International Journal for Multidisciplinary Research (IJFMR)

E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Plant Diseases Classification

Sahana R[1], Radhika Relekar[2], Srustisingh DT[3], S. Firoze Ahamed[4], Prof. Prasad PS[5]

[1,2,3,4]Student, Presidency university
[5]Professor, Presidency university

Abstract

This project aims to create an effective deep learning model for classifying plant diseases using image data from the PlantVillage dataset. The primary objective is to accurately detect different plant diseases and identify healthy plants, aiding in agricultural monitoring and decision-making. The dataset comprises labeled images of plants with diseases affecting crops such as tomatoes, peppers, and potatoes.

A Convolutional Neural Network (CNN) serves as the foundation for this image classification task, implemented using the TensorFlow and Keras libraries. The workflow involves preprocessing the data, dividing it into training and testing subsets, and training the model using categorical cross-entropy as the loss function. The model achieves a high accuracy of 96% when tested on a dataset of 2,065 images. Additional evaluation metrics, including precision, recall, and F1-score, highlight the model's reliability, particularly in detecting diseases

This classification system provides a practical solution for early disease detection, contributing to reduced crop losses and improved agricultural outcomes. Future work could involve leveraging transfer learning, deploying the model in real-world scenarios, and extending its application to other crops and diseases.

I Introduction

Agriculture plays a vital role in human survival, with food security serving as a cornerstone of societal stability. However, the rising incidence of plant diseases poses a serious challenge to agricultural productivity, endangering global food supplies. Effective and timely disease detection is critical to safeguarding crop health, mitigating risks, and supporting sustainable farming practices. Early identification of plant diseases not only minimizes the impact on crop yields but also optimizes resource usage and reduces dependence on chemical treatments, which can harm the environment and lead to financial losses.

Historically, plant disease detection relied on human observation, where farmers or agricultural experts inspected crops for signs of disease. While these methods are valuable, they are often subjective, timeintensive, and less effective for large-scale operations. Moreover, traditional approaches lack the precision and scalability required to address the complexities of modern agriculture.

Recent advancements in technology, particularly in image processing and artificial intelligence (AI), have transformed plant disease detection. Early computational techniques, such as K-means clustering and support vector machines (SVM), offered promising results by analyzing visual characteristics of diseased plants. However, these methods depended heavily on manual feature extraction and segmentation, limiting their adaptability to diverse agricultural conditions.

Deep learning (DL) technologies, particularly convolutional neural networks (CNNs), have emerged as powerful tools for plant disease detection. Unlike earlier approaches, CNNs automatically extract significant features from raw images, enabling precise disease classification. These models are wellsuited for large datasets and varying imaging conditions, providing robust solutions for disease identification. Furthermore, the adoption of transfer learning has enhanced CNN performance, allowing pre-trained models to be fine-tuned for specific datasets, improving efficiency even for smaller-scale applications.

This paper explores the evolution of plant disease detection technologies, tracing the shift from traditional image processing methods to cutting-edge deep learning techniques. It highlights the strengths and limitations of these approaches, examines recent research in the field, and identifies opportunities for future advancements. By leveraging AI-driven solutions, this study aims to contribute to increased agricultural productivity and global food security.

Addressing the challenges of plant disease management underscores the need for automated and accurate detection systems to minimize crop losses. This capstone project focuses on developing a deep learning-based framework for classifying plant diseases using the PlantVillage dataset. The goal is to design a robust model capable of identifying a variety of plant diseases and recognizing healthy crop conditions, empowering farmers and stakeholders to make informed decisions.

The dataset comprises thousands of labeled, high-resolution images of plants, covering multiple species such as tomatoes, peppers, and potatoes, as well as various disease states, including Pepper Bacterial Spot, Tomato Mosaic Virus, Tomato Early Blight, Tomato Yellow Leaf Curl Virus, and Potato Late Blight. The data preprocessing pipeline includes resizing images, normalizing pixel values, and splitting the dataset into training, validation, and test subsets to ensure balanced and comprehensive model evaluation.

A CNN architecture, implemented using TensorFlow and Keras, is developed for this project. Training strategies, such as data augmentation, are employed to address class imbalances and improve the model's generalizability. The model is optimized using categorical cross-entropy as the loss function and the Adam optimizer. Hyperparameters, including learning rate, batch size, and epoch count, are fine-tuned to enhance model performance.

