

Tomato Plant Disease Detection From Leaf Images Using Deep Learning

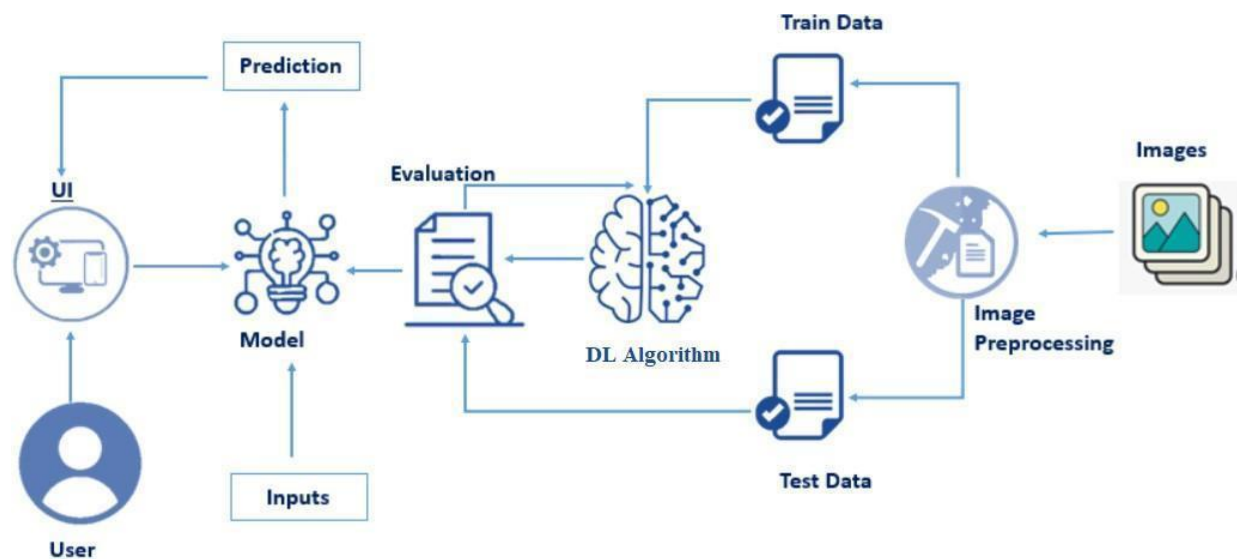
Introduction:

Tomato Plant is prone to having various diseases during its growth stage. If these diseases are not cured on time, they lead to the death of the plant and the harvest does not give a good yield. Moreover, in a tomato farm this disease can be transferred from one plant to other very rapidly. The farmers are concerned about their harvest.

It is not possible for the farmers to pay the agriculture experts hefty fees every time their plants develop a disease. We have come up with a solution to this problem. We have trained an AI model which can be used by farmers to check the disease their tomato plant might be having. The users need to upload image of a tomato leaf to the website and click on the submit button. Our model will give its prediction as to which disease is probable based on the image. The users can then visit the local agriculture store to ask for medicines for particular plant diseases. Our model can predict up to 9 tomato plant diseases and is also capable of pointing out if the plant looks healthy.

This model is useful for farmers, agriculture scientists, home farmers, gardeners, etc. This AI model is made using Convolutional Neural networks and under CNN we will be using transfer learning. Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in image analysis and classification. We used Transfer Learning technique ResNet152V2 that is more widely used as a transfer learning method in image analysis, and it is highly effective.

Technical Architecture:



Prior Knowledge:

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

- Deep Learning Concepts

- CNN: <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>
- ResNet-50: <https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33>
- Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.
Link: https://www.youtube.com/watch?v=lj4I_CvBnt0

Prerequisites:

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

- Anaconda navigator, VS Code and Spyder:
 - Refer the link below to download anaconda navigator
 - Link (VS Code) : <https://youtu.be/1ra4zH2G4o0>
 - Link (Spyder) : <https://youtu.be/5mDYijMfSzs>
- Python packages:
- Open anaconda prompt as administrator
- Type “pip install numpy” and click enter.
- Type “pip install pandas” and click enter.
- Type “pip install tensorflow ==2.3.2” and click enter.
- Type “pip install keras ==2.3.1” and click enter.
- Type “pip install Flask” and click enter.

