Final Project Report Template

Date	11 July 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis

1. Introduction

1.1 Project overviews

Develop a deep learning model for detecting tomato plant diseases from leaf images. By analysing leaf characteristics and symptoms, this project aims to accurately classify various diseases, enabling early detection, targeted treatment, and crop management practices for tomato farmers, ensuring the crop yield and health.

1.2 Project Objective

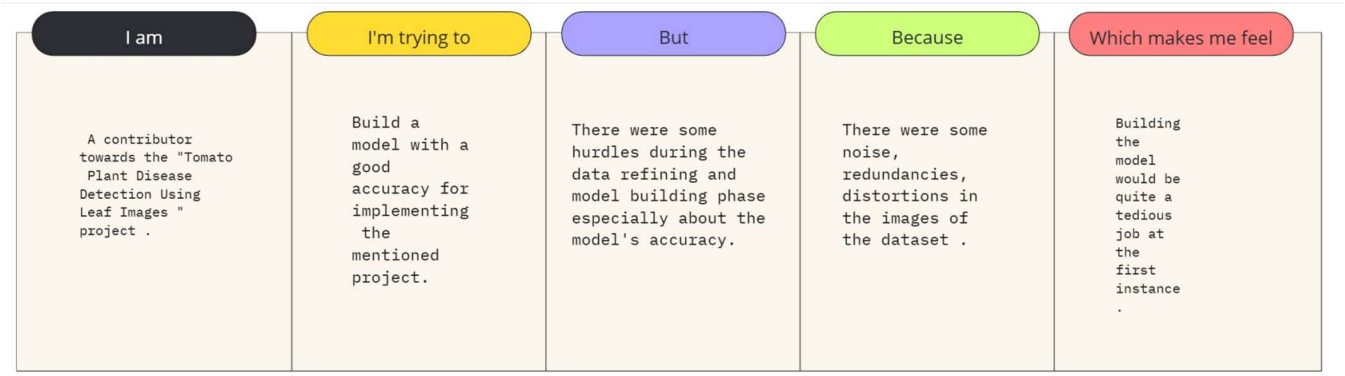
1. **Accurate Disease Classification:** To develop a model to distinguish between multiple diseases and healthy leaves.
2. **Early Detection:** To enable timely interventions to minimize crop loss.
3. **Deep Learning Model:** To implement the deep learning model to assist tomato farmers in early disease detection and management and thus improve crop yield, reduce economic losses due to disease outbreaks, and promote sustainable farming practices.
4. **Integration:** Integrate the disease detection model into agricultural extension services for providing recommendations and interventions to farmers. Enhance agricultural advisory services, facilitate the timely disease control measures and empower farmers with actionable insights.

2. Project Initialization and Planning Phase

Date	27 June 2024
Team ID	SWTID1719992739
Project Name	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	3 Marks

2.1 Define Problem Statements (Customer Problem Statement Template):

Our problem statement is to create a deep learning model to detect tomato plant disease through images of their leaves.



2. Project Initialization and Planning Phase

Date	27 June 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	3 Marks

2.2 Project Proposal (Proposed Solution) template

This project proposal outlines a solution to address a specific problem. With a clear objective, defined scope, and a concise problem statement, the proposed solution details the approach, key features, and resource requirements, including hardware, software, and personnel.

Project Overview	
Objective	Develop an automated system that accurately identifies and classifies various diseases in tomato plants based on analysis of digital leaf images.
Scope	This project will include the necessary steps to prepare the data for modelling. This project will involve designing and training a CNN Model (Resnet 152 V2) but will not include exploration of multiple deep learning architectures. This project will result in a functional application prototype. This project will be trained to detect a predefined set of common tomato plant diseases. A simple and intuitive interface allowing users to upload images and view the results.
Problem Statement	
Description	Tomato Plant Disease Detection from Leaf Images is a technology that utilizes image analysis and machine learning to automatically identify diseases in tomato plants.

Impact	Increased tomato crop yield and improved food security, Reduced reliance on chemical pesticides.
Proposed Solution	
Approach	We used a pre trained model of Resnet152V2(transfer learning) and trained our model with the data freely available on Kaggle. We also used additional 2 dense layers for better accuracy.
Key Features	Simple and intuitive interface to the users to upload images of tomato leaves. Quick and accurate predictions of the disease present in the uploaded leaf image. Identification of the specific disease affecting the tomato plant. Techniques to augment data, increasing the model's robustness to various image conditions.