The trained model achieves an impressive test accuracy of 96% on a dataset of 2,065 images, effectively classifying plant diseases. Key performance metrics, including precision, recall, and F1-scores, are calculated for individual classes, confirming the model's reliability. Additional tools, such as confusion matrices and accuracy/loss graphs, are used to visualize and evaluate the model's performance.

This system provides a practical and scalable solution for plant disease detection, with potential applications in real-world agriculture through integration with mobile apps and IoT platforms. The project lays the groundwork for future improvements, such as leveraging transfer learning for larger datasets, implementing real-time monitoring systems, and extending the model to include additional crops and disease types. By applying machine learning, this work promotes sustainable farming practices, better crop management, and enhanced food security.

II Methodology

The methodology for this plant disease classification project was structured into several key stages to develop a reliable and efficient model for detecting plant diseases from images. Each step was carefully executed to ensure optimal performance.

1. Dataset Collection and Exploration

The project utilized the publicly available PlantVillage dataset, which provides labeled images of healthy and diseased plant leaves across various species, including tomatoes, peppers, and potatoes. The dataset features images of several diseases such as Pepper Bacterial Spot, Tomato Mosaic Virus, Tomato Early Blight, Tomato Yellow Leaf Curl Virus, and Potato Late Blight. Exploratory Data Analysis (EDA) was performed to investigate the dataset's composition, assess class distributions, and examine image dimensions.

2. Data Preprocessing

The data underwent several preprocessing steps to prepare it for model training:

•Image Resizing: Images were resized to a consistent dimension (e.g., 128x128 or 224x224 pixels) to standardize inputs for the model.

•Normalization: Pixel values were scaled to the range [0, 1], improving training efficiency by enabling faster convergence.
•Data Splitting: The dataset was divided into training, validation, and testing subsets with a ratio of 70:15:15, ensuring a balanced evaluation of the model's generalization capabilities.
•Data Augmentation: To address class imbalance and enhance model robustness, augmentation techniques like rotation, flipping, zooming, and shifting were applied to the training images.

3. Model Architecture Design

The classification model's architecture was designed to extract and process image features efficiently:
•Feature Extraction: Layers designed to capture spatial patterns and features from the images were added, with activation functions ensuring non-linear learning.
•Dimension Reduction: Pooling operations reduced the spatial size of features, lowering computational costs while retaining essential information.
•Classification Layers: Fully connected layers were used to map the extracted features to output categories, with a final activation layer designed for multi-class classification.
•Regularization: Dropout layers were integrated to reduce overfitting by randomly disabling neurons during training, improving the model's ability to generalize.

4. Model Training

The training process incorporated several strategies and configurations to achieve optimal results:
•Loss Function: Categorical cross-entropy was employed to optimize multi-class classification performance.
•Optimizer: The Adam optimizer was used due to its adaptive learning rate and computational efficiency.
•Training Parameters: A batch size of 32 and a maximum of 50 epochs were set to allow proper convergence of the model.
•Early Stopping: To prevent overfitting, training was stopped automatically when the validation loss stabilized.
•Hardware Acceleration: The training process leveraged GPU support to significantly reduce computation time.

5. Model Evaluation

Once the model was trained, its performance was assessed using a range of metrics to ensure reliability:
•Accuracy: The overall accuracy of the model was calculated on the test dataset.
•Precision, Recall, and F1-Score: These metrics were evaluated for each class to measure the model's ability to handle imbalanced categories.
•Confusion Matrix: A confusion matrix was used to visualize predictions, identify misclassifications, and analyze class-level performance.
•Performance Visualization: Plots of training and validation accuracy and loss were generated
to monitor the model's progress during training and identify potential areas of improvement.

III Deep Learning Architectures

The plant disease classification model is based on a Convolutional Neural Network (CNN), a robust method for image classification tasks. CNNs are effective at extracting spatial and hierarchical features from plant leaf images, making them ideal for accurate disease detection. This architecture is designed to achieve a balance between computational efficiency and high accuracy, ensuring effective classification across multiple disease categories.
The model begins with an input layer that accepts pre-processed images resized to 128x128x3 (height, width, and color channels), ensuring uniformity during training. The core structure consists of three convolutional blocks. Each block applies a set of learnable filters (kernels) that extract spatial features such as edges and textures. ReLU (Rectified Linear Unit) activation is used to introduce non-linearity, enabling