Project Objectives:

- Know fundamental concepts of Deep Learning Concepts :CNN.Resnet152v2
- Gain knowledge in the pre-trained model Resnet152v2.
- Gain knowledge of Web Integration.

Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- The ResNet Model analyzes the image, then the prediction is showcased on the Flask UI.

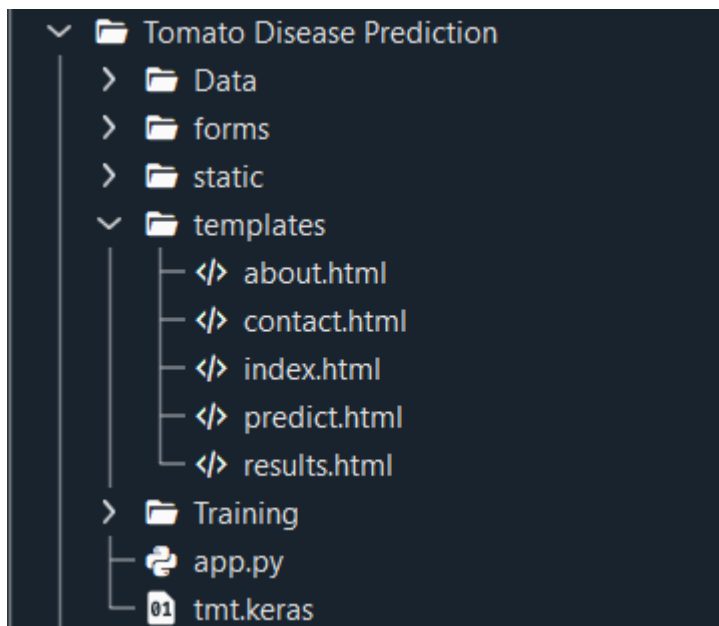
To accomplish this, we must complete all the activities and tasks listed below

- Data Collection.
 - Create a Train and Test path.

- Data Pre-processing.
- Import the required library
- Configure ImageDataGenerator class
- Apply ImageDataGenerator functionality to Trainset and Test set
- Model Building
 - Pre-trained CNN model as a Feature Extractor
 - Adding Dense Layer
 - Configure the Learning Process
 - Train the model
 - Save the Model
 - Test the model
- Application Building
 - Create an HTML file
 - Build Python Code

Project Structure:

Create a Project folder which contains files as shown below



- Static folder contains css files
- Template folder contains all 4 HTML pages.
- Data folder contains Training and Validation images
- Training file consist of train.ipynb , tmt.keras model

Milestone 1: Collection of Data

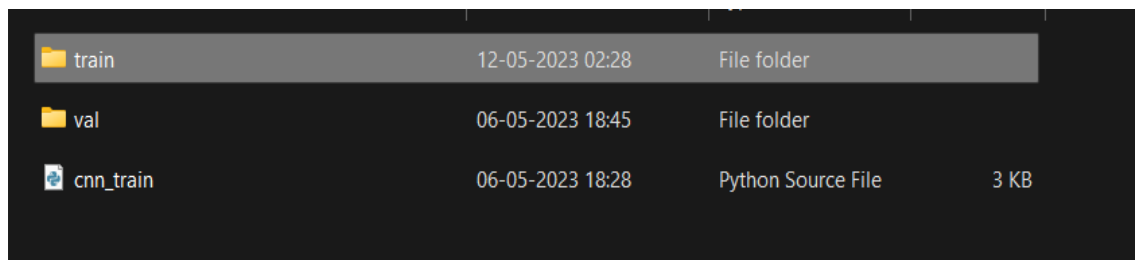
In this project, we have collected images of 10 types of Tomato Leaf images like Heatly, Spider Mites, Yellow leaf curl, etc. and they are saved in the respective sub directories with their respective names.

