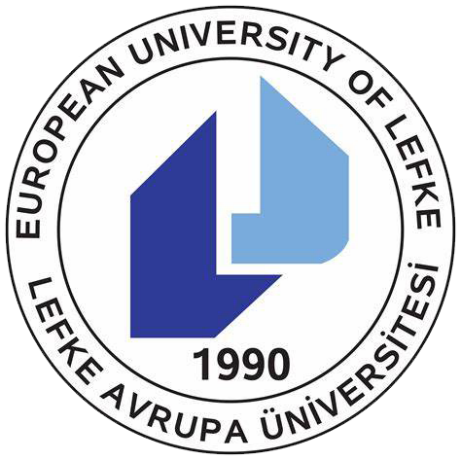
****

**EUROPEAN UNIVERSITY OF LEFKE**

FACULTY OF ENGINEERING

Graduation Project 2

Digital Farmers:

Harvesting Insights to Tackle Leaf Diseases

### Seward Richard Mupereri

### 20140175

This project uses machine learning and cloud technology, to quickly spot early signs of diseases on tomato plants. The main idea is to create a useful tool for farmers that allows them to catch plant problems early. This will help them grow better crops and use fewer pesticides. The project's main goals are to show how technology can change farming, with a focus on accurate disease detection and an easy-to-use interface.

**Supervisor**

Dr. Ferhun Yorgancıoğlu

Publish Date

29-05-2024

**Abstract**

This project titled "Digital Farmers: Harvesting Insights to Tackle Leaf Diseases" is proposed to be utilizing machine learning and cloud technology to allow early detection of diseases in tomato crops. The primary purpose is to provide an easy-to-use tool for the farmers to detect the diseases of plant health in the early stages so that better crop yields and lesser dependency on pesticides can be ensured. This report describes the methodologies used, which are data collection, model building, and web application design. The proposed solution uses a Convolutional Neural Network (CNN) trained over a dataset of tomato leaf images for the classification of different diseases. The model is deployed using FastAPI on the backend and ReactJS on the frontend to enable an interactive platform that supports the users in uploading an image as input and receiving an immediate diagnostic result. Such a solution opens the prospects of the application of advanced technology in the direction of ensuring sustainable agricultural practices, as well as enhancing the aspect of food security. The project also highlights the reflections on the challenges encountered, such as the quality of the data and the model performance, and the steps taken to address the same. Prospective works that involve further improvement, such as the expansion of the dataset and the work on a mobile application, are discussed in the direction of bettering the efficiency and accessibility of the solution.

**Table Of Contents**

[Seward Richard Mupereri i](#_Toc167750991)

[20140175 i](#_Toc167750992)

[Dr. Ferhun Yorgancıoğlu i](#_Toc167750993)

[Publish Date i](#_Toc167750994)

[29-05-2024 i](#_Toc167750995)

[1. Introduction 1](#_Toc167750996)

[1.1. Problem definition 1](#_Toc167750997)

[1.2. Goals 2](#_Toc167750998)

[2. Literature Survey 3](#_Toc167750999)

[2.1. Similar Project 1: Plant Disease Detection Using Image Processing and Machine Learning 3](#_Toc167751000)

[2.2. Similar Project 2: Plant-Disease-Detection 3](#_Toc167751001)

[2.3. Similar Project 3: Farmassist 4](#_Toc167751002)

[3. Background Information 5](#_Toc167751003)

[3.1. Required & Used Software 5](#_Toc167751004)

[3.2. Other Software 5](#_Toc167751005)

[4. Design Documents 6](#_Toc167751006)

[4.1. Data Flow Diagram (DFD) - Level 1 6](#_Toc167751007)

[4.2. Context Diagram 8](#_Toc167751008)

[5. Methodology 9](#_Toc167751009)

[5.1. Introduction 9](#_Toc167751010)

[5.2. Software Development Practices 9](#_Toc167751011)

[5.2.1. Requirements Phase 10](#_Toc167751012)

[5.2.2. Design Phase 10](#_Toc167751013)

[5.2.3. Implementation Phase 10](#_Toc167751014)

[5.2.4. Testing Phase 10](#_Toc167751015)

[5.2.5. Deployment Phase 11](#_Toc167751016)

[5.3. Data Collection and Preparation 11](#_Toc167751017)

[5.3.1. Data Sources 11](#_Toc167751018)

[5.3.2. Data Collection Process 11](#_Toc167751019)

[5.3.3. Data Preprocessing 12](#_Toc167751020)

[5.4. Model Building and Training 13](#_Toc167751021)

[5.4.1. Model Architecture 13](#_Toc167751022)

[5.4.2. Why Convolutional Neural Networks (CNNs) 13](#_Toc167751023)

[5.4.3. Detailed Description of the Model Architecture 13](#_Toc167751024)

[5.4.4. Training Process 14](#_Toc167751025)

[5.4.5. Hyperparameters Used 14](#_Toc167751026)

[5.4.6. Hyperparameter Tuning 15](#_Toc167751027)

[5.4.7. Tools and Libraries Used 15](#_Toc167751028)

[6. Model Evaluation 15](#_Toc167751029)

[6.1. Evaluation Metrics 15](#_Toc167751030)

[6.2. Performance Analysis 19](#_Toc167751031)

[6.3. Visualizations and Results 20](#_Toc167751032)

[7. Model Deployment 22](#_Toc167751033)

[8. Web Application Development 22](#_Toc167751034)

[8.1. Frontend Development 22](#_Toc167751035)

[8.1.1. Tools and Frameworks Used 22](#_Toc167751036)

[8.1.2. Frontend Components 23](#_Toc167751037)

[8.1.3. User Interface Design 24](#_Toc167751038)

[8.2. Backend Development 28](#_Toc167751039)

[8.2.1. Tools and Frameworks Used 28](#_Toc167751040)

[8.2.2. Architecture and Design 28](#_Toc167751041)

[8.2.3. Key Functionalities 28](#_Toc167751042)

[8.2.4. Endpoint Descriptions 28](#_Toc167751043)

[8.3. Integration 29](#_Toc167751044)

[8.3.1. Integration Approach 29](#_Toc167751045)

[8.3.2. Communication 29](#_Toc167751046)

[8.3.3. Process Flow 29](#_Toc167751047)

[8.3.4. Challenges 29](#_Toc167751048)

[9. Code Explanation 31](#_Toc167751049)

[9.1. main.py (FastAPI Backend) 31](#_Toc167751050)

[9.2. React Components 34](#_Toc167751051)

[9.2.1. App.jsx 34](#_Toc167751052)

[9.2.2. BodySection.jsx 34](#_Toc167751053)

[9.2.3. UploadSection.jsx 35](#_Toc167751054)

[9.2.4. NavBar.jsx 36](#_Toc167751055)

[10. Testing and Validation 36](#_Toc167751056)

[10.1. Testing Strategies 36](#_Toc167751057)

[10.2. Manual Testing: 36](#_Toc167751058)

[10.3. API Testing with Postman: 36](#_Toc167751059)

[10.4. Validation 36](#_Toc167751060)

[10.5. Iterative Improvements: 37](#_Toc167751061)

[11. Challenges and Solutions 37](#_Toc167751062)

[11.1. Challenges 37](#_Toc167751063)

[11.2. Solutions 38](#_Toc167751064)

[12. Conclusion 39](#_Toc167751065)

[12.1. Summary 39](#_Toc167751066)

[12.2. Reflection 40](#_Toc167751067)

[12.3. Benefits 41](#_Toc167751068)

[12.4. Future Tasks 41](#_Toc167751069)

[12.5. Ethics 42](#_Toc167751070)

[References 43](#_Toc167751071)

**Table of Figures**

[Figure 1: Data Flow Diagram (Level 0) - Tomato Disease Detection System 2](#_Toc167997254)

[Figure 2: Data Flow Diagram (Level 1) - Tomato Disease Detection 7](#_Toc167997255)

[Figure 3: Context Diagram - Tomato Disease Detection System 8](#_Toc167997256)

[Figure 4: Sample images from PlantVillage Dataset 12](#_Toc167997257)

[Figure 5: Training and Validation Accuracy 16](#_Toc167997258)

[Figure 6: Training and Validation Loss 17](#_Toc167997259)

[Figure 7: Confusion matrix 18](#_Toc167997260)

[Figure 8:Sample Predictions 21](#_Toc167997261)

[Figure 9: Desktop Homepage 25](#_Toc167997262)

[Figure 10:Desktop Image Upload 25](#_Toc167997263)

[Figure 11: Desktop Results 26](#_Toc167997264)

[Figure 12: Desktop Learn More about Results 26](#_Toc167997265)

# 

1. Introduction
   1. Problem definition

Tomato farming is confronted with a stiff challenge in the accurate and timely detection of diseases on tomato crops, and the problem gives rise to economic and environmental challenges. This central problem arises fundamentally because the diseases are detected at a very late juncture and only after they have done an enormous amount of damage to the crops. In turn, the farmers not only suffer an economic loss in the form of low agricultural produce but even an environmental loss due to the unnecessary use of pesticides. My project presents an innovative answer to this question. My solution lies in the use of machine learning and cloud computing, in order to revolutionize the disease detection process in the domain of tomato farming. The project takes a simplified approach:

* **Data Cleaning and Preprocessing**: Fusing, data cleaning, and preprocessing of a large-scale dataset of healthy and diseased tomato leaves, and data cleaning and preprocessing using environments like Jupyter Notebook and Python (Keras API) to generate a standardized and preprocessed dataset that can be used for machine learning analysis.
* **Model Building and Training**: Application of Convolutional Neural Networks (CNNs) as part of the TensorFlow framework for the training of a model that can accurately classify healthy and diseased tomato leaves.
* **Model Deployment**: Deployment of the trained model on Google Cloud using TensorFlow Serving to provide users the ability of interacting with the model using a web-based interface.
* **Web-Based Application**: Construction of a user-friendly web application that provides users, even without a good technical background, the ability of easily uploading images of their tomato plants and receiving an instantaneous diagnosis of diseases.

**Example Problems:**

* Accurately detecting disease early to void losing crop yield.
* Minimizing losses but ensuring a high yield.
* Use of less pesticides as disease does not spread easily.
  1. Goals

There are certain objectives that my project is going to cater to, and they are as follows:

* **Early Disease Detection**: Ensure easy and accurate detection of disease in leaves of tomato plants so that correctives can be applied from the end of the farmers.
* **Sustainability**: Support environment-friendly farming practices by reducing the dependence on pesticides.
* **User-Friendly Interface**: Ensure that the technology solutions are available to all, irrespective of the level of technical know-how.

**Detailed Goals:**

* **Early Disease Detection**: My system, with the help of which disease can be detected in the crop at a really early stage, will lead to less usage of pesticides and provide for sustainable agriculture as well as a lower environmental footprint
* **Sustainability**: By detecting early, the diseases by my project will help decrease the usage of pesticides and thereby help foster sustainable agriculture as well as lessen the environmental impact.
* **User-Friendly Interface**: I recommend a user-centered web application that shall accept the uploaded images of the tomato plants by the farmer and provide an immediate report on the condition of the plants irrespective of the level of techno-competency of the farmer.



Figure 1: Data Flow Diagram (Level 0) - Tomato Disease Detection System

1. Literature Survey
   1. Similar Project 1: Plant Disease Detection Using Image Processing and Machine Learning

This was the first project I compared my project to. This project uses computer vision and machine learning to identify diseases on plant leaves. (Kulkarni, 2021)

**Comparison:**

**Scope:** This project is capable of detecting 5 different plant leaves, whereas my project will have a focus on detecting disease on just tomato leaves. Focusing on just one plant allows me to have my focus on a simpler model allowing me to fine-tune the model really well to achieve higher accuracies.

**User Experience**: This project was developed using a python framework, Flask while my project was developed using FastAPI which has features which make it better when working with APIs with a superior runtime compared to flask. (Turing, 2024)

**Conclusion:** My projects focus on just the tomato plant and the use of different tools sets my project apart.

* 1. Similar Project 2: Plant-Disease-Detection

This project uses PyTorch and a convolutional neural network to predict disease of any kind into 39 different categories This project was published on Github. (Bhikadiya, 2022)

**Comparison:**

**Scope:** This project aims to be able to detect 39 different diseases compared to mine which will only predict 10 diseases which means I might be able to have a less complex model meaning any fine tuning or debugging would be easier.