the network to capture complex patterns in the data. Following each convolutional layer, MaxPooling operations with a 2x2 pooling size downsample the feature maps, reducing spatial dimensions while preserving key features. This results in progressively smaller feature maps as the data flows through the layers.
To prevent overfitting, Dropout layers with a rate of 0.4 are added after the pooling layers. These layers randomly deactivate neurons during training, encouraging the network to generalize better and avoid memorizing the training data. The output from the convolutional and pooling layers is then flattened into a 1D vector using a Flatten layer, preparing the data for the fully connected layers.
The fully connected layers, responsible for learning high-level abstractions from the extracted features, include two dense layers. The first dense layer has 256 neurons, while the second layer contains 128 neurons, both with ReLU activation. A Dropout layer is also applied after the first dense layer to further mitigate overfitting. The output layer contains 15 neurons, each corresponding to one of the 15 disease classes, including a healthy class. A softmax activation function is used in the output layer, generating a probability distribution over the 15 classes, with the sum of probabilities equal to 1.
The model is trained using the Adam optimizer with a learning rate of 0.001, which adapts the learning rate during training to speed up convergence. The categorical cross-entropy loss function is selected for multi-class classification, and model performance is evaluated using metrics such as accuracy, precision, recall, and F1-score. The training process runs for 50 epochs with a batch size of 32, and early stopping is employed to halt training when validation loss no longer improves. Here is a summary of the architecture components:

Layer Type	Filters/Units	Kernel Size	Activation	Output Shape
Input Layer	-	-	-	(128, 128, 3)
Convolutional Layer 1	32	3*3	ReLU	(128, 128, 32)
Max Pooling Layer 1	-	2*2	-	(64, 64, 32)
Dropout Layer 1	-	-	-	(64, 64, 32)
Convolutional Layer 2	64	3*3	ReLU	(64, 64, 32)
Max Pooling Layer 2	-	2*2	-	(32, 32, 64)
Dropout Layer 2	-	-	-	(32, 32, 64)
Convolutional Layer 3	128	3*3	ReLU	(32, 32, 128)
Max Pooling Layer 3	-	2*2	-	(16, 16, 128)
Flatten Layer	-	-	-	(32768)
Fully Connected Layer 1	256	-	ReLU	(256)
Dropout Layer 3	-	-	-	(256)
Fully Connected Layer 2	128	-	ReLU	(128)
Output Layer	15	-	Softmax	(15)