You can download the dataset used in this project using the below link

Dataset: <https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf>

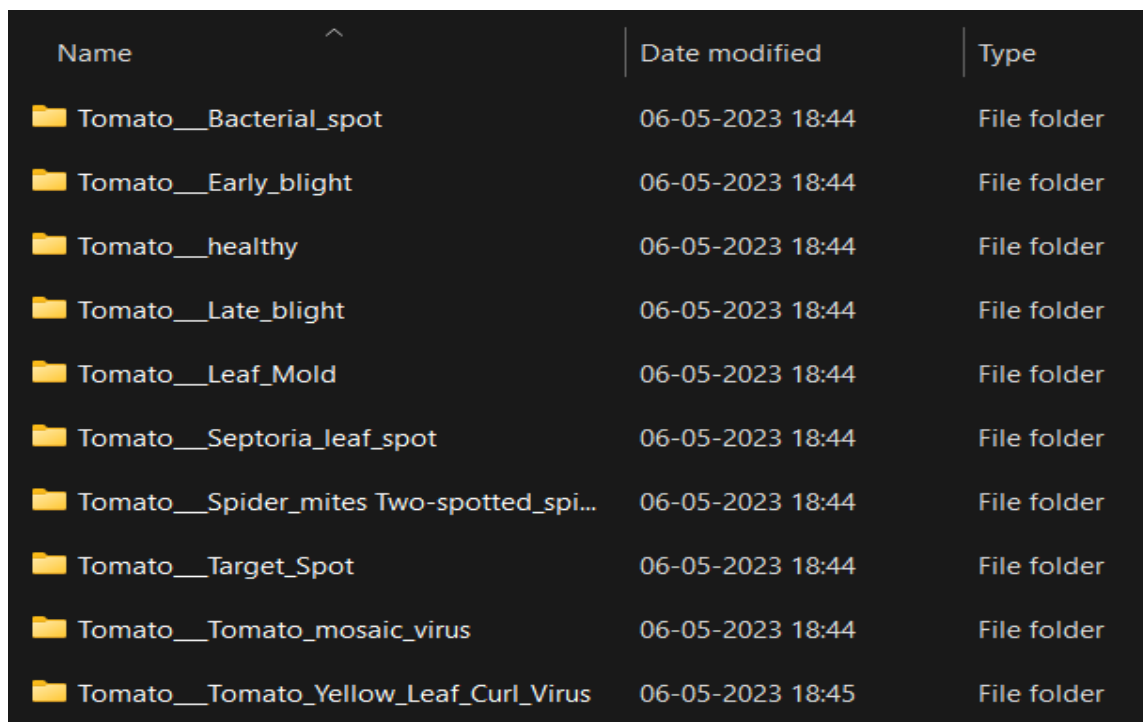
Activity 1: Creating training and Testing Dataset

Here we can see that the training and testing data is given in 2 separate folders.



train	12-05-2023 02:28	File folder	
val	06-05-2023 18:45	File folder	
cnn_train	06-05-2023 18:28	Python Source File	3 KB

Inside the training folder there are several folders for different classes.



Name	Date modified	Type
Tomato__Bacterial_spot	06-05-2023 18:44	File folder
Tomato__Early_blight	06-05-2023 18:44	File folder
Tomato__healthy	06-05-2023 18:44	File folder
Tomato__Late_blight	06-05-2023 18:44	File folder
Tomato__Leaf_Mold	06-05-2023 18:44	File folder
Tomato__Septoria_leaf_spot	06-05-2023 18:44	File folder
Tomato__Spider_mites Two-spotted_spi...	06-05-2023 18:44	File folder
Tomato__Target_Spot	06-05-2023 18:44	File folder
Tomato__Tomato_mosaic_virus	06-05-2023 18:44	File folder
Tomato__Tomato_Yellow_Leaf_Curl_Virus	06-05-2023 18:45	File folder

The same is the case for 'val' folder.

Sample data:



Milestone 2: Image Pre-processing

In this milestone, we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Loading the Dataset

Download the dataset from the Kaggle

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

Mounted at /content/drive

[ ] import os
    os.chdir('/content/drive/MyDrive/dataset/tomato')
    os.listdir()

['cnn_train.py', 'train', 'val']
```

Activity 2: Import the libraries

Import the necessary libraries as shown in the image

```
Importing the Libraries

[ ] import os
    import warnings

    warnings.filterwarnings('ignore')

    import numpy as np
    import pandas as pd

    import matplotlib.pyplot as plt
    from PIL import Image

    from tensorflow import keras
    from tensorflow.keras.applications import ResNet152V2
    from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
    from tensorflow.keras.models import Model
```

Configure ImageDataGenerator Class

Configure ImageDataGenerator Class