Resource Requirements

Resource Type	**Description**	**Specification/Allocation**
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	I5 processor and Nvidia GTX 1650 4GB
Memory	RAM	8 GB
Storage	Disk space for data, models, and logs	Data: 188MB, Model: 447MB
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	Numpy, Pandas, Matplotlib, Tensorflow.
Development Environment	IDE, version control	VS Code, GitHub

Data		
Data	Source, size, format	Kaggle, 188 MB, Images (JPG)

2. Initial Project Planning Template

Date	27 June 2024
Team ID	SWTID1719992739
Project Name	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	4 Marks

2.3 Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create a product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Team Formation and Project Selection	USN-1	We formed a team and chose the project title .	1	High	Saptarshi, Krishanu, Shriya, Argha	27 June 2024	28 June 2024
Sprint-2	Data collection	USN-2	As a team , we researched and gathered the dataset from Kaggle website .	2	Medium	Saptarshi, Krishanu, Shriya, Argha	2nd July 2024	3rd July 2024
Sprint-3	Data preprocessing	USN-3	We did the preprocessing on the dataset to remove redundancies and irrelevant details and make the data more reliable .	2	Medium	Saptarshi, Krishanu, Shriya, Argha	02nd July 2024	3rd July 2024
Sprint-4	Building the model	USN-4	We worked as a team and built a model for our problem statement .	5	High	Saptarshi, Krishanu, Shriya,	05 July 2024	07 July 2024

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
						Argha		
Sprint-5	Application building	USN-5	We build the application using Flask	5	High	Saptarshi, Krishanu, Shriya , Argha	08 July 2024	10 July 2024

3. Data Collection and Preprocessing Phase

Date	2nd July 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	2 Marks

3.1 Data Collection Plan & Raw Data Sources Identification Template

Elevate your data strategy with the Data Collection plan and the Raw Data Sources report, ensuring meticulous data curation and integrity for informed decision-making in every analysis and decision-making endeavor.

Data Collection Plan Template

Section	Description
Project Overview	The objective of this machine learning project is to develop a model capable of detecting diseases in tomato leaves using image classification techniques. This will involve using convolutional neural networks (CNNs) to identify various diseases in tomato plants from leaf images, helping farmers to take timely action and improve crop yield.
Data Collection Plan	Data will be collected from the Kaggle dataset "Tomato Leaf" which contains images of tomato leaves categorized by disease type. This dataset provides a diverse range of leaf conditions necessary for training a robust model.

Raw Data Sources Identified	The primary raw data source is a publicly available dataset on Kaggle.

Raw Data Sources Template

Source Name	Description	Location/URL	Format	Size	Access Permissions
Tomato Leaf Dataset	This dataset contains images of tomato leaves categorized into various disease types and a healthy category. The images are labeled and organized into directories for each category.	https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf	Image (JPG)	188 MB	Public

3. Data Collection and Preprocessing Phase

Date	2nd July 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	2 Marks

3.2 Data Quality Report Template

The Data Quality Report Template will summarize data quality issues from the selected source, including severity levels and resolution plans. It will aid in systematically identifying and rectifying data discrepancies.

Data Source	Data Quality Issue	Severity	Resolution Plan
Tomato Leaf Dataset	Image Resizing Discrepancies	low	**Resolution Plan:** Ensure all images are resized to a consistent dimension of 256x256 pixels using the `ImageDataGenerator`.