**Methodology:** The machine learning model for this project was developed using PyTorch while my project was developed using Tensorflow which excels performs better in production environments and PyTorch excels in research and dynamic projects. (Alvi, 2024)

**Conclusion:** While both of our projects used python comparing to this project allowed me to be able to compare two different approaches comparing to successful project..

* 1. Similar Project 3: Farmassist

The last project I compared to what also published to GitHub. This project was more of a smart farming app that uses an IoT and AI—powered disease detection tool to help farmers on the field. It was developed as a Flutter App with a Firebase backend. It has a farm management tool with collected data and monitoring systems. (Jasson, 2021)

**Comparison:**

**Scope**: This project detects diseases of quite a number of various plants and includes functionality like monitoring systems and farm management whereas my project just focuses on tomato diseases.

**Platform**: This project was developed for mobile devices where as my application is a web application which can be easily used on both mobile and desktop. Which is more accessible.

**Conclusion**: This project had various features that mine does not have and it had a focus on data collection and monitoring when my project just focused on accurate predictions.

1. Background Information
   1. Required & Used Software

To successfully develop this project, I had to choose a set of tools and software that will help me do that. Below is a list of all the software which were used during the development of this project.

* **Data Collection:**
  + **Kaggle:** The dataset I used t train, validate and test the model where from Kaggle Datasets which came pre-processed.
* **Programming Languages:**
  + **Python**: Tis was the main language I used for the model building and fine tuning.
  + **JavaScript**: JavaScript is the programming language for React which was used to develop the user interface.
  + **HTML & CSS**: These languages are supporting languages for the React application.
* **Model Training and Tuning:**
  + **TensorFlow (Python):** For the machine learning the framework I used was Tensorflow.
* **Model Deployment:**
  + **Render:** The model, backend and frontend were all deployed on Render to ensure scalability, availability, and reliable performance.
* **Web Development:**
  + **FastAPI (Python):** For the web application the backend was developed using the FastAPI web framework in Python.
  + **React**: I used react to develop a user-friendly and responsive user experience.
* **Version Control:**
  + **Git**: I used Git for code management and version control.
  1. Other Software

Below are other software which were used in the project:

* **Vector Graphic Editing and Design Software:**
  + **Adobe Illustrator:** All graphic work and design were designed using Adobe illustrator.
* **Code Editor:**
  + **Visual Studio Code:** All the coding and commands were done in the VS Code IDE.
  + **Jupyter Notebook**: I used Jupyter Notebook for all for all the model training and fine-tuning.

1. Design Documents
   1. Data Flow Diagram (DFD) - Level 1

Figure 2: Data Flow Diagram (Level 1) - Tomato Disease Detection shows how the data will flow across the different components of the disease detection system.

**Processes and Data Flows:**

1. **Image Upload (by Farmers):**

* The first step is a farmer uploads an image on the web application.
* **Data Flow:** The web application receives the image and sends it to the FastAPI backend.

1. **Image Preprocessing:**

* The FastAPI backend receives the image and reads it into memory and converts it to a format that the model will be able to receive as input.
* **Data Flow**: The formatted image data is then sent to the model for processing.

1. **Disease Detection:**

* The TensorFlow model processes the image data to make a prediction on the image.
* **Data Flow**: The results from the processing are generated.

1. **Result Generation:**

* The results from the prediction model are then formatted to a format that the user will be able to easily understand.
* **Data Flow**: The new formatted response is then sent back to the frontend.

1. **User Feedback:**

* The results are then displayed back to the user using the web interface.

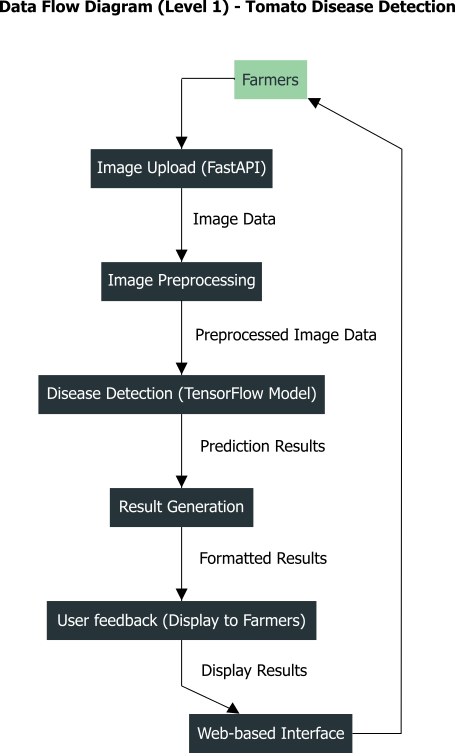
****

Figure 2: Data Flow Diagram (Level 1) - Tomato Disease Detection

* 1. Context Diagram

**Components:**

1. **System (Tomato Disease Detection System):**

* This component includes all the internal processes of the system like image upload, preprocessing, disease detection, result generation and the user feedback.

1. **External Entities:**

* Farmers
* Cloud Storage (Render)
* Machine Learning Model (TensorFlow Model)
* Web Interface

**Interactions:**

1. **Farmers:**

* Upload images of tomato leaves to the system.
* Receive a prediction on the leaf from the system.

1. **Cloud Storage (Render):**

* Stores the mode, backend and frontend servers.
* Hosts the whole production project.

1. **Machine Learning Model (TensorFlow Model):**

* Detects the leaf disease based on the uploaded image.
* Returns the prediction results for the user.

1. **Web Interface:**

* Allows farmers to interact with the system.
* Displays prediction results to the user.

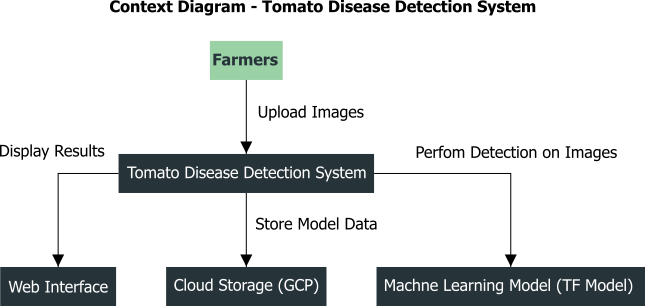


Figure 3: Context Diagram - Tomato Disease Detection System

1. Methodology
   1. Introduction

This project is approached and developed with a goal of creating a successful and user-friendly approach toward a systematic and structured tomato disease detection system. The description of the methodologies and approaches taken ensures that modern tools and frameworks, which have gained immense respect in the software development, are used in this case.

In the explanation below is the detailed application of methodologies and the importance of the process to ensure the project is successful. The project methodologies are varied, right from machine learning for detecting diseases, preprocessing their data to enhance images, and model training and evaluation to web development for interactive purposes to develop a user interface.

The model was developed using TensorFlow, for its robustness and widespread use in machine Learning with a FastAPI backend. For the frontend I chose to use React as it is efficient in building very dynamic and very responsive, we application easily. The combination of these tools demonstrates a full stack approach in the development phases. (Prakash, 2023)

The phases are:

1. **Requirements**: Understanding what exactly needs to be done and documenting it.
2. **Design**: Having a detailed plan on how on development and design for the system.
3. **Implementation:** Creating all the necessary code files with the relevant code written.
4. **Testing**: Running tests on the code to ensure that it meets all the requirements and does what it is supposed to do.
5. **Deployment**: Deploying the backend and frontend servers to a live environment.

This methodology section provides a guide that is clear and comprehensive for the development process. It explains how each stage of development was done and explains the decisions made to make sure that the project’s goals were achieved efficiently. It also acts as a blueprint for others who wish to replicate or work on this project. Having a structured approach to development ensures a smooth and more efficient project lifecycle. (Denham, 2013)

* 1. Software Development Practices

**Overview**

The Waterfall model has a methodical way and a plethora of advantages; that's why it was selected for this project. For example, it has clear documentation, easiness in tracking progress, and well-defined development levels, which are helpful for a single-person developer. This makes it simple for me to know what has been done and what is remaining to be achieved in every single stage of development. (Corrales, 2022)

**Advantages of the Waterfall Model:**

* **Simplicity and Ease of Use**: Each phase has unique deliverables that, in conjunction with the review process, enable one to follow it easily.
* **Structured Approach**: A sequential flow provides one with a structured approach, and therefore, it makes one systematically approach any of these phases.
* **Clear Documentation**: Documenting at every stage helps one understand how well the project is moving forward.
* **Simplified Progress Tracking**: It becomes a piece of cake to know what is going on with the project and at what point.
* **Appropriate for Solo Development**: Its linear nature makes most of the things manageable for a sole developer.
  + 1. Requirements Phase

I was drawn to this topic by machine-learning social media content that demonstrated how people used machine learning model to better their day to day lives. I was driven by the fact that I have a great interest in agriculture and I wanted to find a problem that a machine learning model could solve in this sector. I chose to system that uses a machine learning model to detect diseases in the tomato plant using a curated and well prepared dataset to make the development stage faster as the project had a time constraint.

* + 1. Design Phase

Architecture design for the system was achieved using a modular approach to allow good separation of concern and easy maintainability. Some of the key design considerations taken include:

1. **Architectural Style:** The system was architected based on a client-server architecture. (Nishtha, 2024)
2. **Modular Design:** The project has been broken into individual, independent modules that include data preprocessing, model training, building, and the web application as modules.
3. **Technological Stack:** The stack to be used is TensorFlow for the best machine learning, the React frontend, and FastAPI on the backend side for the most stable and practical work.
   * 1. Implementation Phase

This represents the actual coding and development of the system components. Key activities:

* **Coding and Development:** The elaborate design was developed using the best tools, modern frameworks, and detailed design from my research.
* **Version Control:** Using Git with Bitbucket allowed for me to track my project and have my code managed and backed up well.
* **Utilization of Tools and Libraries:** The model was developed using TensorFlow and connected to a FastAPI backend which communicated with a React frontend for a responsive user interface.
  + 1. Testing Phase

At this stage, it was to establish validation that the system meets the desired standards and functions the way it is supposed to. There are applied different methodologies of testing:

* + - * 1. **Unit Tests:** Here, various elements are tested for the proper performance.
        2. **Integration Tests:** It tests the proper functioning of the different components of the system when integrated.
        3. **System Tests:** This test checks that the requirements are supported by the entire system.
        4. **Model Evaluation:** The model was evaluated on a test set for performance, and set measures such as accuracy and loss are recorded. (Chollet, 2023)
        5. **Visualization:** The plotted results graphically while assessing the predictions made by the respective model for the accuracy and reliability of the predictions.
    1. Deployment Phase

This phase of deployment was all about the release of the system into the live environment. Major activities were as follows:

* + **Server Setup**: Preparation of servers that will host the application.
  + **Environment Configuration**: Any setup elements in the environment that made the application run smoothly.
  + **Accessibility**: The system should be accessible to the users with negligible downtime.
  1. Data Collection and Preparation
     1. Data Sources

The data for this study has been obtained from Kaggle and is under the dataset called "PlantVillage." (EMMANUEL, 2018) The dataset contains plant leaves affected by various diseases, but the focus is more on the leaves of tomato, potato, and pepper plants. The images of a potato and pepper plant were not included since my project was expected to familiarize the model regarding disease identification in tomato plants. I chose this dataset because of its high quality and detailed labelling which are very important for a good dataset..