```
[ ] datagen=keras.preprocessing.image.ImageDataGenerator(rescale=1/255, validation_split=0.3)
    datagen2=keras.preprocessing.image.ImageDataGenerator(rescale=1/255)
```

Applying ImageDataGenerator Functionality To Train Set, Validation and Test Set

```
[ ] #Training and validation dataset

train = datagen.flow_from_directory('./train', seed=123, subset='training')
val = datagen.flow_from_directory('./train', seed=123, subset='validation')

#Test dataset for evaluation
test = datagen2.flow_from_directory('./val')
```

Found 7035 images belonging to 10 classes.
Found 3015 images belonging to 10 classes.
Found 1000 images belonging to 10 classes.

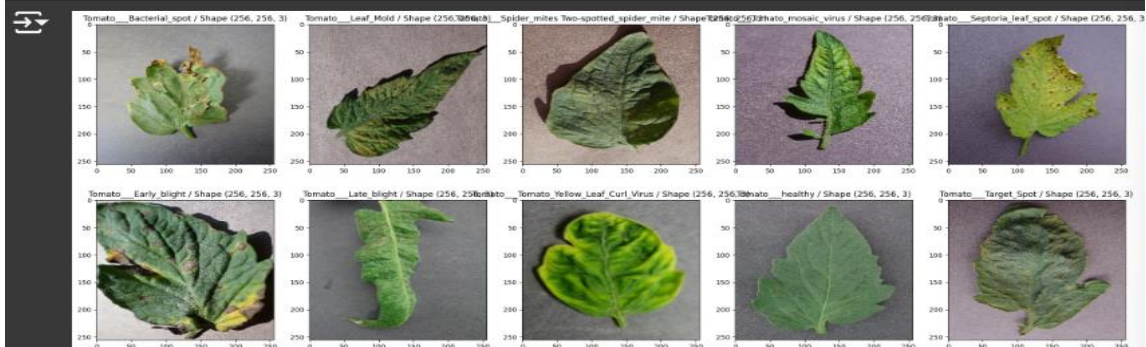
Preview of Images

#Training data visualization

```
classes= os.listdir('./train')

plt.figure(figsize=(25,10))

for i in enumerate (classes):
    pic=os.listdir('./train/'+i[1])[0]
    image=Image.open('./train/'+ i[1]+ '/' +pic)
    image=np.asarray(image)
    plt.subplot(2,5,i[0]+1)
    plt.title('{0} / Shape {1}'. format(i[1], image.shape))
    plt.imshow(image)
plt.show()
```



Milestone 3: Model Building

Activity 1: Pre-trained CNN model as a Feature Extractor

```
def get_model():  
    base_model = ResNet152V2(input_shape=(256,256,3), include_top=False)  
    for layers in base_model.layers[:140]:  
        layers.trainable = False  
    for layers in base_model.layers [140:]:  
        layers.trainable = True  
    x = base_model.output  
    x = GlobalAveragePooling2D()(x)  
    x = Dense (1000, activation='relu')(x)  
    pred=Dense(10, activation='softmax')(x)  
    model = Model(inputs=base_model.input, outputs=pred)  
    return model
```

Activity 2: Adding Dense Layer and Average Pooling layer