3. Data Collection and Preprocessing Phase

Date	2nd July 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	6 Marks

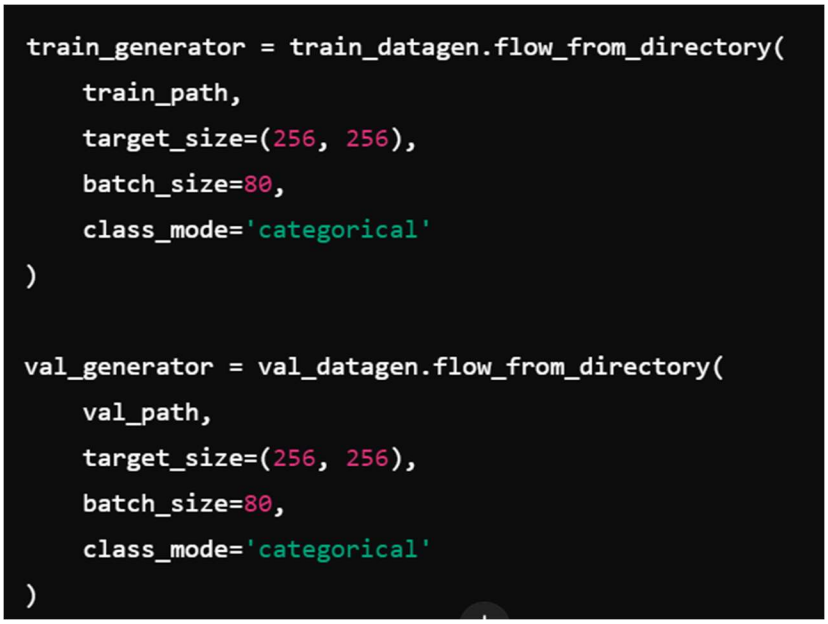
3.3 Preprocessing Template

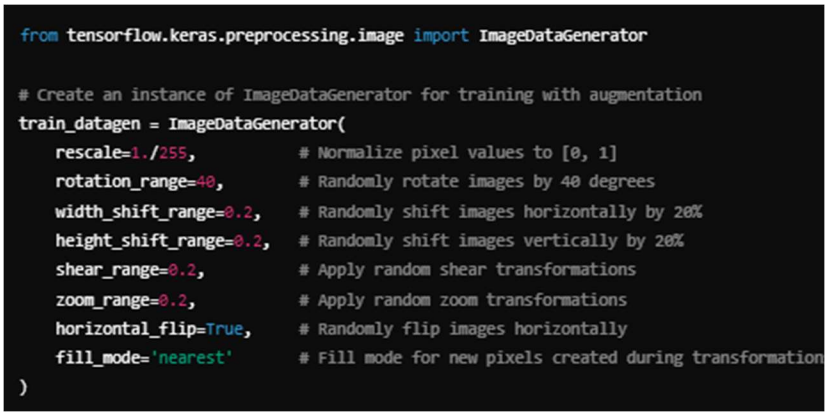
The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	**Description**
Data Overview	The dataset used for this project consists of images of tomato leaves, categorized into healthy leaves and leaves affected by various diseases. The dataset is split into training and validation sets.
Resizing	Resize images to a target size of 256x256 pixels.
Normalization	Normalize pixel values to the range [0, 1] by rescaling with a factor of 1./255.
Data Augmentation	Apply augmentation techniques such as flipping, rotation, shifting, zooming, or shearing.
Denoising	denoising is not applied in this project.

Edge Detection	Edge detection is not explicitly used in this project.
Color Space Conversion	Color space conversion is not explicitly used in this project.
Image Cropping	Image cropping is not explicitly used in this project.
Batch Normalization	Batch normalization is applied through the use of pre-trained models which include normalization layers.
Data Preprocessing Code Screenshots	

Loading Data	Python
	`from tensorflow.keras.preprocessing.image import ImageDataGenerator`
	`# Create an instance of ImageDataGenerator for training with augmentati`
	`train_datagen = ImageDataGenerator(`
	`rescale=1./255, # Normalize pixel values to [0, 1]`
	`rotation_range=40, # Randomly rotate images by 40 degrees`
	`width_shift_range=0.2, # Randomly shift images horizontally by 2`
	`height_shift_range=0.2, # Randomly shift images vertically by 20%`
	`shear_range=0.2, # Apply random shear transformations`
	`zoom_range=0.2, # Apply random zoom transformations`
	`horizontal_flip=True, # Randomly flip images horizontally`
	`fill_mode='nearest' # Fill mode for new pixels created during`
	`)`
	`# Create an instance of ImageDataGenerator for validation without augme`
	`val_datagen = ImageDataGenerator(rescale=1./255) # Only normalize pixe`

Resizing	
Normalization	

Data Augmentation	
Denoising	-
Edge Detection	-
Color Space Conversion	-
Image Cropping	
Batch Normalization	-

4. Model Development Phase Template

Date	11 July 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	5 Marks

4.1 Model Selection Report

In the model selection report for future deep learning and computer vision projects, various architectures, such as CNNs or RNNs, will be evaluated. Factors such as performance, complexity, and computational requirements will be considered to determine the most suitable model for the task at hand.

Model Selection Report:

Model	Description
ResNet152V2 (Used in our model)	ResNet152V2 is a deep convolutional neural network with 152 layers, designed for image recognition. It improves upon the original ResNet by using pre-activation (batch normalization and ReLU before convolutions) and residual connections, enhancing training stability and performance. It's highly effective for tasks like object detection and image classification. (Epoch 15 – loss: 0.0148 , accuracy- 0.9952, val_loss- 0.0656,val_accuracy- 0.9790, Training Time :- 3 hrs)
Mobilenetv2	MobileNetV2 is a lightweight convolutional neural network designed for mobile and edge device applications. It uses depthwise separable convolutions to reduce computational complexity and introduces inverted residuals and linear bottlenecks to improve efficiency and accuracy. With 53 layers, MobileNetV2 is

	effective for tasks like image classification, object detection, and semantic segmentation on resource-constrained devices. (Epochs-loss: 0.0148, accuracy: 0.9952, val_loss: 0.0656,val_accuracy:0.9790,training time: 3.5 hrs)
DenseNet	DenseNet (Dense Convolutional Network) features a unique architecture with dense connectivity, linking each layer to every other layer. This design improves gradient flow, encourages feature reuse, and reduces the number of parameters, enhancing efficiency. DenseNet has variants such as DenseNet-121, DenseNet-169, DenseNet-201, and DenseNet-264, named according to the number of layers they contain. DenseNet is highly effective for image classification, object detection, and segmentation tasks (Epochs-loss: 0.0148, accuracy: 0.9952, val_loss: 0.0656,val_accuracy:0.9790,training time: 4 hrs)