* + 1. Data Collection Process

The data was collected by downloading the dataset from Kaggle and organizing the images according to the type of disease they represent. Then, the folders that were part of the dataset on tomato diseases were:

* *Tomato\_Bacterial\_spot*
* *Tomato\_Early\_blight*
* *Tomato\_Late\_blight*
* *Tomato\_Leaf\_Mold*
* *Tomato\_Septoria\_leaf\_spot*
* *Tomato\_Spider\_mites\_Two\_spotted\_spider\_mite*
* *Tomato\_\_Target\_Spot*
* *Tomato\_\_Tomato\_YellowLeaf\_\_Curl\_Virus*
* *Tomato\_\_Tomato\_mosaic\_virus*
* *Tomato\_healthy*

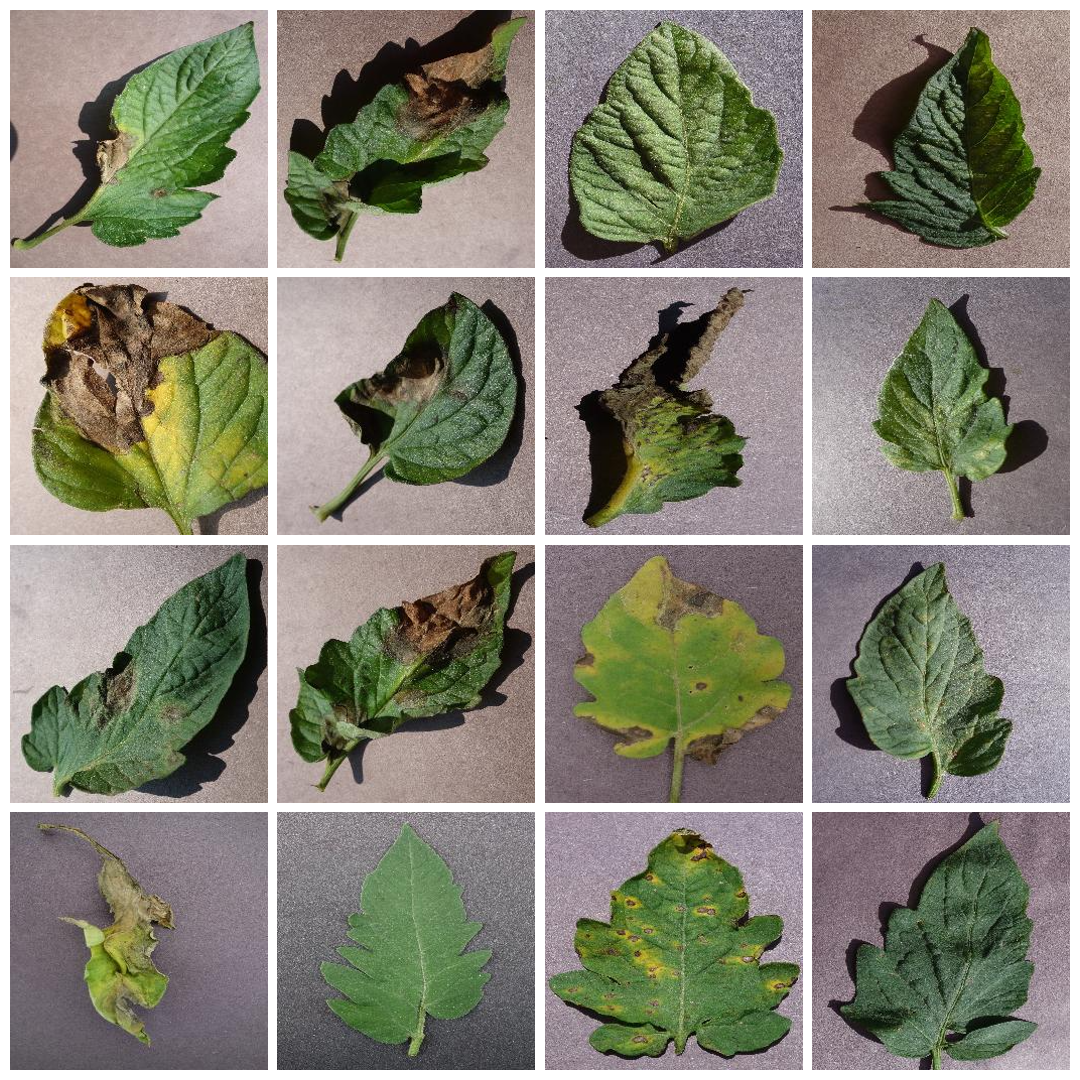


Figure 4: Sample images from PlantVillage Dataset

Each folder contained images labeled with the corresponding disease, facilitating the training process by providing clear and structured data. In total, there were approximately 18,160 images of tomato leaves used for training the model.

* + 1. Data Preprocessing

**Normalization**

Normalization methods are data preprocessing procedures for making the data uniform. This is performed by the general process of sizing down all the images to a standard size of 256x256 pixels and scaling pixel values to lie between 0 and 1. Having consistent input data being fed to the model allows for improved performance and accuracy in the machine learning model. (Agarwal, 2020)

**Data Augmentation**

I increased the size of the dataset using basic techniques like rotation, flipping, and color adjustment, among others. For every image, several multiples are taken. The model could, therefore, generalize more for new data points not seen. There would also be much lower chances of overfitting. (AWS, 2024)

**Data Partitioning**

The dataset is divided into the training, validation, and test sets. The training set trains the model, hyperparameters are tuned on the validation set, with the possibility of evaluating the model's performance in intermediate steps and, finally, the test set. Care is taken so that the train, validation, and test sets are representative and balanced. The current dataset in this research work was split into the training, validation, and test sets at a ratio of 70%, 20%, and 10% of the data, respectively. The training dataset itself was augmented, which might enhance the robustness. The present original unbiased images were further used to create augmented validation and test data sets to evaluate the CNN model. (Igareta, 2021)

* 1. Model Building and Training
     1. Model Architecture

The model architecture was carefully designed to take advantage of the strengths of Convolutional Neural Networks (CNNs), which are well-suited for image classification tasks like this project where I am classifying tomato leaves. CNNs perform well for tasks on recognizing patterns or features in the images since they work on learning spatial hierarchies. The architecture chosen included many convolutional layers and was followed by max-pooling, then fully connected to proceed toward the final output. (Mandal, 2024)

* + 1. Why Convolutional Neural Networks (CNNs)

The reason for choosing CNNs is that they perform well in tasks related to image recognition. They form part of high-accuracy schemes for many problems in the field of computer vision, so their ability makes them efficient users in the detection of tomato disease from leaf images. CNNs use this hierarchies of features during deep learning to help the model to be able to identify extremely complex patterns in images, which are helpful in discriminative identification of different diseases. (Lang, 2021)

* + 1. Detailed Description of the Model Architecture

The model architecture consists of the following components:

1. **Input Layer**: The input layer receives images of size 256x256 with 3 color channels (RGB).
2. **Normalization and Data Augmentation**: Resizing, and rescaling, among other types of image augmentation.
3. **Convolutional Layers**: Stacked on the convolutional layers are many layers, which are then fitted with an activation function that extracts the features from that image.
4. **Max-Pooling Layers**: It reduces the size of feature maps so that when the dimensions of feature maps are significant, that reduction in spatial dimensions prevents overfitting the network.
5. **Flatten Layer**: A flatten layer to convert the 2D feature maps into a 1D vector.
6. **Fully Connected Layers**: Dense layers with ReLU activation functions for classification. (Dremio, 2024)
7. **Output Layer**: A dense layer with a softmax activation function to output the probabilities of each class (disease type). (Oppermann, 2024)
   * 1. Training Process

Proper training and optimization of the model needs a few important steps.

Steps Involved in Training the Model

1. **Data Preparation**: Load and preprocess images, normalizations, and augmentations.
2. **Model Compilation**: A model has to be compiled with an optimizer and a loss function along with the evaluation metrics. The model was compiled with the Adam optimizer, Sparse Categorical Cross entropy loss function, and accuracy as the performance metric. (Kitchell, 2022)
3. **Training**: Train that model to that predefined training dataset, using validation data so the model can monitor the performance toward making adjustments.
4. **Evaluation**: Evaluate the model using the test set to have the final performance of the model.
   * 1. Hyperparameters Used

Hyperparameters used in the training process:

* **Learning Rate: 0.001**
  + **Why this value:** This value of the learning rate—0.001—can be considered a default starting point for most models. The model learns at a decent pace, and this learning rate does not shoot over the minima of the loss function.
  + **Effect on training:** Too low a learning rate slows training hugely and, at worst, stalls it at local optimums; too high a learning rate: can learn fast convergence, but it might be so aggressive that it can skip global optimums. (Brownlee, Understand the Impact of Learning Rate on Neural Network Performance, 2020)
* **Batch Size: 32**
  + **Why this value**: This would be the optimum value that best balances memory efficiency and training speed. Allow many updates in the gradient to fit in the GPU memory.
  + **Effect on training**: Low batch size updates more frequently; hence, it can be more robust concerning its convergence but is noisy. Larger batch sizes make the training occur faster but can result in less stable training. (Devansh, 2022)
* **Number of Epochs: 25**
  + **Why this value**: Training for 25 epochs is generally sufficient to observe the model's learning trends and make necessary adjustments without overfitting.
  + **Effect on training**: Training for too few number of epochs might lead to underfitting, where the model hasn't learned enough from the data. Training for too many epochs can lead to overfitting, where the model learns the noise in the training data and kind of memorizes the images and classes. (Rsvmukhesh, 2023)
* **Optimizer: Adam**
  + **Why this optimizer**: Adam is one of the most popular optimizers that comes with properties of an adaptive learning rate and does well while working with sparse gradients for noisy problems.
  + **Effect on training**: The Adam optimizer adjusts the learning rate for each parameter individually, which can lead to fast and robust convergence compared to straightforward gradient descent. (Brownlee, Understand the Impact of Learning Rate on Neural Network Performance, 2020)
    1. Hyperparameter Tuning

Hyperparameter tuning was performed on the optimization of the learning rate, optimization of the batch size, and optimization of the number of epochs. It evaluates the search spaces by using grid and random searches, which are the two most valid ways to get the best hyperparameters and ensure excellent performance of the model. (Jordan, 2017)

* + 1. Tools and Libraries Used

The following tools and libraries were used to build and train the model:

* **TensorFlow**: A library to build and train the neural network.
* **Keras**: A high-level library to build the model structure and make the training a little more comfortable.

1. Model Evaluation

Model performance was computed on the test set. The evaluation comprises losses, accuracies, and predictions of the model for new data, which are then visualized concerning the model's performance in terms of accuracy and reliability.

* 1. Evaluation Metrics

To assess the performance of the model, several evaluation metrics were used. These metrics provide a comprehensive understanding of the model's accuracy and overall effectiveness and reliability in classifying tomato plant diseases.

1. **Accuracy**

These provide a simple measure for the overall performance of the model where accuracy represents the proportion of true positive and accurate negative results to the total amount of cases analysed. (D, 2019)

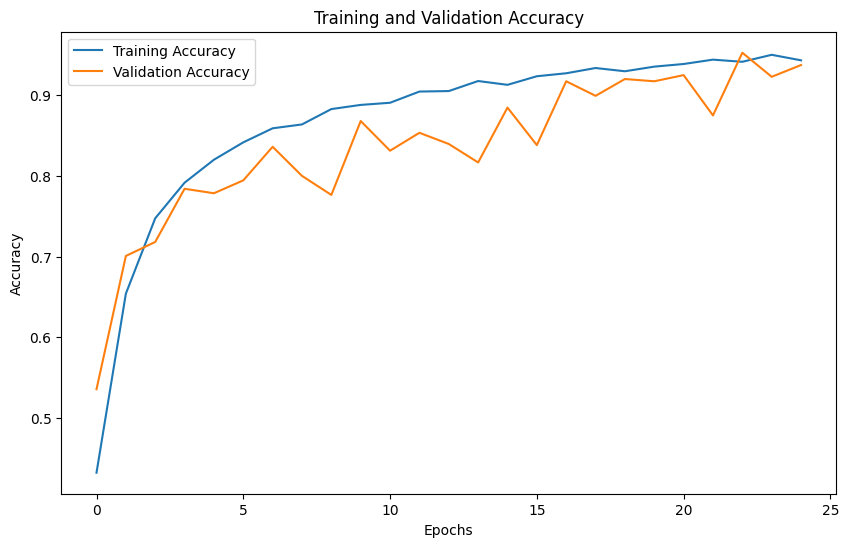


Figure 5: Training and Validation Accuracy

**Interpretation**

As we can see from the plot, the accuracy of the training and validation increased monotonically up to a little more than 90%. Hence, the model learned the relevant features in the data and could generalize the same relevant features to new data, as evidenced by the relatively high validation accuracy..

