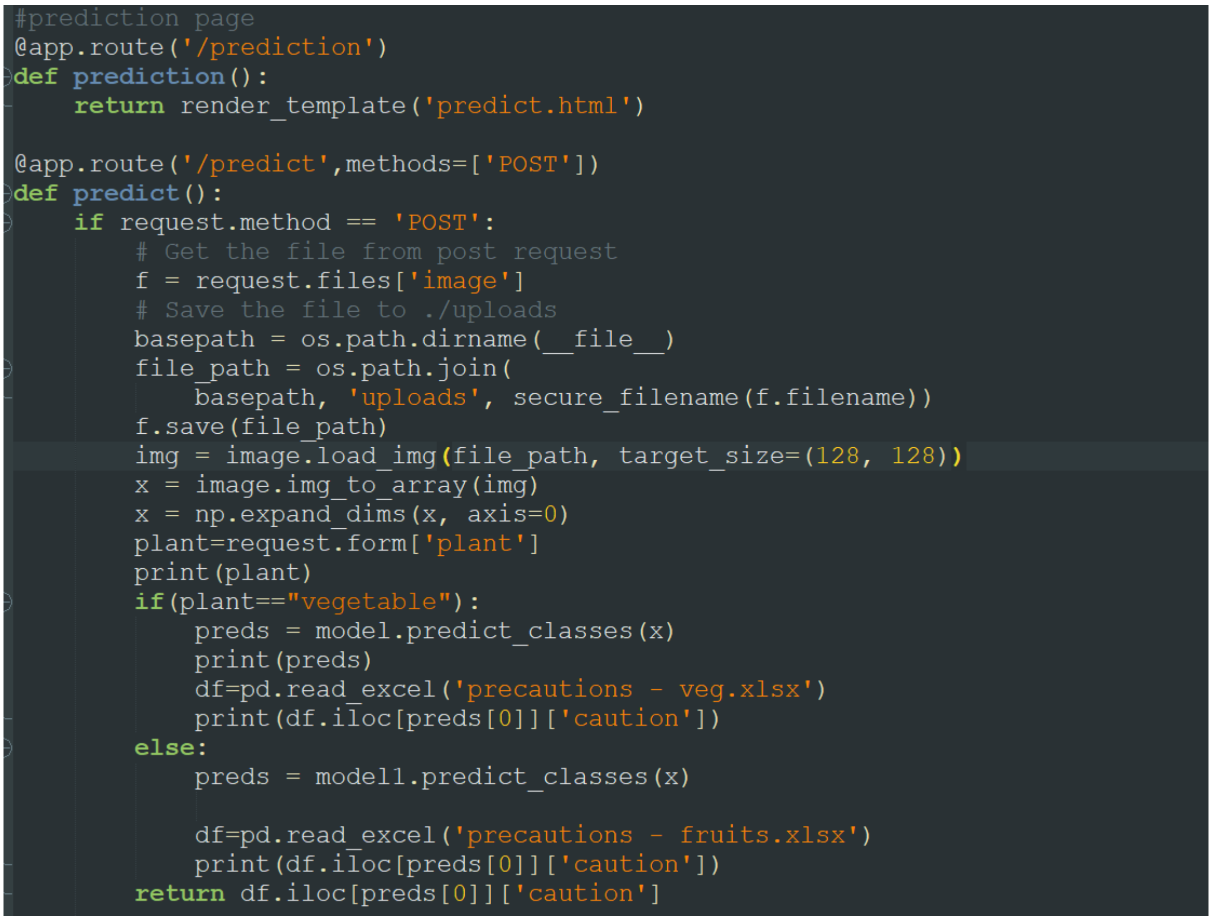
Step 3: **Configure the home page**



```python
#home page
@app.route('/')
def home():
    return render_template('home.html')
```

Step 4: **Pre-process the frame and run**

Pre-process the captured frame and give it to the model for prediction. Based on the prediction the output text is generated and sent to the HTML to display. We will be loading the precautions for fruits and vegetables excel file to get the precautions based on the output and return it to the HTML Page.

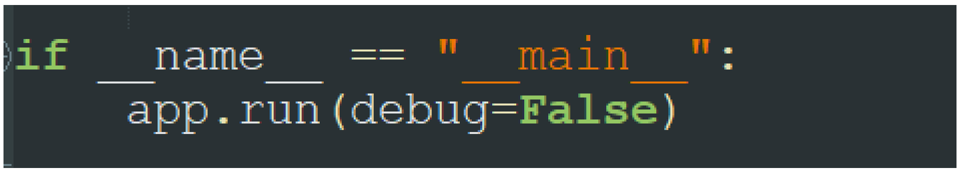


```python
#prediction page
@app.route('/prediction')
def prediction():
    return render_template('predict.html')

@app.route('/predict',methods=['POST'])
def predict():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']
        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        plant=request.form['plant']
        print(plant)
        if(plant=="vegetable"):
            preds = model.predict_classes(x)
            print(preds)
            df=pd.read_excel('precautions - veg.xlsx')
            print(df.iloc[preds[0]]['caution'])
        else:
            preds = model1.predict_classes(x)

            df=pd.read_excel('precautions - fruits.xlsx')
            print(df.iloc[preds[0]]['caution'])
        return df.iloc[preds[0]]['caution']
```

Run the flask application using the run method. By default, the flask runs on 5000 port. If the port is to be changed, an argument can be passed and the port can be modified.