```
model=get_model()  
model.summary()
```

Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/234545216/234545216> 1s 0us/step
Model: "functional"

Layer (type)	Output Shape	Param #	Connected
input_layer (InputLayer)	(None, 256, 256, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	input_layer
conv1_conv (Conv2D)	(None, 128, 128, 64)	9,472	conv1_pad
pool1_pad (ZeroPadding2D)	(None, 130, 130, 64)	0	conv1_conv
pool1_pool (MaxPooling2D)	(None, 64, 64, 64)	0	pool1_pad
conv2_block1_preact_bn (BatchNormalization)	(None, 64, 64, 64)	256	pool1_pool
conv2_block1_preact_relu (Activation)	(None, 64, 64, 64)	0	conv2_block1_preact_bn
conv2_block1_1_conv (Conv2D)	(None, 64, 64, 64)	4,096	conv2_block1_preact_relu
conv2_block1_1_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block1_1_conv
conv2_block1_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block1_1_bn
conv2_block1_2_pad	(None, 66, 66, 64)	0	conv2_block1_1_relu

(Conv2D)			
conv5_block3_2_bn (BatchNormalization)	(None, 8, 8, 512)	2,048	conv5_block3_2_bn
conv5_block3_2_relu (Activation)	(None, 8, 8, 512)	0	conv5_block3_2_relu
conv5_block3_3_conv (Conv2D)	(None, 8, 8, 2048)	1,050,624	conv5_block3_3_conv
conv5_block3_out (Add)	(None, 8, 8, 2048)	0	conv5_block3_out
post_bn (BatchNormalization)	(None, 8, 8, 2048)	8,192	conv5_block3_out
post_relu (Activation)	(None, 8, 8, 2048)	0	conv5_block3_out
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	post_relu
dense (Dense)	(None, 1000)	2,049,000	global_average_pooling2d
dense_1 (Dense)	(None, 10)	10,010	dense

Total params: 60,390,658 (230.37 MB)
Trainable params: 56,430,338 (215.26 MB)
Non-trainable params: 3,960,320 (15.11 MB)

Activity 3. Configure the Learning Process

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


Activity 4: Training the model

```
model.fit(train, batch_size=80, epochs=15, validation_data=val)
```

```
Epoch 1/15
220/220 ————— 2918s 13s/step - accuracy: 0.6506 - loss: 1.1542 - val_accuracy: 0.9522 - val_loss: 0.1507
Epoch 2/15
220/220 ————— 162s 669ms/step - accuracy: 0.9753 - loss: 0.0923 - val_accuracy: 0.9619 - val_loss: 0.1145
Epoch 3/15
220/220 ————— 202s 668ms/step - accuracy: 0.9949 - loss: 0.0271 - val_accuracy: 0.9725 - val_loss: 0.0901
Epoch 4/15
220/220 ————— 147s 664ms/step - accuracy: 0.9974 - loss: 0.0137 - val_accuracy: 0.9745 - val_loss: 0.0769
Epoch 5/15
220/220 ————— 147s 665ms/step - accuracy: 0.9980 - loss: 0.0099 - val_accuracy: 0.9715 - val_loss: 0.0885
Epoch 6/15
220/220 ————— 168s 758ms/step - accuracy: 0.9988 - loss: 0.0068 - val_accuracy: 0.9731 - val_loss: 0.0838
Epoch 7/15
220/220 ————— 183s 672ms/step - accuracy: 0.9986 - loss: 0.0061 - val_accuracy: 0.9718 - val_loss: 0.0855
Epoch 8/15
220/220 ————— 147s 664ms/step - accuracy: 0.9995 - loss: 0.0050 - val_accuracy: 0.9672 - val_loss: 0.0942
Epoch 9/15
220/220 ————— 147s 664ms/step - accuracy: 0.9996 - loss: 0.0050 - val_accuracy: 0.9745 - val_loss: 0.0761
Epoch 10/15
220/220 ————— 147s 664ms/step - accuracy: 0.9996 - loss: 0.0035 - val_accuracy: 0.9751 - val_loss: 0.0793
Epoch 11/15
220/220 ————— 203s 667ms/step - accuracy: 0.9985 - loss: 0.0042 - val_accuracy: 0.9755 - val_loss: 0.0804
Epoch 12/15
220/220 ————— 147s 663ms/step - accuracy: 0.9997 - loss: 0.0024 - val_accuracy: 0.9678 - val_loss: 0.1198
Epoch 13/15
220/220 ————— 223s 758ms/step - accuracy: 0.9991 - loss: 0.0051 - val_accuracy: 0.9731 - val_loss: 0.0811
Epoch 14/15
220/220 ————— 148s 666ms/step - accuracy: 0.9993 - loss: 0.0029 - val_accuracy: 0.9721 - val_loss: 0.0870
Epoch 15/15
220/220 ————— 147s 664ms/step - accuracy: 0.9990 - loss: 0.0033 - val_accuracy: 0.9738 - val_loss: 0.0819
<keras.src.callbacks.history.History at 0x7ae321514910>
```

Activity 5: Testing the model