1. **Loss**

Loss measures how good or how bad, regarding actual data, the prediction of a model is. The less the loss, indicates good performance.



Figure 6: Training and Validation Loss

**Interpretation**

it is clear that the training and validation losses were decreasing ideally as a function of training epochs. This indicates that the model has been improving its prediction during the whole training period to represent the target more perfectly. Similar to training loss, the score tracks very close to the loss. That implies the model does not overfit.

1. **Confusion Matrix**

A confusion matrix is the explicit and clear-cut representation of model performance in the actual and predicted classification. One can quickly notice the weak classes, categories, or groups when the performance is put in an actual versus predicted classification. (Narkhede, 2018)

**Confusion Matrix**

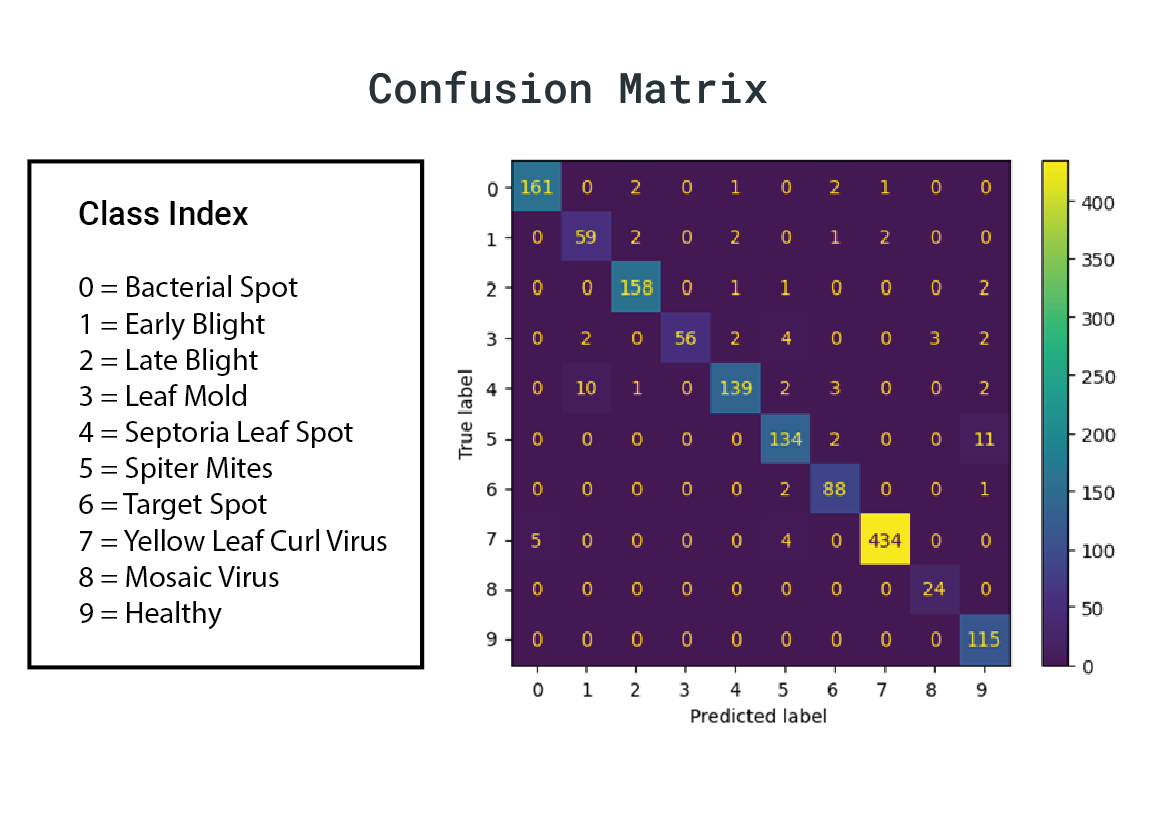


Figure 7: Confusion matrix

**Interpretation**

1. **Diagonal Dominance**: The diagonal elements of the confusion matrix give the number of correct predictions of each class. The very high values in the diagonal indicate the excellent performance of the model in classifying the classes.
   * **Bacterial spot—**161 incorrect predictions, with the minimum number of misclassifications.
   * **Early blight—**59 incorrect predictions, with a lower number of misclassifications**.**
   * **Late blight—**158 incorrect predictions, with very few misclassifications**.**
   * **Leaf Mold—**56 incorrect predictions, with a low number of misclassifications.
   * **Septoria leaf spot—**139 incorrect predictions, with a lower number of misclassifications.
   * **Spider Mite:** 134 true positives and a very small number of false negatives.
   * **Target Spot:** 88 true positives and very low false negatives.
   * **Yellow Leaf Curl Virus:** 434 true positives but deficient number of false negatives.
   * **Mosaic Virus:** 24 true positives and zero false negatives.
   * **Healthy:** 115 true positives and zero false negatives.
2. **Off-Diagonal Elements**: These elements are off the diagonal. Any presence here will account for a misclassification error; hence, low values here will indicate good performance.
   * Overall, over most classes, there are fewer misclassifications.
   * **Yellow Leaf Curl Virus**: Very few are being misclassified as Tomato\_\_\_Bacterial\_spot(5).
   * Between Leaf Mold, and Septoria leaf spot, i.e., 2 and 10 misclassifications.).
3. **Class-Specific Analysis**:
   * **Bacterial spot:** Low misclassifications and 161 correct predictions.
   * **Yellow Leaf Curl Virus:** The maximum actual predictions from this class are, i.e., 433, which means that the model was highly influential in classifying this disease.
   * Mosaic Virus and Healthy classes are almost classified correctly with almost zero misclassifications of importance.
4. **Confusion Between Specific Classes**:
   * Leaf Mold and Septoria leaf spot are confused with each other slightly. This might be perhaps due to symptom mimicry of both classes with each other.
   * There is some confusion about the Yellow Leaf Curl Virus with Bacterial Spot, that is to say, that classes two and three may have some visually similar features, resolved by focused improvement overall; the model works well according to the confusion matrix, especially the class Yellow Leaf Curl Virus and with focused improvement, accuracy will be even more.
5. **Cross-Validation**

Cross-validation was done to check the stability of the model. This can be done by randomly partitioning the complete dataset into subsets and training different models with the help of these subsets. Once the model is trained on subsets, the performances are averaged to arrive at a productive estimate of the model.

1. **K-Fold Cross-Validation**

When K-fold cross-validation is applied randomly, the data is partitioned as a whole into K equal folds. Therefore, the algorithm is trained on K-1 of the folds and tested on the remaining data. This is done for each of the K folds, where one fold tests all and averages the results overall for evaluation. (Brownlee, A Gentle Introduction to k-fold Cross-Validation, 2023)

* 1. Performance Analysis

**Presentation of Results**

In this section the results of the model on the test set with the evaluation metrics were shown. The purpose of this evaluation is to showed the tomato plant diseases model ingesting without error in-classify.

**Analysis and Interpretation**

The evaluation results revealed how well the model performed in comparison to the test set when it came to accuracy and loss. Data Augmentation, OPL techniques (oversampling/undersampling) and proper Hyper parameter tuning on well designed network all trained on Keras functional API gave glaciers how to improve accuracy and loss\_Function=device) that signaifies it gave good results for both the measure with respect to the k-fold range. It could be conclusively stated that the model largely generalized on the data which it never had seen before (on which the model was deployed) and thus good to go for real life use case in tomato plant disease prediction.

* 1. Visualizations and Results

**Sample Predictions**

The displays a few test sample predictions made by the model using different data generation blocks with a resolution of 1024x1024. For each image it will display the class that was actually and which was predicted along with the confidence level with which it was predicted. These visualizations assist in interpretation of the predictive power and consistency of the model..

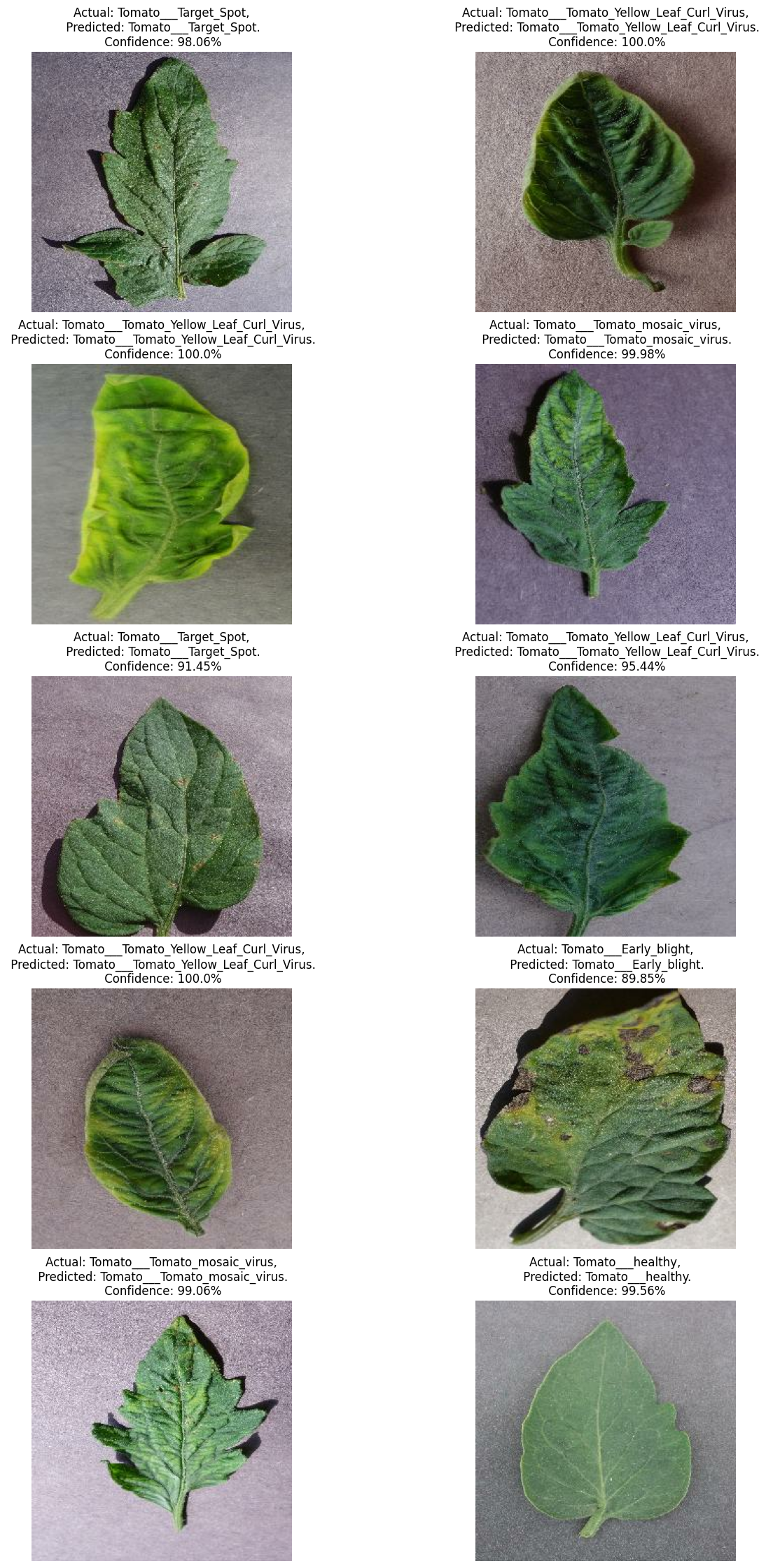


Figure 8:Sample Predictions

1. Model Deployment

Our model was deployed on Render (Cloud Platform), a modern cloud provider that lets you code in almost any language and provides a web services deployment environment with horizontal scaling. I choose Render for its ease of use, low cost, and an efficient, highly available infrastructure that can run at very high performance.