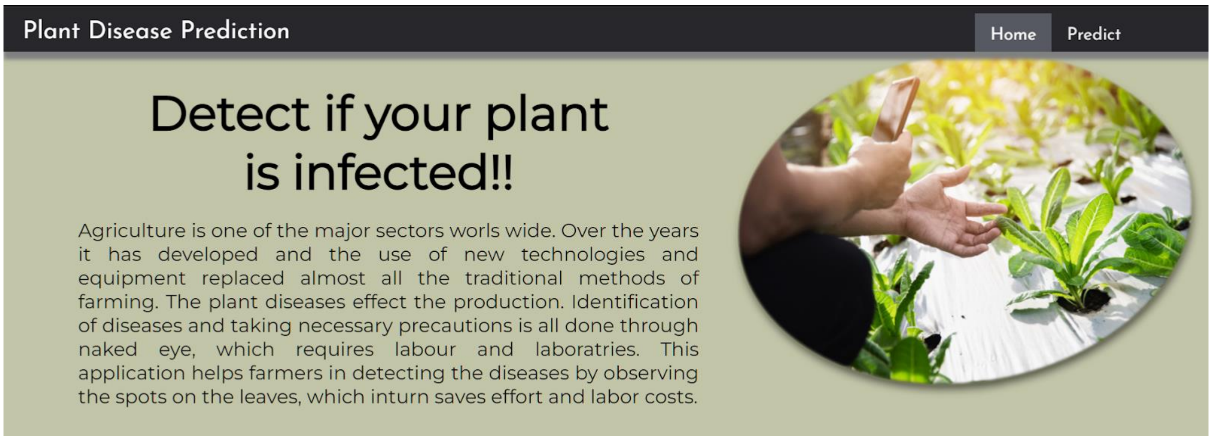


```python
if __name__ == "__main__":
    app.run(debug=False)
```

Build HTML Pages

Build the UI where a home page will have details about the application, a prediction page where a user is allowed to browse an image and get the predictions.

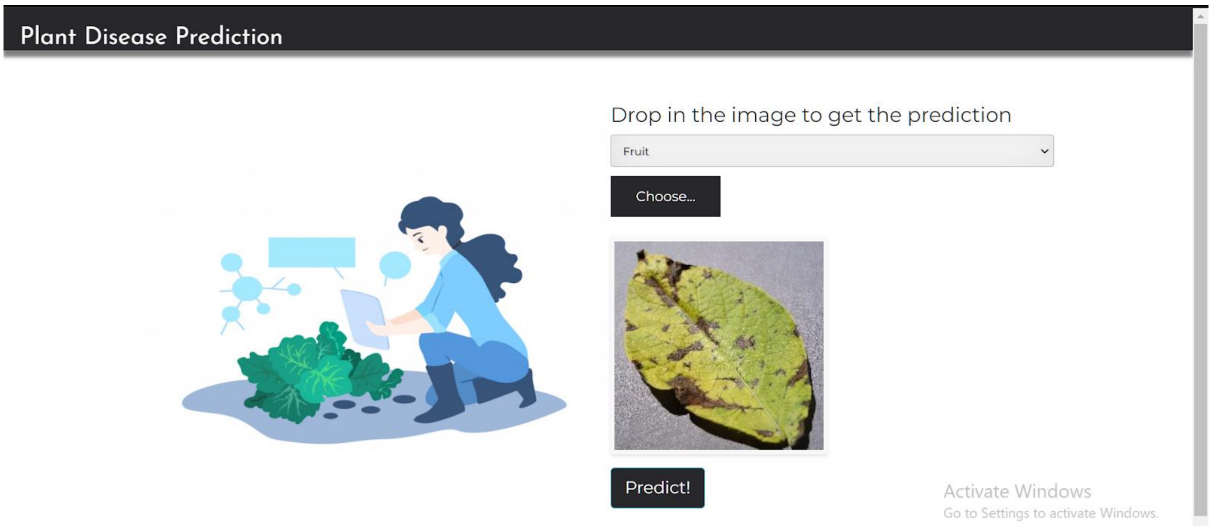
The home page looks like this.


Plant Disease Prediction

Detect if your plant is infected!!