```
#Prediction and visualizations

classes = os.listdir('./val')

plt.figure(figsize=(18,28))

for i in enumerate(classes):

    pic = os.listdir('./val/'+i[1])

    pic = pic[np.random.randint(len(pic)-1)]

    image = Image.open('./val/' + i[1] + '/' + pic)

    image = np.asarray(image)

    pred = np.argmax(model.predict(image.reshape(-1,256,256,3)/255))
    for j in list(enumerate(list(test.class_indices.keys()))):
        if pred == j[0]:
            prediction = j[1]
    plt.subplot(5,2,i[0]+1)

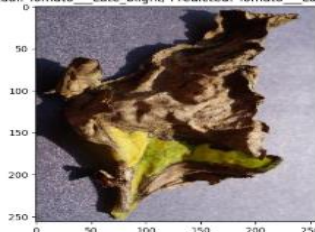
    plt.title('Actual: {0}/ Predicted: {1}'.format(i[1], prediction))

    plt.imshow(image)

plt.show()
```

1/1 11s 11s/step
 1/1 0s 49ms/step
 1/1 0s 58ms/step
 1/1 0s 40ms/step
 1/1 0s 43ms/step
 1/1 0s 39ms/step
 1/1 0s 46ms/step
 1/1 0s 45ms/step
 1/1 0s 42ms/step
 1/1 0s 44ms/step

Actual: Tomato_Late_blight/ Predicted: Tomato_Late_blight



Actual: Tomato_Spider_mites Two-spotted_spider_mite/ Predicted: Tomato_Leaf_Mold



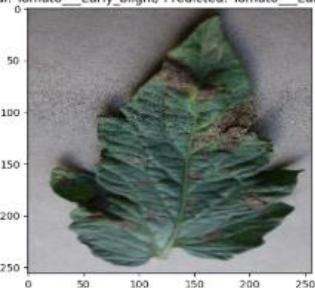
Actual: Tomato_Target_Spot/ Predicted: Tomato_Target_Spot



Actual: Tomato_Tomato_mosaic_virus/ Predicted: Tomato_Leaf_Mold



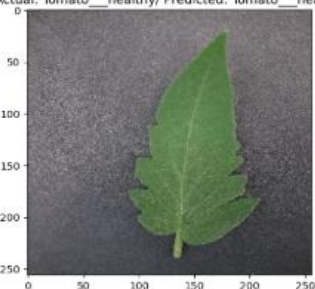
Actual: Tomato_Early_blight/ Predicted: Tomato_Early_blight



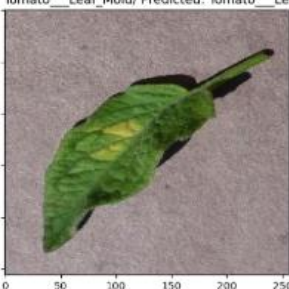
Actual: Tomato_Tomato_Yellow_Leaf_Curl_Virus/ Predicted: Tomato_Tomato_Yellow_Leaf_Curl_Virus



Actual: Tomato_healthy/ Predicted: Tomato_healthy



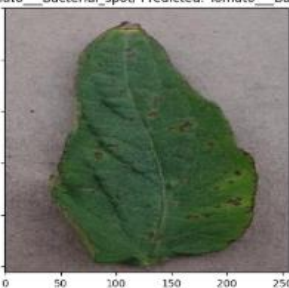
Actual: Tomato_Leaf_Mold/ Predicted: Tomato_Leaf_Mold



Actual: Tomato_Septoria_leaf_spot/ Predicted: Tomato_Septoria_leaf_spot



Actual: Tomato_Bacterial_spot/ Predicted: Tomato_Bacterial_spot



Milestone 4: Save the model

The model is saved as tmt.keras. A keras file is a data file saved in the keras format. It contains multidimensional arrays of scientific data.