**Why Render?** Render makes rolling out ML models and web applications a pleasant experience with CI/CD, AA and global CDN. Even better, Render supports a range of runtimes and languages so it works effortlessly with our serverside API (FastAPI) and model (TensorFlow).

**User-Friendly Interface:** Render's user-friendly interface makes the deployment process straightforward and efficient and easy for a beginner. The platform offers clear and detailed instructions at every step, from setting up the environment to deploying the application. This includes:

* **Intuitive Dashboard:** The dashboard that Render provides is very straightforward and allows the user to manage their deployed applications, databases and other services all in just one place.
* **Automatic Build and Deploy:** I just had to link my BitBucket Repo and Render took over and build and deployed the services for me. This took away a lot of complexity when deploying.
* **Environment Configuration:** the platform takes care to create the environment variables and secrets in a simple way, so that the sensitive data are securely managed.
* **Comprehensive Documentation:** Render has a comprehensive documentation that is well explained which includes tutorial and examples to help users create and deploy their applications in no time.

**Deployment Process:** Deployment on Render is reading the instructions on the platform, making sure that my environment is properly set up and making sure that all libraries were implemented correctly. This involved:

* **Setting Up the Project:** setting up a new project & connecting it to a BitBucket repository.
* **Configuring the Environment:** Using Render's intuitive interface to set environment variables and configure the deployment environment.
* **Deploying the Application:** Deploying the FastAPI backend and TensorFlow model using Render auto-build and deploy. Render's automated scripts and build processes made this a frictionless process, with minimal manual intervention.

And in the case of the deployment described above simply choosing Render made it efficient and hassle-free: due to its powerful features, and user-friendly interface. This not only successfully deployed the application but also maintained its performance, high availability, giving a smooth experience to both the developer and users.

1. Web Application Development
   1. Frontend Development
      1. Tools and Frameworks Used

The fronted of this project was build using ReactJS library of JavaScript and with component library fro materialUI. MaterialUI offered Google's Material Design components out of the box which made it very fast to code and looked visually nice. These prebuilt components made it user-friendly and responsive user experience which not only good in designs but also the predictions were quite informative to the user.

* + 1. Frontend Components

***App.jsx***

The App. It is the root component of a jsx component It lays out the foundational structure such that the global CSS baseline is included for uniform styling on these browsers. This component consists of two other components: NavigationBar (NavBar) and BodySection, the latter is the main area of the screen.

* **CssBaseline:** Provides baseline stylesheet.
* **NavBar:** To render navbar where all site wide navigation happens.
* **BodySection:** This component will be for the main content area of the application.

***BodySection.jsx***

The BodySection. This is JSX component which is dealing with major widget body area. Creates a visual design layout where the background is styled with the image and contains the UploadSection component where the user can upload images and see the results.

* **Background Styling:** Will give background styling with image fetching fixed background image.
* **UploadSection:** Renders the UploadSection component for uploading an image and its results.

***NavBar.jsx***

The NavBar. The below one is jsx component which works as navigation bar for the application. It includes links to different parts of the application and external resources and uses the Material-UI responsive utilities appropriately for screen sizes.

* **Responsive:** Material-UI's responsive utilities easily keep the app fit for every screen size.
* **Navigation Links:** Take you to Home, Development and Contact pages.
* **Theming and Styling:** Uses Material-UI for consistent theming and styling.

***UploadSection.jsx***

The UploadSection. JSX component that manages an image upload as well as displays the processed result. Allows you to do three things — upload an image to be analyzed, preview the uploaded image, submit image for prediction, and see the detailed results. Moreover, it consists of "Learn More" part which describes all the disease case..

* **Image upload:** The functionality that allows users to upload images for analysis.
* **Image Preview:** This will display uploaded image preview.
* **Submit for Analysis:** It sends the image to the backend for prediction and shows the results.
* **Display the result:** The disease of interest is shown along with its probability of being that type of disease and further details of the disease.
* **LEARN MORE:** Explain the identified disease in more details — causal agent, symptoms, distribution, and control measures.
  + 1. User Interface Design

During the development of the user interface the focus was ion, functionality, aesthetic appeal, consistency, accessibility and responsiveness

**Design Principles**

1. **Functional**: The design is intended to allow users to understand the application, move between tasks, and focus on work rather than the design.
2. **Aesthetic Appeal**: Visual elements were hand-picked to make it look sharp and professional, such as MaterialUI components and custom styling.
3. **Consistency**: The goal was to maintain a consistent brand look through our the user interface.
4. **Accessibility**: The web application was designed to very accessible on almost any device that has a browser and can upload images to a page.
5. **Responsiveness**: There was an emphasis on making sure that the web application is responsive and works seamlessly on any device.

**User Experience Considerations**

1. **Ease of Use**: The web application needed to be very easy an intuitive with clear instructions and feedback for the users.
2. **Visual Feedback**: The web application needed to provide clear and useful feedback to the user.
3. **Performance:** The web application was designed to perfume really well and not letting the user have a bad experience when they need a prediction.
4. **Navigation**: A clear menu for navigating the web application was a priority to allow users to navigate the web application.

**Screenshots of the Web Application**

**Home Page**

The home page introduces the tool and allows users to upload an image for analysis.

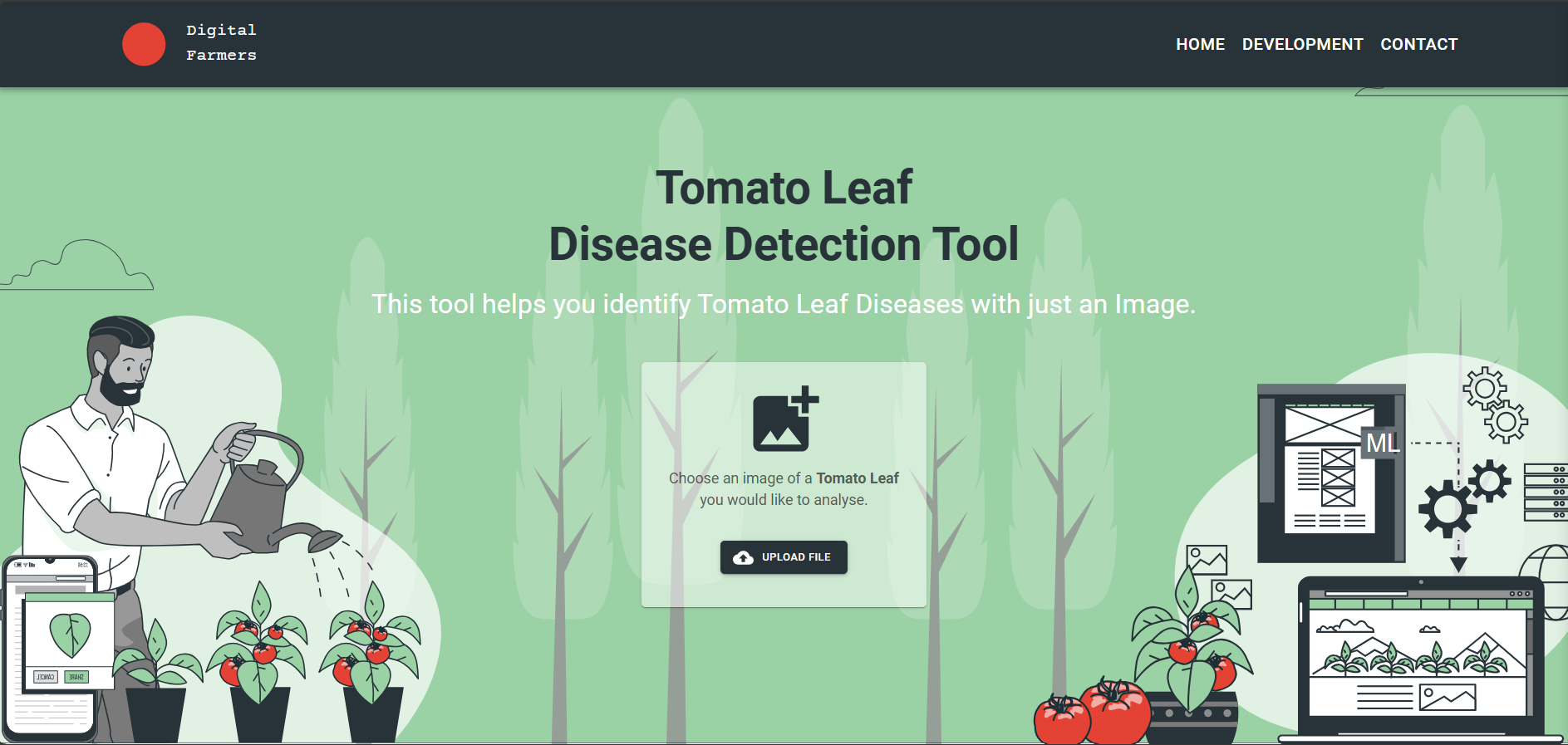
****

Figure 9: Desktop Homepage

On the home page once the user selects an image to upload, they are shown a preview before they submit it for analysis

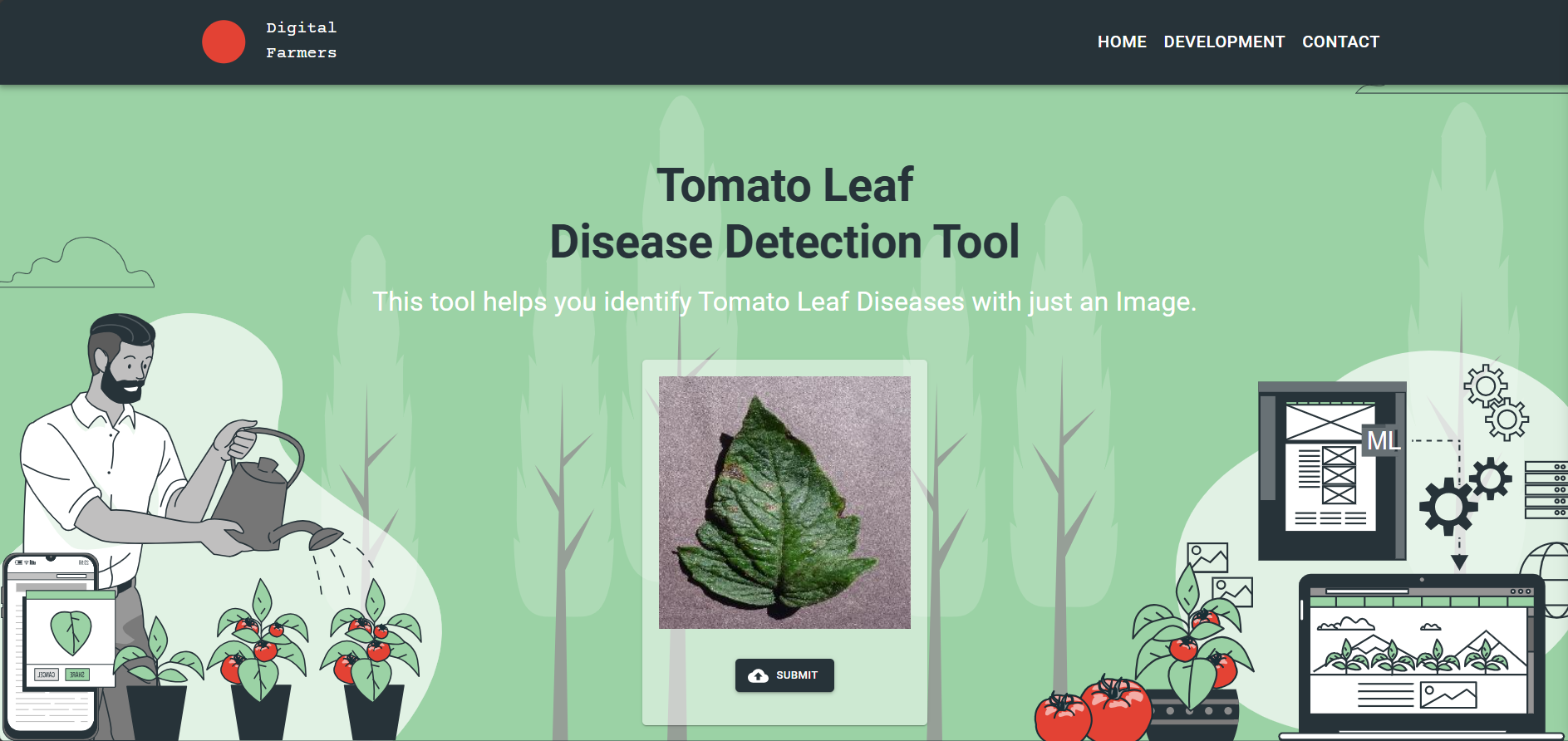
****

Figure 10:Desktop Image Upload

**Results Page**

The results page displays the predicted disease, confidence level, and offers a "Learn More" button for detailed information.