Agriculture is one of the major sectors worls wide. Over the years it has developed and the use of new technologies and equipment replaced almost all the traditional methods of farming. The plant diseases effect the production. Identification of diseases and taking necessary precautions is all done through naked eye, which requires labour and laboratries. This application helps farmers in detecting the diseases by observing the spots on the leaves, which inturn saves effort and labor costs.

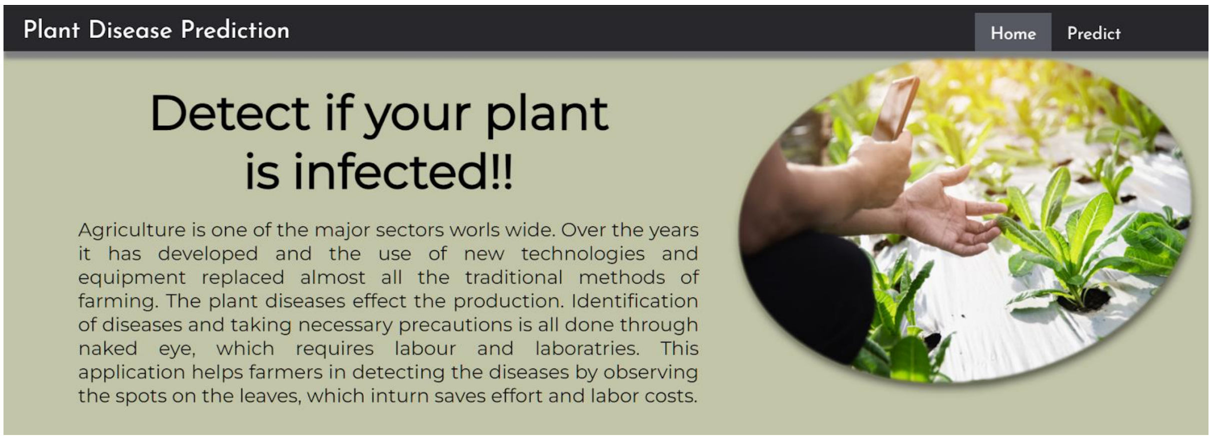
After clicking on predict button, you will be redirected to the prediction page where you can browse the images.


Plant Disease Prediction

Drop in the image to get the prediction

Step 2: Open the browser and navigate to localhost:5000 to check your application

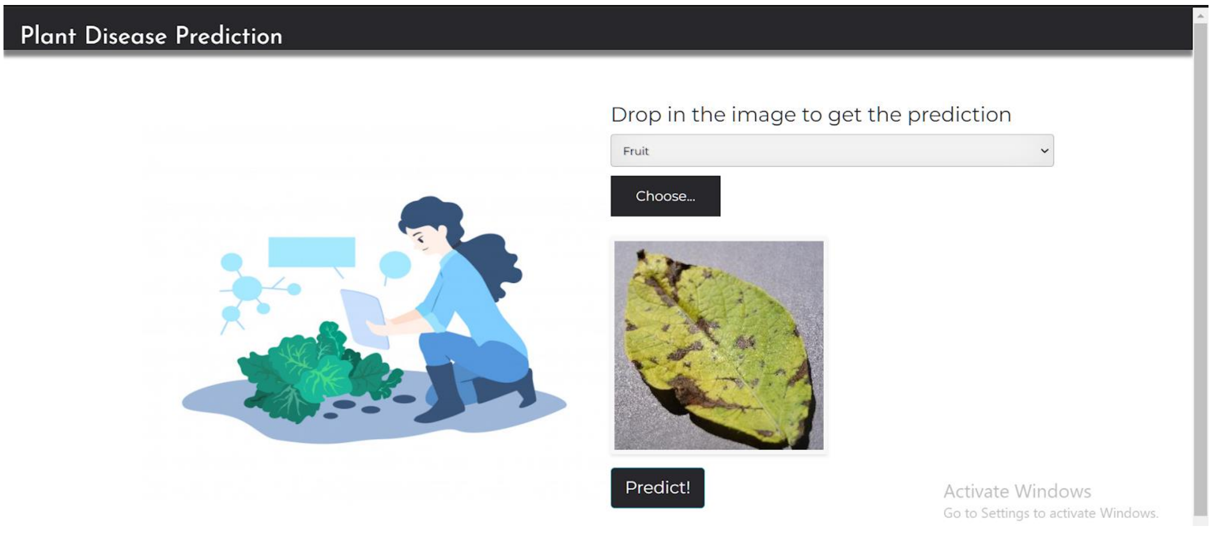
The home page looks like this.