```
] from tensorflow.keras.models import load_model

# Saving the model
model.save('tmt.keras')

# Loading the model
model = load_model('tmt.keras')

] import joblib
import os
joblib.dump(model, 'tmt.joblib')

['tmt.joblib']
```

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the user where they have to upload the image for predictions. The entered image is given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script

Activity 1: Building HTML Pages

- We use HTML to create the front-end part of the web page.
- Here, we have created 4 HTML pages- home.html, about.html, predict.html,result.html.

Activity 2: Build Python Code

Write the below code in Flask app.py python file script to run the Object Project. To upload image in UI, To display the image in UI:

```

import pickle
import warnings
warnings.filterwarnings('ignore')
import os
from PIL import Image
from flask import Flask, request, render_template
import numpy as np
from tensorflow.keras.models import load_model

model = load_model('tmt.keras')
app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html')
@app.route('/about')
def about():
    return render_template('about.html')
@app.route('/details')
def pred():
    return render_template('predict.html')
@app.route('/result', methods = ['GET', 'POST'])
def predict():
    if request.method == "POST":
        f=request.files['image']
        basepath=os.path.dirname(__file__) #getting the current path i.e where app.py is present
        #print("current path", basepath)
        filepath=os.path.join(basepath, 'Data', 'val', 'Tomato_healthy', f.filename) #from anywhere in the system we )
        #print("upload folder is", filepath )
        f.save(filepath)
        image = Image.open(filepath)
        image = np.asarray(image)
        pred = np.argmax(model.predict(image.reshape(-1,256,256,3)/255))

classes = ['Tomato_Bacterial_spot', 'Tomato_Early_blight', 'Tomato_healthy', 'Tomato_Late_blight', 'Tomato_Leaf_Mold', 'Tomato_Septoria_Leaf_Spot', 'Tomato_Spider_mites', 'Tomato_Target_Spot', 'Tomato_Tomato_mosaic_virus', 'Tomato_Leaf_Yellowing', 'Tomato_Yellow_Leaf_Curl_Virus']
keys = ['Tomato_Bacterial_spot', 'Tomato_Early_blight', 'Tomato_Late_blight', 'Tomato_Leaf_Mold', 'Tomato_Septoria_Leaf_Spot', 'Tomato_Spider_mites', 'Tomato_Target_Spot', 'Tomato_Tomato_mosaic_virus', 'Tomato_Leaf_Yellowing', 'Tomato_Yellow_Leaf_Curl_Virus']
for j in list(enumerate(keys)):
    if pred == j[0]:
        prediction = j[1]
        if prediction == 'Tomato_Bacterial_spot':
            return render_template('results.html', prediction_text = "to have Bacterial spots.")
        elif prediction == 'Tomato_Early_blight':
            return render_template('results.html', prediction_text = "to have Early Blights.")
        elif prediction == 'Tomato_healthy':
            return render_template('results.html', prediction_text = "to be healthy.")
        elif prediction == 'Tomato_Late_blight':
            return render_template('results.html', prediction_text= "to have Late Blights.")
        elif prediction == 'Tomato_Septoria_Leaf_Spot':
            return render_template('results.html', prediction_text = "to have Septoria Leaf Spot. ")
        elif prediction == 'Tomato_Spider_mites':
            return render_template('results.html', prediction_text = "to have Spider Mites.")
        elif prediction == 'Tomato_Target_Spot':
            return render_template('results.html', prediction_text= "to have Target spots.")
        elif prediction == 'Tomato_Tomato_mosaic_virus':
            return render_template('results.html', prediction_text= "to have Tomato Mosaic Virus.")
        elif prediction == 'Tomato_Leaf_Yellowing':
            return render_template('results.html', prediction_text= "to have Leaf Yellowing.")
        elif prediction == 'Tomato_Yellow_Leaf_Curl_Virus':
            return render_template('results.html', prediction_text = "to have Tomato Yellow Leaf Curl Virus.")