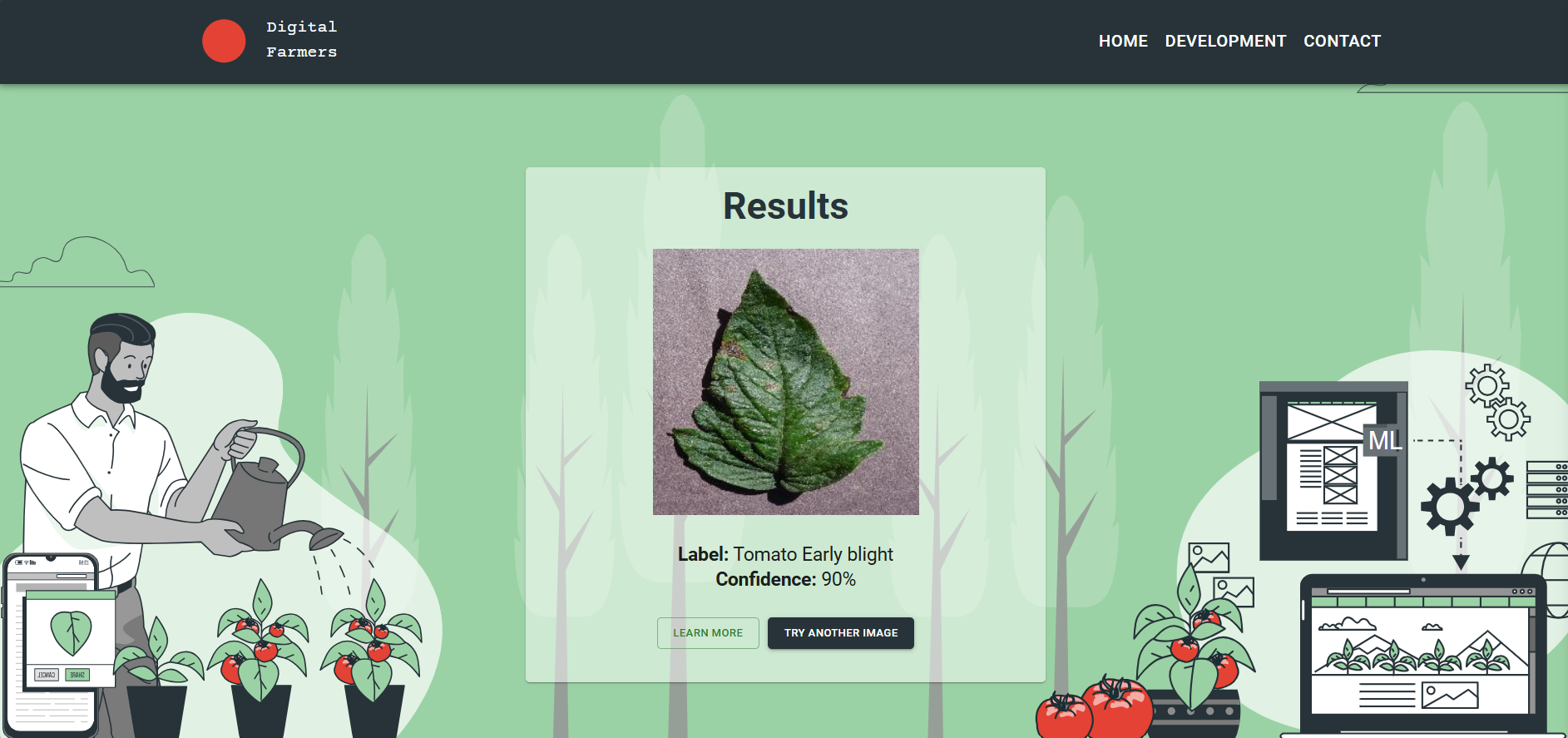
****

Figure 11: Desktop Results

**Detailed Information Page**

The detailed information page provides comprehensive details about the detected disease, including the causal agent, distribution, symptoms, conditions for development, and control measures.

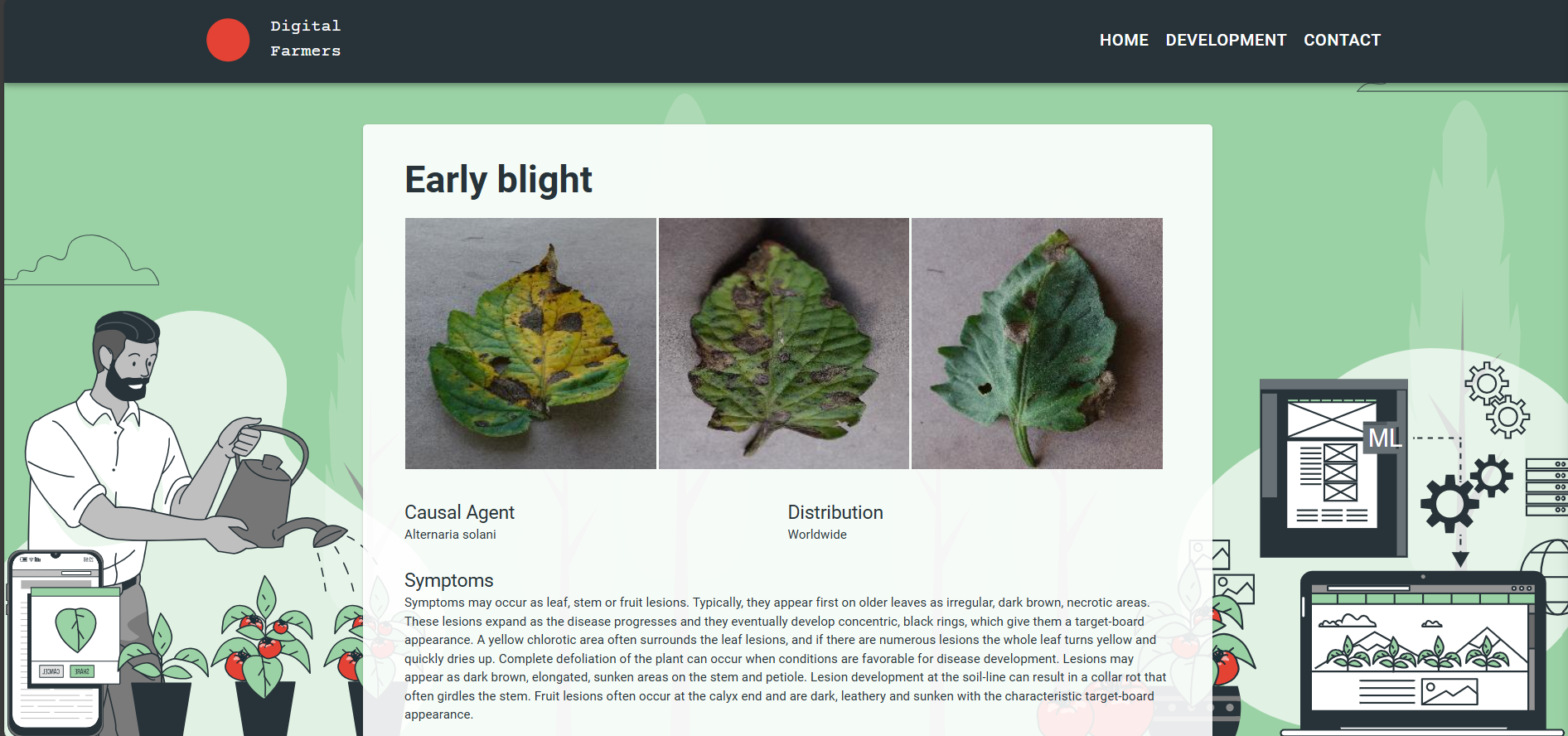
****

Figure 12: Desktop Learn More about Results

**Mobile View**

The application was optimized for mobile devices, ensuring a responsive and user-friendly experience on smaller screens.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Table 1: Mobile view screenshots

* 1. Backend Development
     1. Tools and Frameworks Used

FastAPI was used to develop the backend for this project. FastAPI is a fast, modern web framework for creating responsive APIs with Python 3.7+. I chose FastAPI because it is a very easy framework to work with, has automatic interactive API documentation all with high performance. The TensorFlow model was connected to the FastAPI backend.

* + 1. Architecture and Design

I followed the RESTful API design architecture for this backend. It is designed to handle HTTP requests from the React fronted, process the uploaded image before it sends it to the model for a prediction and returns the results back to the frontend. The API endpoints are stateless and they follow the REST principles for simplicity and scalability. (Dowdell, 2022)

**Key components include:**

* **FastAPI**: handles all the requests to the API and the responses from it.
* **TensorFlow**: A pretrained model that I designed is loaded and it perfoms predictions on uploaded images.
* **CORS Middleware**: This defines which origins are allowed access to send requests to the API. (Hobbs, 2019)
  + 1. Key Functionalities

The main functionalities provided by the backend include:

1. **Model Loading and Prediction:**

* The backend loads a trained TensorFlow model from a chosen directory.
* The /predict endpoint takes image file uploads from user input, then does the needed processing on the images, and finally classify them using the prediction model.

1. **Image Processing:**

* The backend handles all the image processing for the uploaded images preparing them to be the model input for a prediction.

1. **CORS Configuration:**

* This configures what frontend is allowed to communi9cate with the API.

1. **Health Check Endpoint:**

* The attached /ping endpoint can be used to perform a simple health check to determine whether the backend service is operational.
  + 1. Endpoint Descriptions

1. **GET /ping:** This is used to check if the backend service is up and running or not same returns a simplest of the response for a string.
2. **POST /predict:** This endpoint takes image uploads and processes them, runs the prediction model and responds the predicted class and confidence score in a JSON format. This image is read, converted into NumPy array and then provided tree to the model for prediction.
   1. Integration
      1. Integration Approach

The actual integration between the frontend and backend components was implemented using RESTful API calls. HTTP requests from the frontend React application to the backend FastAPI application endpoints. This way each does what it is good in and should do frontend handles user interactions and just the main interface job backend does the heavy load processing and predictions.

* + 1. Communication

1. **API Calls:**

* In the front-end part we use fetch API to perform the HTTP requests to the backend endpoints.
* The /predict endpoint has been used for integration and the image is uploaded and we get the prediction results.

1. **Data Formats:**

* **JSON**: The backend returns responses in JSON format, which is easy to parse and handle in the frontend.
* **Multipart Form Data**: The frontend uploads images using multipart form data, which the backend processes to extract the image files.
  + 1. Process Flow

1. **Image Upload:**

* Users upload images through the frontend interface using the UploadSection component.
* The uploaded image is sent to the backend /predict endpoint via a POST request.

1. **Prediction Request:**

* The backend reads the uploaded image, processes it, and uses the TensorFlow model to make a prediction.
* The prediction result, including the predicted class and confidence score, is returned to the frontend in JSON format.

1. **Displaying Results:**

* The frontend receives the prediction results and displays them to the user.
* Users can view the predicted disease, confidence level, and additional information about the disease if they choose to learn more.
  + 1. Challenges

1. **Cross-Origin Resource Sharing (CORS):**

* Ensuring that the frontend could communicate with the backend required configuring CORS settings.
* The backend was configured to allow requests from the frontend's origin (e.g., http://localhost:3000), ensuring smooth communication.