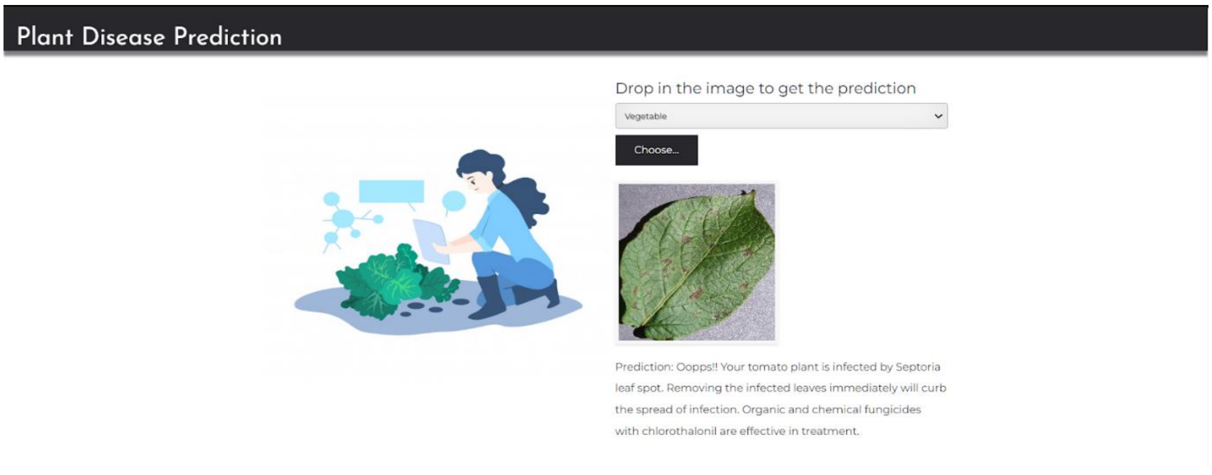

Plant Disease Prediction

Detect if your plant is infected!!

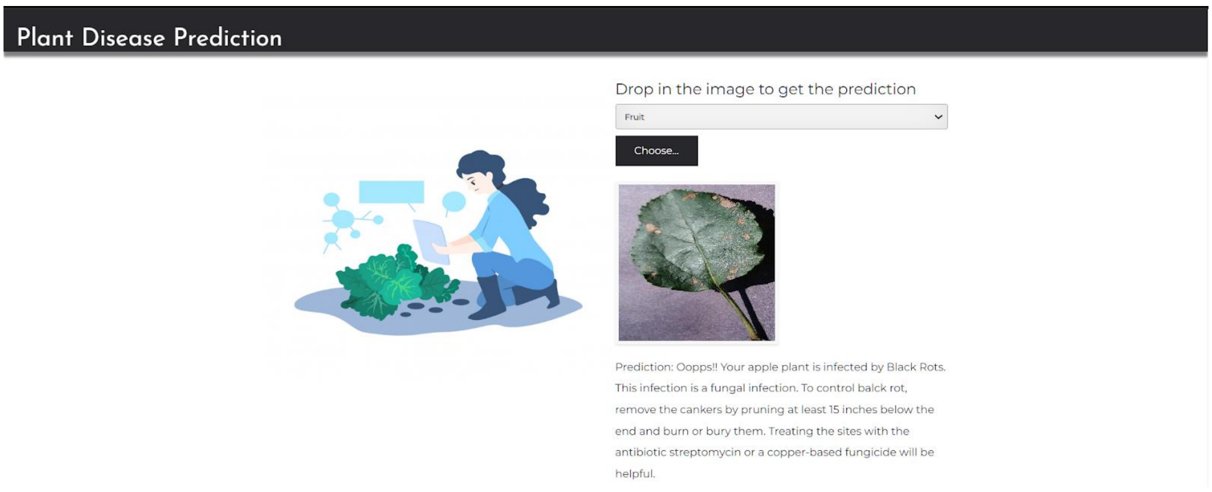
Agriculture is one of the major sectors worls wide. Over the years it has developed and the use of new technologies and equipment replaced almost all the traditional methods of farming. The plant diseases effect the production. Identification of diseases and taking necessary precautions is all done through naked eye, which requires labour and laboratries. This application helps farmers in detecting the diseases by observing the spots on the leaves, which inturn saves effort and labor costs.

After clicking on predict button, you will be redirected to the prediction page where you can browse the images.


Plant Disease Prediction
Drop in the image to get the prediction
Fruit
Choose...
Predict!
Activate Windows
Go to Settings to activate Windows.



Prediction: Oopps!! Your tomato plant is infected by Septoria leaf spot. Removing the infected leaves immediately will curb the spread of infection. Organic and chemical fungicides with chlorothalonil are effective in treatment.



Prediction: Oopps!! Your apple plant is infected by Black Rots. This infection is a fungal infection. To control balck rot, remove the cankers by pruning at least 15 inches below the end and burn or bury them. Treating the sites with the antibiotic streptomycin or a copper-based fungicide will be helpful.