4. Model Development Phase Template

Date	9 July 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	10 Marks

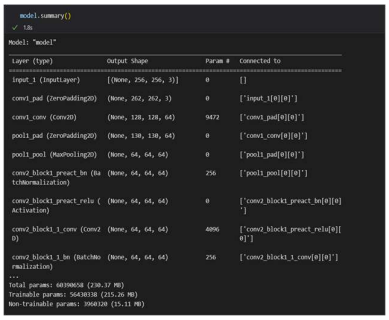
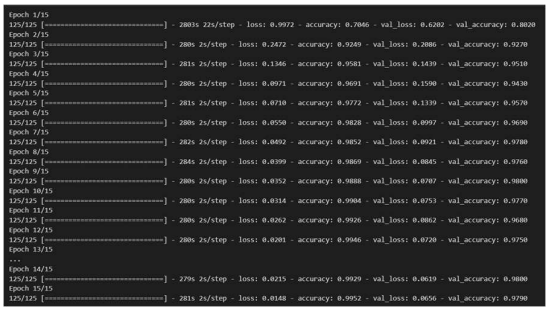
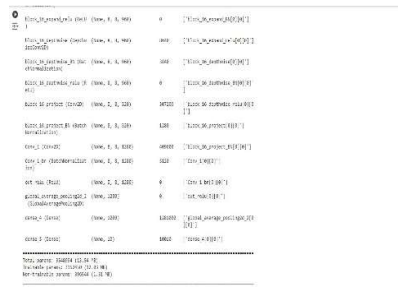
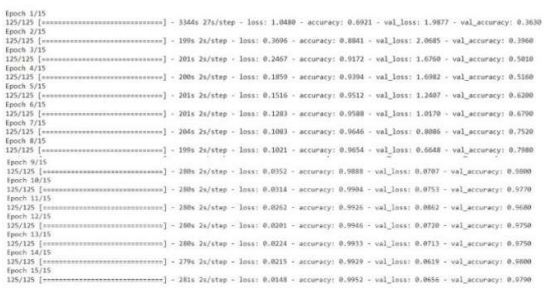
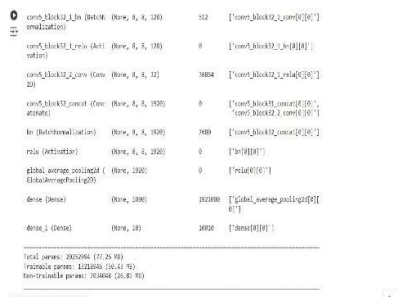
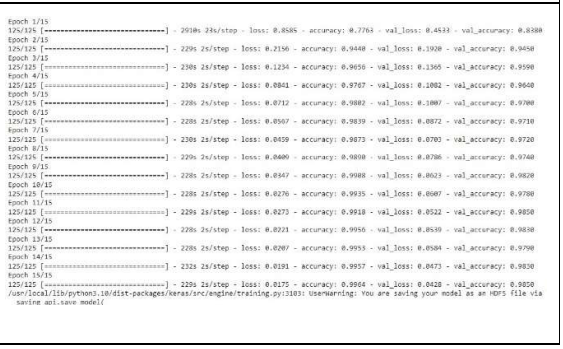
4.2 Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):

Paste the screenshot of the model training code

Model Validation and Evaluation Report (5 marks):

Model	Summary	Training and Validation Performance Metrics
ResNet152V2		
Mobilenetv2		
DenseNet		

5. Model Optimization and Tuning Phase Template

Date	11 july 2024
Team ID	SWTID1719992739
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases through Leaf Image Analysis
Maximum Marks	10 Marks

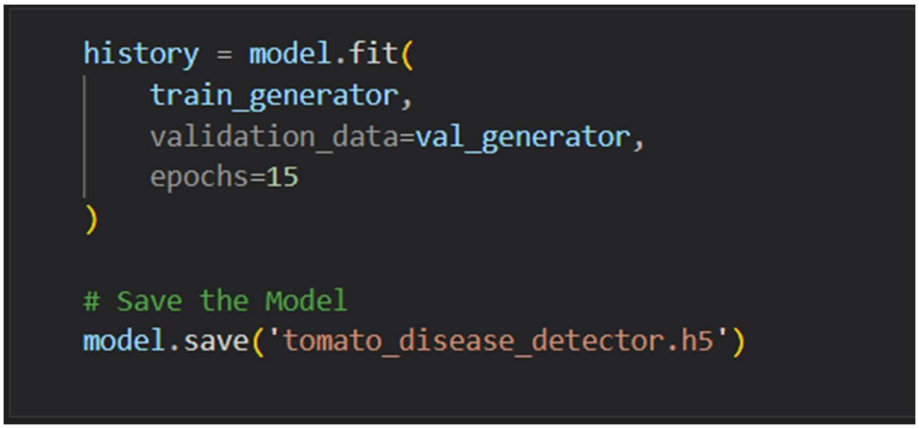
5.1 Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