1. **File Handling and Processing:**

* Handling image uploads and ensuring they were correctly processed for prediction posed some initial challenges.
* The backend was designed to read image files as NumPy arrays and preprocess them appropriately for model input.

1. **Model Loading and Performance:**

* Loading the TensorFlow model and ensuring it performed predictions efficiently was crucial.
* Proper error handling was implemented to manage potential issues with model loading or prediction failures.

1. Code Explanation
   1. main.py (FastAPI Backend)

**Overview of the file:**

The main.py file is the entry point for the FastAPI backend. It sets up the FastAPI application, configures CORS middleware, loads the TensorFlow model, and defines the API endpoints for handling requests.

**Importing Modules and Libraries**

import tensorflow as tf

from fastapi import FastAPI, File, UploadFile

from fastapi.middleware.cors import CORSMiddleware

import uvicorn

import numpy as np

from io import BytesIO

from PIL import Image

import os

* **TensorFlow**: Used for loading the pre-trained model and making predictions.
* **FastAPI**: Framework for building the API.
* **CORS Middleware**: Allows requests from the frontend (React application).
* **Uvicorn:** ASGI server to run the FastAPI application.
* **NumPy:** Handles numerical operations on arrays.
* **PIL**: Used for image processing.

**Creating the Web Application and Enabling CORS**

app = FastAPI()

origins = [

    "http://localhost",

    "http://localhost:3000",

]

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=origins,

    allow\_credentials=True,

    allow\_methods=["\*"],

    allow\_headers=["\*"],

)

* FastAPI app: Initializes the FastAPI application.
* CORS Middleware: Configures the allowed origins for cross-origin requests, enabling the frontend to communicate with the backend.

**Loading the Model and Defining Class Names**

MODEL\_DIR = os.getenv("MODEL\_DIR", "models/1")

try:

    MODEL = tf.keras.models.load\_model(MODEL\_DIR)

except Exception as e:

    raise RuntimeError(f"Failed to load model. Error: {e}")

CLASS\_NAMES = [

    'Tomato\_\_\_Bacterial\_spot',

    'Tomato\_\_\_Early\_blight',

    'Tomato\_\_\_Late\_blight',

    'Tomato\_\_\_Leaf\_Mold',

    'Tomato\_\_\_Septoria\_leaf\_spot',

    'Tomato\_\_\_Spider\_mites Two-spotted\_spider\_mite',

    'Tomato\_\_\_Target\_Spot',

    'Tomato\_\_\_Tomato\_Yellow\_Leaf\_Curl\_Virus',

    'Tomato\_\_\_Tomato\_mosaic\_virus',

    'Tomato\_\_\_healthy'

]

* **Model Loading**: Loads the TensorFlow model from the specified directory.
* **Class Names**: Defines the list of class names that the model can predict.

**Defining a Test Route for the API**

@app.get("/ping")

async def ping():

    return "Nawanadem!"

* **Ping Endpoint**: A simple endpoint to test if the server is running. Returns a greeting message.

**Converting File Data to Image Array**

def read\_file\_as\_image(data: bytes) -> np.ndarray:

    image = np.array(Image.open(BytesIO(data)))

    return image

* **read\_file\_as\_image**: Converts the uploaded image file to a NumPy array suitable for model input.

**Making Predictions of Uploaded Images**

@app.post("/predict")

async def predict(file: UploadFile = File(...)):

    image = read\_file\_as\_image(await file.read())

    img\_batch = np.expand\_dims(image, 0)

    predictions = MODEL.predict(img\_batch)

    predicted\_class = CLASS\_NAMES[np.argmax(predictions[0])]

    confidence = np.max(predictions[0])

    return {

        'class': predicted\_class,

        'confidence': float(confidence)

    }

* **/predict Endpoint**: Handles image upload, processes the image, makes a prediction using the loaded model, and returns the predicted class and confidence score.
* **read\_file\_as\_image**: Reads and converts the uploaded image.
* **np.expand\_dims**: Adds a batch dimension to the image array.
* **MODEL.predict**: Makes a prediction on the image.
* **CLASS\_NAMES**: Maps the predicted index to the corresponding class name.
* **Confidence**: Calculates the confidence score of the prediction.

**Running the Web Application on Local Server**

if \_\_name\_\_ == "\_\_main\_\_":

    uvicorn.run(app, host='localhost', port=8000)

* Uvicorn Server: Runs the FastAPI application on localhost:8000.
  1. React Components
     1. App.jsx

**Overview:**

The App.jsx file is the main entry point of the React application. It sets up the global structure of the application by including the main components like NavBar and BodySection.

**Code:**

import { CssBaseline } from "@mui/material";

import NavBar from "./components/NavBar";

import BodySection from "./components/BodySection";

const App = () => {

    return (

        <>

            <CssBaseline />

            <NavBar />

            <BodySection />

        </>

    );

}

export default App;

**Explanation:**

* + - **CssBaseline**: A Material-UI component that provides a consistent baseline to build upon.
    - **NavBar**: The navigation bar component at the top of the application.
    - **BodySection**: The main body section that contains the primary content of the application.
    1. BodySection.jsx

**Overview**:

The BodySection.jsx file defines the main content area of the application. It includes the UploadSection where users can upload images for disease detection.

**Code**:

import \* as React from "react";

import { Box } from "@mui/material";

import UploadSection from "./UploadSection";

const BodySection = () => {

    return (

        <Box>

            <div

                style={{

                    backgroundImage: `url("./img/bg.svg")`,

                    backgroundPosition: "center",

                    backgroundRepeat: "no-repeat",

                    backgroundAttachment: "fixed",

                    backgroundSize: "cover",

                    minHeight: "100vh",

                }}>

                    <UploadSection/>

                </div>

        </Box>

    );

};

export default BodySection;

**Explanation:**

* + - **Box**: A Material-UI component used for layout purposes.
    - **Background Image:** Sets a background image for the main content area to enhance visual appeal.
    - **UploadSection:** Embeds the UploadSection component where users can interact with the tool.
    1. UploadSection.jsx

**Overview**:

The UploadSection.jsx file handles the image upload functionality and displays the prediction results. It includes states for managing the uploaded image, prediction results, and the current state of the interface.

**Explanation:**

* + - **State Management:** Uses React's useState hook to manage the current state of the component, the selected image, the prediction result, and other relevant states.
    - **Image Upload Handling:**
      * Provides a file input for users to select an image.
      * Handles the file upload and displays the selected image.
    - **Prediction Submission:**
      * Submits the uploaded image to the backend /predict endpoint.
      * Receives and processes the prediction result.
    - **Displaying Results:**
      * Displays the prediction result, including the predicted class and confidence score.
      * Provides a "Learn More" button for users to get detailed information about the predicted disease.
    - **Reset Functionality**: Allows users to reset the component state and upload a new image.
    1. NavBar.jsx

**Overview:**

The NavBar.jsx file defines the navigation bar at the top of the application, which includes links to different sections like Home, Development, and Contact.

**Explanation:**

* **Responsive Design:** Adapts to different screen sizes using Material-UI's useTheme and useMediaQuery hooks.
* **Menu Handling:**
  + Toggles the menu for mobile views.
  + Provides navigation links to various sections.
* **Styling**: Customizes the appearance of the navigation bar using Material-UI components and styles.

1. Testing and Validation
   1. Testing Strategies

In the development of the tomato disease detection system, manual testing was the primary strategy used to ensure the robustness and accuracy of the application. The process involved testing the entire application workflow and verifying its performance with real-world data.

* 1. Manual Testing:
* **Purpose**: To perform end-to-end testing from the perspective of a real user.
* **Scope**: Included uploading images, receiving predictions, and verifying that the system's response was accurate and timely.
* **Tools**: Conducted manually by developers and testers using various devices to ensure responsiveness and functionality across platforms.
  1. API Testing with Postman:
* **Purpose**: To test the backend API endpoints independently.
* **Scope**: Ensured that the API endpoints correctly handled requests and returned appropriate responses.
* **Tools**: Used Postman to simulate API requests and validate the responses from the server.
  1. Validation

Validation of the system involved using real-world data to ensure that the model performed well under practical conditions. This process included:

* Data Collection: Gathered real-world images of tomato leaves affected by different diseases, which were not part of the training dataset.
* Model Evaluation: The model was evaluated using these new images to measure its performance in predicting the correct disease class. The evaluation metrics included accuracy, loss and confusion matrix as detailed in the Model Evaluation section.

The validation process confirmed that the model could generalize well to unseen data and provided accurate predictions, thereby validating its effectiveness in a real-world scenario.

* 1. Iterative Improvements:
* Continuous Improvement: The development team adopted an agile approach to continuously improve the application. Regular updates were deployed to address bugs, add new features, and optimize existing functionalities based on ongoing user feedback.
* Monitoring and Evaluation: Post-deployment, the system was monitored to gather data on its performance and user interactions. This data was analyzed to identify areas for further enhancement, ensuring the application remained effective and user-friendly.

Through these comprehensive testing and validation strategies, the tomato disease detection system was refined to provide reliable and accurate predictions, offering a valuable tool for users in the agricultural sector.

1. Challenges and Solutions
   1. Challenges

**1. Data Quality and Quantity:**

* **Description**: Acquiring a high-quality and diverse dataset was one of the major challenges. The initial dataset contained images of varying quality, and some classes had significantly fewer images than others, leading to imbalanced data.

**2. Model Performance:**

* **Description**: Achieving high accuracy and generalization capability with the model was difficult due to the complexity of distinguishing between similar-looking diseases and the presence of noise in the data.

**3. Integration of Frontend and Backend:**

* **Description**: Ensuring seamless communication between the frontend React application and the backend FastAPI service posed challenges, especially in terms of handling file uploads and processing predictions efficiently.

**4. User Experience (UX):**

* **Description**: Designing an intuitive and responsive user interface that worked well across different devices and screen sizes required careful planning and multiple iterations.

**5. Real-time Prediction Latency:**

**Description**: Providing real-time predictions with low latency was crucial for user satisfaction. However, processing images and generating predictions quickly while ensuring accuracy was challenging.

**6. Deployment and Scalability:**

* **Description**: Deploying the model in a production environment and ensuring it could handle multiple requests concurrently without significant downtime or performance degradation was a critical challenge.
  1. Solutions

**1. Data Quality and Quantity:**

**Solution:**

* **Data Augmentation:** Applied various data augmentation techniques such as rotation, flipping, and zooming to artificially increase the size and diversity of the dataset.
* **Curated Dataset**: Filtered out low-quality images and focused on curating a balanced dataset with sufficient samples for each class.

**2. Model Performance:**

**Solution:**

* **Hyperparameter Tuning**: Experimented with different hyperparameters such as learning rate, batch size, and the number of epochs to find the optimal configuration for the model.
* **Regularization Techniques:** Implemented dropout and batch normalization to prevent overfitting and improve the model's generalization capability.

**3. Integration of Frontend and Backend:**

**Solution**:

* **API Design:** Designed a clear and efficient API for handling image uploads and returning predictions.
* **Error Handling:** Implemented robust error handling on both frontend and backend to manage potential issues during file upload and prediction processing.

**4. User Experience (UX):**

**Solution**:

* **Material-UI**: Leveraged Material-UI components to build a consistent and responsive design.

**5. Real-time Prediction Latency:**

**Solution:**

* **Model Optimization:** Optimized the model by reducing its size and complexity without compromising accuracy.
* **Asynchronous Processing:** Used asynchronous functions in the backend to handle predictions and return responses quickly.

**6. Deployment and Scalability:**

**Solution**:

* **Containerization**: Used Docker to containerize the application, making it easier to deploy and scale.
* **Load Balancing**: Implemented load balancing to distribute incoming requests evenly across multiple instances of the application, ensuring high availability and reliability.

By addressing these challenges with the mentioned solutions, the project successfully developed a robust, efficient, and user-friendly tomato disease detection system.