__name__ == "__main__":
    app.run(debug=True)

```

Getting local host in the terminal while running app.py:

```

* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
* Restarting with watchdog (windowsapi)

```

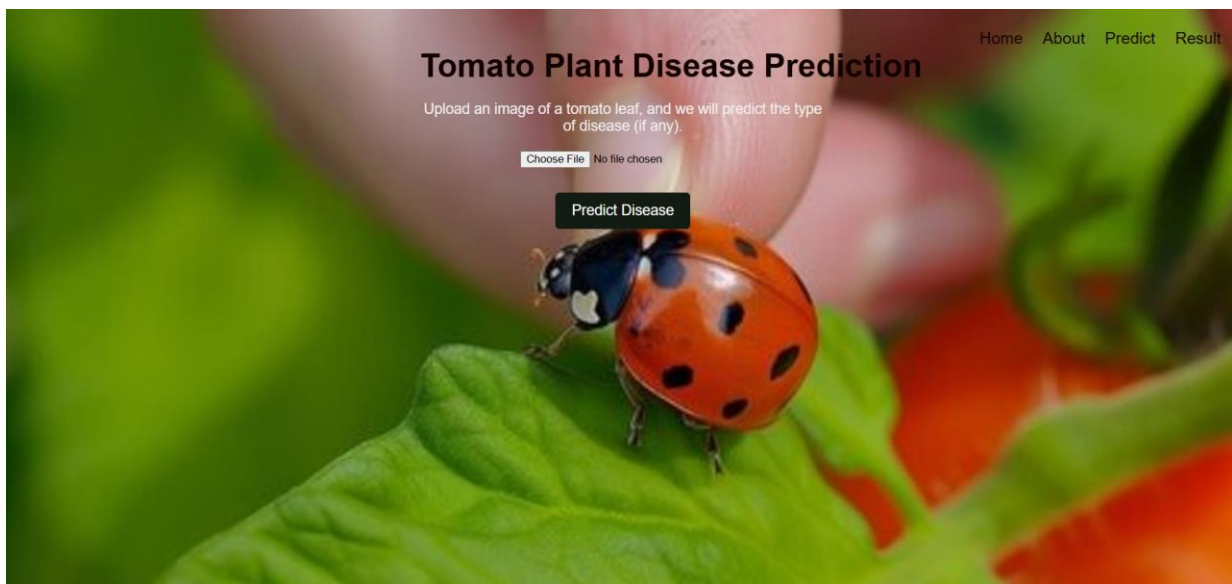

home.html is displayed below:



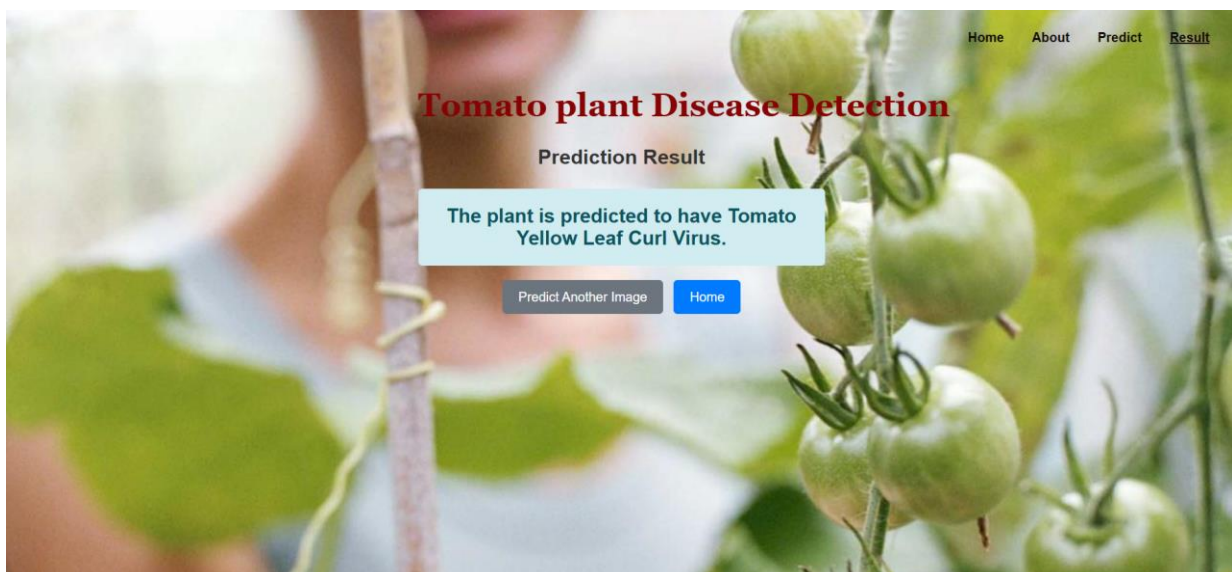
About.html is displayed below:



Predict.html is displayed below:



Result.html is displayed below Input:1



Input:2



Output: 2