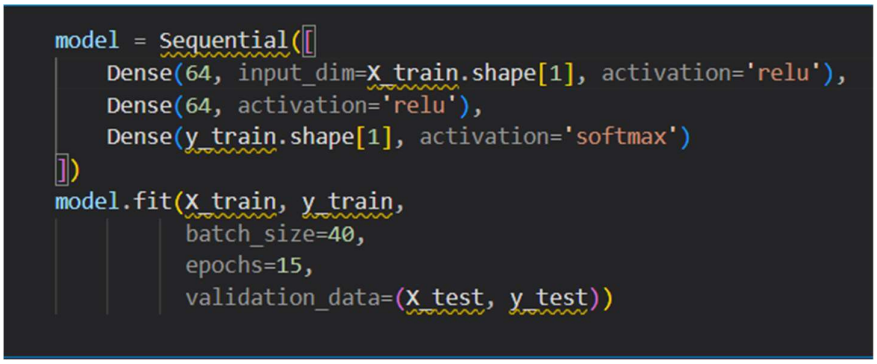
Model	Tuned Hyperparameters
ResNet152V2	Batch size: It refers to the number of training examples utilized in one iteration. It determines how many samples from the training dataset will be used to compute the error and update the model parameters. Epochs: It is one complete pass through the entire training dataset. During one epoch, the learning algorithm works through all the training samples once, and in larger datasets, this is often divided into several batches. The number of epochs determines how many times the learning algorithm will work through the entire dataset. More epochs can lead to better learning but also increases the risk of overfitting.

<table>
<tr>
<td></td>
<td>
Activation Function: An activation function in a neural network defines the output of a node given an input or set of inputs. It introduces non-linearity into the model, allowing the network to learn complex patterns. Without activation functions, the neural network would behave like a linear regression model, incapable of solving complex tasks.

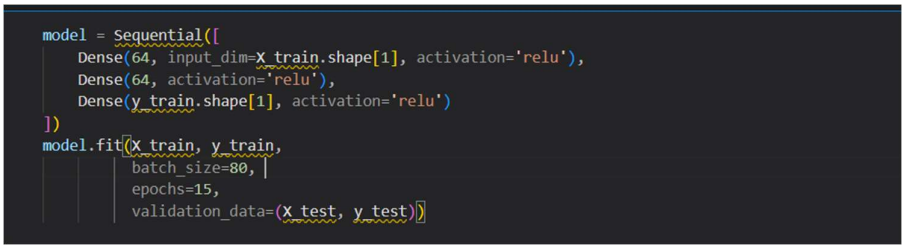


</td>
</tr>
<tr>
<td>Mobilenetv2</td>
<td>
Batch size: It refers to the number of training examples utilized in one iteration. It determines how many samples from the training dataset will be used to compute the error and update the model parameters.

Epochs: It is one complete pass through the entire training dataset. During one epoch, the learning algorithm works through all the training samples once, and in larger datasets, this is often divided into several batches. The number of epochs determines how many times the learning algorithm will work through the entire dataset. More epochs can lead to better learning but also increases the risk of overfitting.
</td>
</tr>
</table>

	Activation Function: An activation function in a neural network defines the output of a node given an input or set of inputs. It introduces non-linearity into the model, allowing the network to learn complex patterns. Without activation functions, the neural network would behave like a linear regression model, incapable of solving complex tasks
	 
	`model = Sequential([`
	`  Dense(64, input_dim=X_train.shape[1], activation='relu'),`
	`  Dense(64, activation='relu'),`
	`  Dense(y_train.shape[1], activation='softmax')`
	`])`
	`model.fit(X_train, y_train,`
	`  batch_size=40,`
	`  epochs=15,`
	`  validation_data=(X_test, y_test))`

DenseNet	Batch size: It refers to the number of training examples utilized in one iteration. It determines how many samples from the training dataset will be used to compute the error and update the model parameters. Epochs: It is one complete pass through the entire training dataset. During one epoch, the learning algorithm works through all the training samples once, and in larger datasets, this is often divided into several batches. The number of epochs determines how many times the learning algorithm will work through the entire dataset. More epochs can lead to better learning but also increases the risk of overfitting. Activation Function: An activation function in a neural network defines the output of a node given an input or set of inputs. It introduces non-linearity into the model, allowing the network to learn complex patterns. Without activation functions, the neural network would behave like a linear regression model, incapable of solving complex tasks.