1. Conclusion
   1. Summary

In this project, a systematic and structured approach was adopted to develop an efficient and user-friendly tomato disease detection system. The methodologies used encompass a wide range of modern tools and frameworks, ensuring a comprehensive solution. Key methodologies included:

* **Software Development Practices**: Followed the Waterfall model to guide the project through its various stages—requirements gathering, design, implementation, testing, deployment, and maintenance.
* **Data Collection and Preparation**: Collected and preprocessed a dataset of tomato leaf images, using techniques like data augmentation and normalization to enhance the dataset's quality and diversity.
* **Model Building and Training**: Developed a Convolutional Neural Network (CNN) using TensorFlow and Keras, optimizing the model through hyperparameter tuning and regularization techniques.
* **Model Evaluation**: Evaluated the model using metrics such as accuracy, loss and confusion matrix, ensuring its effectiveness and reliability.
* **Web Application Development**: Built a responsive and user-friendly web application using ReactJS for the frontend and FastAPI for the backend, integrating the machine learning model to provide real-time predictions.
* **Testing and Validation**: Conducted thorough manual testing and validation using real-world data, gathering user feedback to iteratively improve the application.
* **Challenges and Solutions**: Addressed various challenges such as data quality, model performance, and integration issues through targeted solutions, ensuring the project's success.
  1. Reflection

The methodologies selected in this project worked well to achieve my outcomes. A linear and well-defined method of the Waterfall model allowed smooth progress from one stage of development to another. New frameworks such as TensorFlow, ReactJS, and FastAPI were employed to ensure that both the machine learning model and the web application were robust and efficient.

**Lessons Learned:**

1. **Importance of Data Quality:**

* Ensuring high-quality and balanced datasets is crucial for developing accurate and reliable machine learning models. Data augmentation techniques significantly improved the model's performance by increasing the dataset's diversity.

1. **Effective Model Optimization:**

* Hyperparameter tuning and regularization techniques were essential in enhancing the model's accuracy and preventing overfitting. Continuous experimentation and validation were key to finding the optimal configuration.

1. **Seamless Integration:**

* Integrating the frontend and backend components required careful planning and robust error handling to ensure smooth communication and functionality. Clear API design and asynchronous processing played a vital role in achieving this.

1. **User-Centric Design:**

* Prioritizing user experience through responsive design and iterative improvements based on user feedback led to a more intuitive and user-friendly application. Regular user testing provided valuable insights for refinement.

1. **Scalability and Deployment:**

* Using containerization and load balancing techniques facilitated the scalable and reliable deployment of the application. These practices ensured the system could handle multiple requests concurrently without performance degradation.

Overall, the project demonstrated the usefulness of combining modern machine learning techniques with robust web development practices to create a practical and valuable tool for tomato disease detection. The lessons learned and methodologies adopted can serve as a blueprint for future projects in similar domains.

* 1. Benefits

1. **Benefits to Users:**
2. **Early Detection**: The system allows farmers and gardeners to detect diseases in tomato plants early, allowing timely intervention and reducing crop loss and preventing economic loss.
3. **Ease of Use**: The user-friendly interface allows a wide range of users even to users with minimal technical expertise, providing an easy way to upload images and get quick predictions on the tomato leaves and take the necessary actions.
4. **Cost-Effective**: The system will reduce expert consultant visits for disease management in tomato plants, so it is a cost-effective solution.
5. **Benefits to Me:**
6. **Deep Learning Skills**: This project helped me update my deep learning knowledge and gave hands-on practice in building CNN models and tuning architectures for image classification tasks in this case I gained experience using TensorFlow.
7. **Full-Stack Development Skills**: Developing both the frontend and backend components improved my skills in full-stack web development, including frameworks like ReactJS and FastAPI.
8. **Project Management:** Following the Waterfall model and handling various stages of the project from requirements gathering to deployment provided valuable experience in project management and structured development methodologies for me which will be very valuable me as someone about to graduate at the time of this writing.
   1. Future Tasks
9. **Expand Dataset:** I can would like to increase the amount of variant and quality images to increase the robustness and accuracy of the model.
10. **Enhance Model:** I can explore different machine learning model architectures other than the CNN so see if I can get improved accuracy.
11. **Mobile App Development**: A mobile app will make the system easily accessible even when the farmer is on the field.
12. **Multi-Language Support**: Having the application is different languages will allow the app to be used by a wide range of farmers from different backgrounds.
13. **Automated Monitoring:** This would allow me to implement automated monitoring of the systems performance and optimise it in real time..
14. **User Documentation and Tutorials**: Creating full user guides and video tutorial to show the users how they can make full use of the system.

.

* 1. Ethics

There are numerous ethical issues that arise from the creation of a tomato plant disease researching bot. Keeping the development of the project into the ethical lane, is important for its acceptability and credibility.

1. **Data Privacy and Security:**

* That the data should be chiffon and secure enough was the first issue and the biggest constraint. These people will have to feel that if they are going to be uploading images of their crops they can trust that the data will not be used or shared without their permission. Secure storage: To safeguard user data, data encryption and secure storage is something that should be implemented with best practices.

1. **Accuracy and Reliability:**

* Accuracy of disease detection is crucial to avoid misdiagnosis resulting into improper treatment and possible crop loss. The model will know that continuous confirmation and retraining is required so that the predictions will be retrained on this data and the model will make more accurate predictions when new data is clustered with a similar combination of independent features. Users are ethically obligated to tell users about how to expect their model to behave (the accuracy) and how not to expect their model to behave (the limitations).

1. **Bias and Fairness:**

* Ensuring that the model is unbiased, i.e — we must ensure that the model does not introduce any bias in the decision making and in a diverse set of crops and growing conditions; the model should provide fair treatment to all users; “Something you can do to work around this is to train on a diverse dataset which reflects different types of tomatoes and the grow environments and always monitor the model to see if there are any signs of bias.

1. **Environmental Impact:**

* The project will impart in sustainable agriculture helps in early detection and management of diseases of tomato. This reduces the use of harmful pesticides and increases the odor level of more eco-friendly farming. Ethical value: to promote sustainable farming, and environmental impact of agriculture must be that small as possible.

**Why Did I Choose This Project?**

I chose this project as I have a strong interest in using modern technology solutions to solve real life, practical issues in the agriculture sector. Agriculture is a basic area on which human life depends on, and the health of the crop directly impacts food security as well as the economic health of everyone. The basic idea that attracted me to this project is that we have the possibility to create a real, positive effect on the lives of the farmers by providing them with a tool to diagnose the disease of the plants in a quick and accurate manner. This does not only help in minimizing the loss of crops but also helps in advancing the cause of sustainable agricultural practices by minimizing the use of harmful chemical treatment in excessive quantities. The application of machine learning and web development for the production of an affordable and effective tool fulfils my own relevancy to technology and its role in enhancing the productivity of agriculture.

**Project Links**

Backend: <https://digital-farmers-harvesting-insights-to.onrender.com>

Frontend: <https://digital-farmers-front-end.onrender.com>

# References

Agarwal, V. (2020, June 13). *Different Types of Normalization in Tensorflow*. Retrieved from Medium.com: https://towardsdatascience.com/different-types-of-normalization-in-tensorflow-dac60396efb0

Alvi, F. (2024, May 22). *What is Natural Language Processing NLP? – Starter’s Guide*. Retrieved from opencv.org: https://opencv.org/blog/natural-language-processing-nlp/

AWS. (2024, May 10). *What Is Data Augmentation?* Retrieved from AWS: https://aws.amazon.com/what-is/data-augmentation/#:~:text=Data%20augmentation%20is%20the%20process,machine%20learning%20(ML)%20models.

Bhikadiya, A. M. (2022, March 8). *Plant-Disease-Detection*. Retrieved from Github: https://github.com/manthan89-py/Plant-Disease-Detection

Brownlee, J. (2020, September 12). *Understand the Impact of Learning Rate on Neural Network Performance*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Brownlee, J. (2023, October 4). *A Gentle Introduction to k-fold Cross-Validation*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/k-fold-cross-validation/#:~:text=Cross%2Dvalidation%20is%20a%20resampling,into%20training%20and%20test%20set.

Chollet, F. (2023, July 24). *Training & evaluation with the built-in methods*. Retrieved from Tensorflow.com: https://www.tensorflow.org/guide/keras/training\_with\_built\_in\_methods

Corrales, E. (2022, June 10). *What is the Waterfall Software Development Methodology?* Retrieved from developer.com: https://www.developer.com/design/sdlc-waterfall-model/

D, E. (2019, December 8). *Accuracy, Recall & Precision*. Retrieved from medium.com: https://medium.com/@erika.dauria/accuracy-recall-precision-80a5b6cbd28d

Denham. (2013, December 31). *Structured Approach To Software Development*. Retrieved from Tutorhunt: https://www.tutorhunt.com/resource/9204/

Devansh. (2022, January 17). *How does Batch Size impact your model learning*. Retrieved from Medium.com: https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa

Dowdell, D. (2022, March 26). *Designing a REST API*. Retrieved from medium.com.

Dremio. (2024, May 28). *ReLU Activation Function*. Retrieved from Dremio: https://www.dremio.com/wiki/relu-activation-function/#:~:text=The%20ReLU%20activation%20function%20works,%3D%20max(0%2C%20x)

EMMANUEL, T. O. (2018). *PlantVillage Dataset*. Retrieved from kaggle.com: https://www.kaggle.com/datasets/emmarex/plantdisease/discussion

Hobbs, S. (2019, April 16). *CORS Tutorial: A Guide to Cross-Origin Resource Sharing*. Retrieved from Auth0: https://auth0.com/blog/cors-tutorial-a-guide-to-cross-origin-resource-sharing/

Igareta, A. (2021, June 2). *How To Split a Tensorflow Dataset into Train, Validation, and Test sets*. Retrieved from Towardsdatascience: https://towardsdatascience.com/how-to-split-a-tensorflow-dataset-into-train-validation-and-test-sets-526c8dd29438

Jasson, R. (2021, October 31). *Farmassist*. Retrieved from Github: https://github.com/farmassistX/farmassist

Jordan, J. (2017, November 2). *Hyperparameter tuning for machine learning models.* Retrieved from jeremyjordan.com: https://www.jeremyjordan.me/hyperparameter-tuning/

Kitchell. (2022, Jun 11). *Loss Functions and Optimizers*. Retrieved from Kitchell: https://kitchell.github.io/DeepLearningTutorial/7lossfunctionsoptimizers.html

Kulkarni, P. (2021, November 22). *Plant Disease Detection Using Image Processing and Machine Learning*. Retrieved from Cornell University: https://arxiv.org/abs/2106.10698

Lang, N. (2021, Dec 5). *Breaking down Convolutional Neural Networks: Understanding the Magic behind Image Recognition*. Retrieved from towardsdatascience: https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4

Mandal, M. (2024, May 24). *Introduction to Convolutional Neural Networks (CNN)*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

Narkhede, S. (2018, May 9). *Understanding Confusion Matrix*. Retrieved from Towards Data Science: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Nishtha. (2024, May 15). *Client Server Architecture - Working, Types, and Examples*. Retrieved from IntelliPaat: https://intellipaat.com/blog/what-is-client-server-architecture/

Oppermann, A. (2024, May 26). *Activation Functions in Deep Learning: Sigmoid, tanh, ReLU*. Retrieved from artemoppermann: https://artemoppermann.com/activation-functions-in-deep-learning-sigmoid-tanh-relu/#:~:text=Simply%20put%2C%20the%20softmax%20activation,interval%20%5B0%2C%201%5D.

Prakash, Y. (2023, Feb 6). *A Full-Stack Project With React and FastAPI — Part 1*. Retrieved from Medium: https://medium.com/this-code/a-full-stack-project-with-react-and-fastapi-part-1-82367f43911a

Rsvmukhesh. (2023, May 19). *Determining the Number of Epochs*. Retrieved from Medium.com: https://medium.com/@rsvmukhesh/determining-the-number-of-epochs-d8b3526d8d06

Turing. (2024, Jan 29). *Python FastAPI vs Flask: A Detailed Comparison*. Retrieved from Turing: https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison#which-wins?