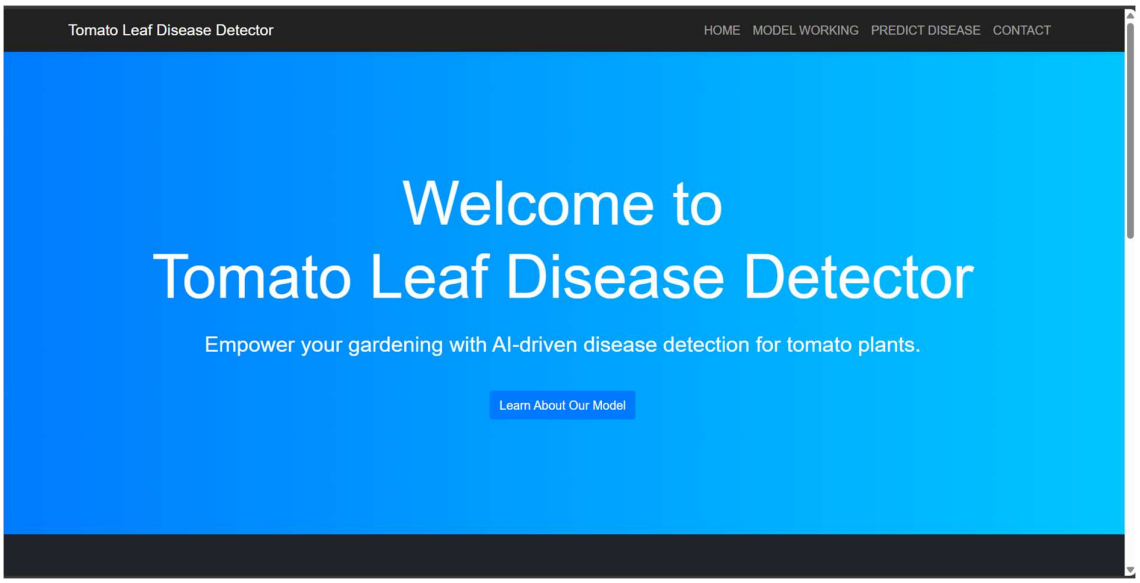
```
model = Sequential([
    Dense(64, input_dim=X_train.shape[1], activation='relu'),
    Dense(64, activation='relu'),
    Dense(y_train.shape[1], activation='relu')
])
model.fit(X_train, y_train,
          batch_size=80,
          epochs=15,
          validation_data=(X_test, y_test))
```

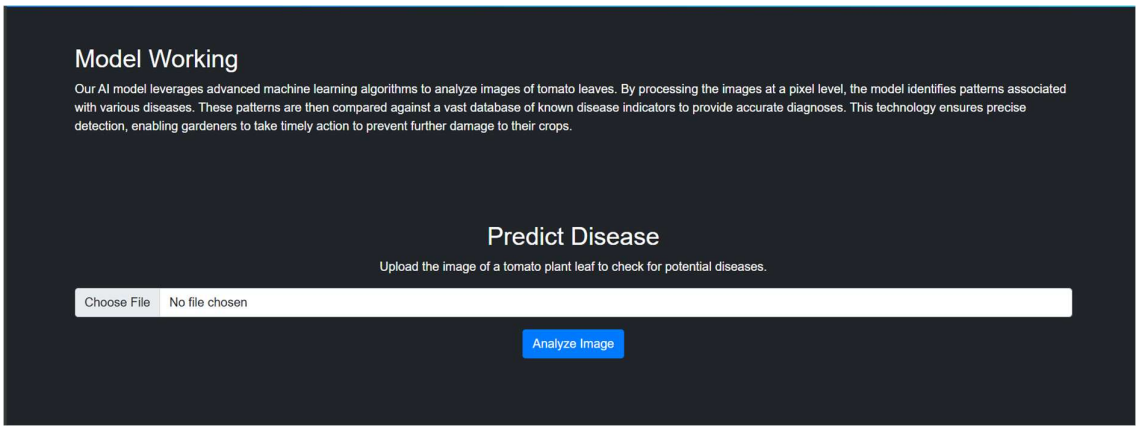
5.2 Final Model Selection Justification (2 Marks):

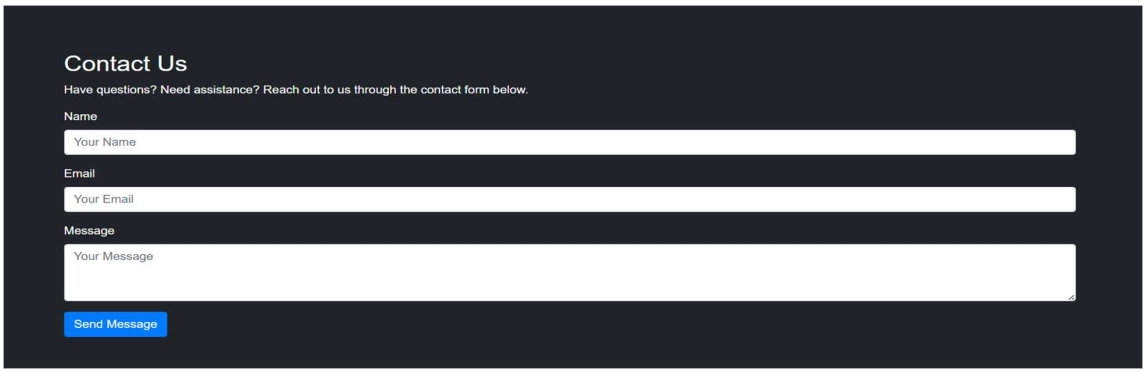
Final Model	Reasoning
ResNet152V2	We chose this model because we got an accuracy of 99.52% in a training time of around 180 minutes. We then tested the model on various images and it worked great without any overfitting whereas the model we made using dense net offered more accuracy (99.64%) but took around 240 minutes to train and was overfitting on some cases while testing on the images. Again, MobileNetV2 offered the same accuracy as ResNet152V2 but considering that it was meant for low power devices we rejected that and proceeded with ResNet152V2 As our final model.

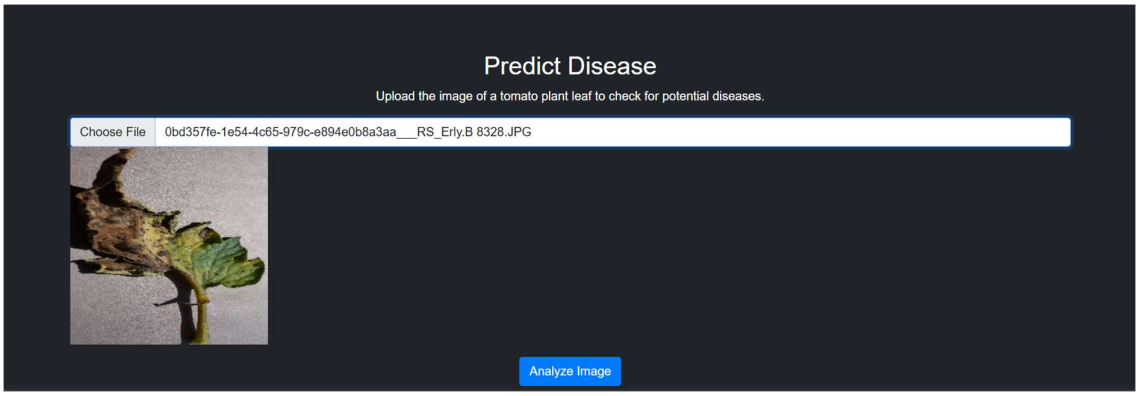
6. Result

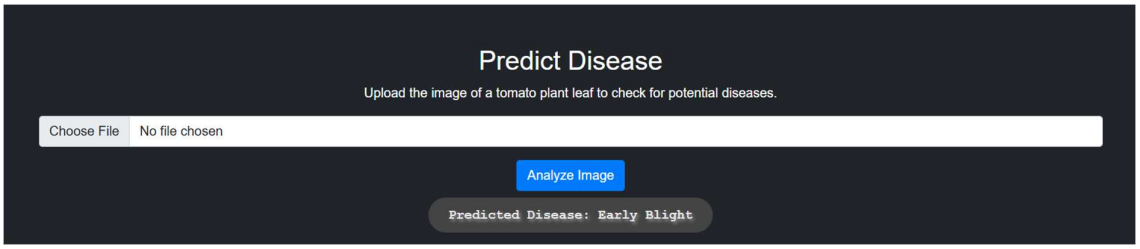
6.1 Output Screenshots











7. Advantages & Disadvantages

7.1 Advantages

7.1.1 Early Detection: The model identifies diseases at an early stage, enabling timely interventions and reducing crop loss.

7.1.**2 User-friendly:** An easy-to-use interface for farmers with minimal technical expertise.

7.1.3 **Increased Yield:** By managing the diseases effectively, it contributes to higher crop productivity.

7.2 Disadvantages

7.2.1 Data dependency: The model requires a large, diverse dataset for accurate model training.

7.2.2 Connectivity Issues: Rural areas may face challenges with internet access for cloud-based solutions.

8. Conclusion

We successfully developed a deep learning model for detecting tomato plant diseases using leaf image analysis. By leveraging Python, Flask, and Transfer Learning, the model enables early disease detection, aiding farmers in effective crop management. The integration with web applications enhances accessibility for farmers, agricultural services, and research institutions, promoting sustainable farming practices and improving crop yield. This innovative approach offers significant benefits in disease control and agricultural productivity.

9. Future Scope

1. **Model Enhancement and Expansion**: Further enhance the model by incorporating more diverse and larger datasets to improve accuracy and generalizability across different tomato varieties and environmental conditions.

2. **Real-time Monitoring and Alerts**: Integrate the model with IoT devices and mobile applications to enable real-time disease monitoring and instant alerts for farmers, facilitating timely intervention and treatment.

3. **Cross-crop Disease Detection**: Adapt and extend the model to detect diseases in other crops, broadening its applicability and providing comprehensive disease management solutions for various agricultural sectors.

10. Appendix

10.1 Source Code

10.1.1 Model Training

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

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

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

```
train_path = 'D:/WORK/Tomato Disease Prediction/5. Project Executable Files/Dataset/train'
val_path = 'D:/WORK/Tomato Disease Prediction/5. Project Executable Files/Dataset/val'
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

# Apply ImageDataGenerator functionality to Trainset and Test set

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(256, 256),
    batch_size=80,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_path,
    target_size=(256, 256),
    batch_size=80,
    class_mode='categorical'
)
```

```
base_model = ResNet152V2(weights='imagenet', include_top=False, input_shape=(256, 256, 3))

# Adding Dense Layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1000, activation='relu')(x)
```

```
predictions = Dense(10, activation='softmax')(x)

# The final model
model = Model(inputs=base_model.input, outputs=predictions)

# Configure the Learning Process
for layers in base_model.layers[:140]:
    layers.trainable=False
for layers in base_model.layers[140:]:
    layers.trainable=True

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

```
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=15
)

# Save the Model
model.save('tomato_disease_detector.h5')
```

```
model.save('/content/drive/MyDrive/Colab
Notebooks/tomato_disease_detector.h5')
```

```
# Import additional libraries for handling image input
from tensorflow.keras.preprocessing import image
import numpy as np

# Load the trained model
model = tf.keras.models.load_model('tomato_disease_detector.h5')

# Define a function to preprocess the input image and make a prediction
def predict_disease(img_path):
    img = image.load_img(img_path, target_size=(256, 256))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0

    prediction = model.predict(img_array)
    class_indices = {v: k for k, v in train_generator.class_indices.items()}
    predicted_class = np.argmax(prediction, axis=1)[0]
```

```
        disease_name = class_indices[predicted_class]

    return disease_name

# Test the function
img_path = 'im1.JPG'  # Replace with the path to your test image
disease_name = predict_disease(img_path)
print(f"The disease detected is: {disease_name}")
```

10.1.2 Flask

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

app = Flask(__name__)

# Load the trained model
try:
    model =
tf.keras.models.load_model('./Training/tomato_disease_detector.h5')
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")

# Define the class indices (update this based on your actual classes)
class_indices = {
    0: "Bacterial Spot",
    1: "Early Blight",
    2: "Late Blight",
    3: "Leaf Mold",
    4: "Septoria Leaf Spot",
    5: "Spider Mites",
    6: "Target Spot",
    7: "Tomato Yellow Leaf Curl Virus",
    8: "Tomato Mosaic Virus",
    9: "Healthy"
}

def predict_disease(img_path):
    try:
        img = image.load_img(img_path, target_size=(256, 256))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array /= 255.0
```

```
                print(f"Image shape after preprocessing:
{img_array.shape}")

        prediction = model.predict(img_array)
        print(f"Raw model prediction: {prediction}")

        predicted_class = np.argmax(prediction, axis=1)[0]
        disease_name = class_indices[predicted_class]
        return disease_name
    except Exception as e:
        print(f"Error in prediction: {e}")
        return "Prediction Error"

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

@app.route('/about')
def about():
    return render_template('index.html', _anchor='about')

@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        if 'file' not in request.files:
            return render_template('index.html', _anchor='predict', result='No
file part')
        file = request.files['file']
        if file.filename == '':
            return render_template('index.html', _anchor='predict', result='No
selected file')
        if file:
            filename = file.filename
            file_path = os.path.join('uploads', filename)
            file.save(file_path)
            disease_name = predict_disease(file_path)
            os.remove(file_path)  # Remove the file after prediction
            return render_template('index.html', _anchor='predict',
result=disease_name)
    return render_template('index.html', _anchor='predict')

@app.route('/contact')
def contact():
    return render_template('index.html', _anchor='contact')

if __name__ == '__main__':
    os.makedirs('uploads', exist_ok=True)
    app.run(debug=True, port=3000)
```

10.2.1 GitHub Link

https://github.com/SaptarshiAcharya/Tomato-Disease-Prediction

10.2.2 Demo Link

https://drive.google.com/file/d/1r_0Ip7OILrBWPzu79F-sgf3TIJUrQEWI/view?usp=sharing