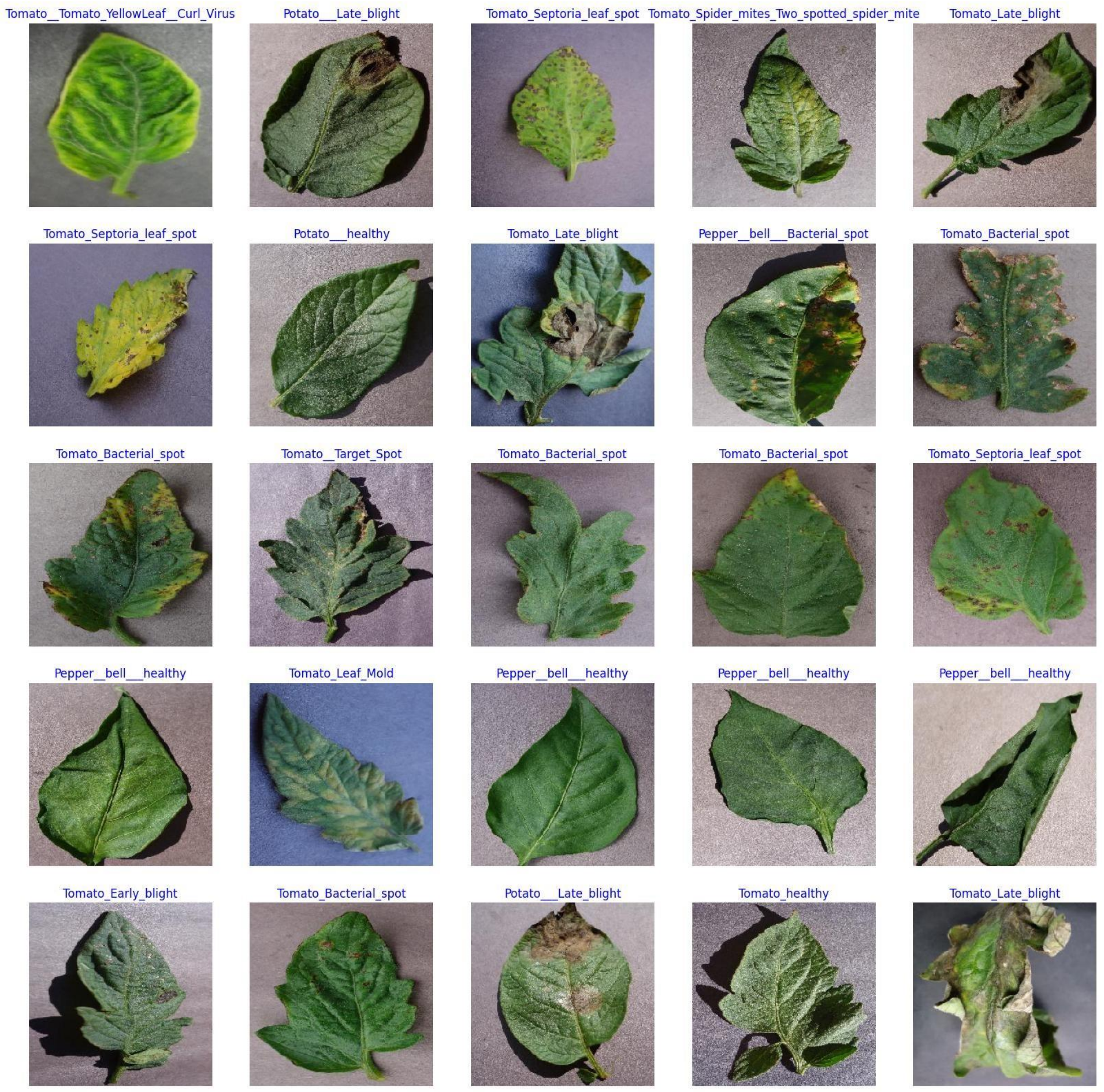


Figure 1 Some of the plant diseases from the PlantVillage dataset

This model is a comprehensive deep learning approach for classifying plant diseases based on leaf images, effectively leveraging strengths in spatial feature extraction and pattern recognition.

IV Data Augmentation

Data augmentation is a critical process in this project to improve the performance and generalization of the Convolutional Neural Network (CNN) for plant disease classification. It helps mitigate class imbalance and boosts the variability of the training data without needing to gather additional images. By applying various transformations to the existing dataset, the model becomes more robust to variations commonly found in real-world agricultural scenarios, such as changes in orientation, scale, and lighting.

Purpose of Data Augmentation

Class Imbalance: Some plant diseases in the PlantVillage dataset have fewer samples than others. Data augmentation helps create a more balanced dataset by artificially increasing the number of samples for underrepresented classes.

Prevention of Overfitting: Augmentation introduces small variations to the training images, preventing the model from memorizing the data and encouraging it to learn generalized features instead. Enhancing Model

Robustness: By simulating real-world conditions such as varying leaf orientation, lighting differences, and noise, the model becomes better equipped to handle unseen data. Techniques Applied for Data Augmentation

Several augmentation methods were utilized on the training images using tools like TensorFlow's ImageDataGenerator or Keras utilities:

Rotation:

Description: Images are randomly rotated within a certain range (e.g., 0–40 degrees).

Purpose: Helps the model recognize disease patterns on leaves in various orientations.

Horizontal and Vertical Flipping:

Description: Images are flipped both horizontally and vertically to generate mirrored versions. Purpose: Ensures the model doesn't overfit to a specific leaf orientation, allowing it to handle different angles.

Zooming:

Description: Random zooming in on the images (e.g., 0–20%).

Purpose: Helps the model capture finer details of disease spots, even when they appear at varying scales.

Shifting:

Description: Random horizontal and vertical shifts (e.g., up to 20% of the image dimensions). Purpose: Prepares the model to recognize plant diseases even when the leaves are not centered in the image.

Brightness and Contrast Adjustment:

Description: Random adjustments to the image's brightness and contrast levels.

Purpose: Simulates different lighting conditions, which are often encountered in outdoor environments.

Rescaling:

Description: Pixel values were normalized by rescaling them to the range [0, 1].

Purpose: Standardizes input images, helping to accelerate convergence during model training. Impact of Data Augmentation

Expanded Dataset: Data augmentation helped increase the size of the training set, allowing the model to generalize better.

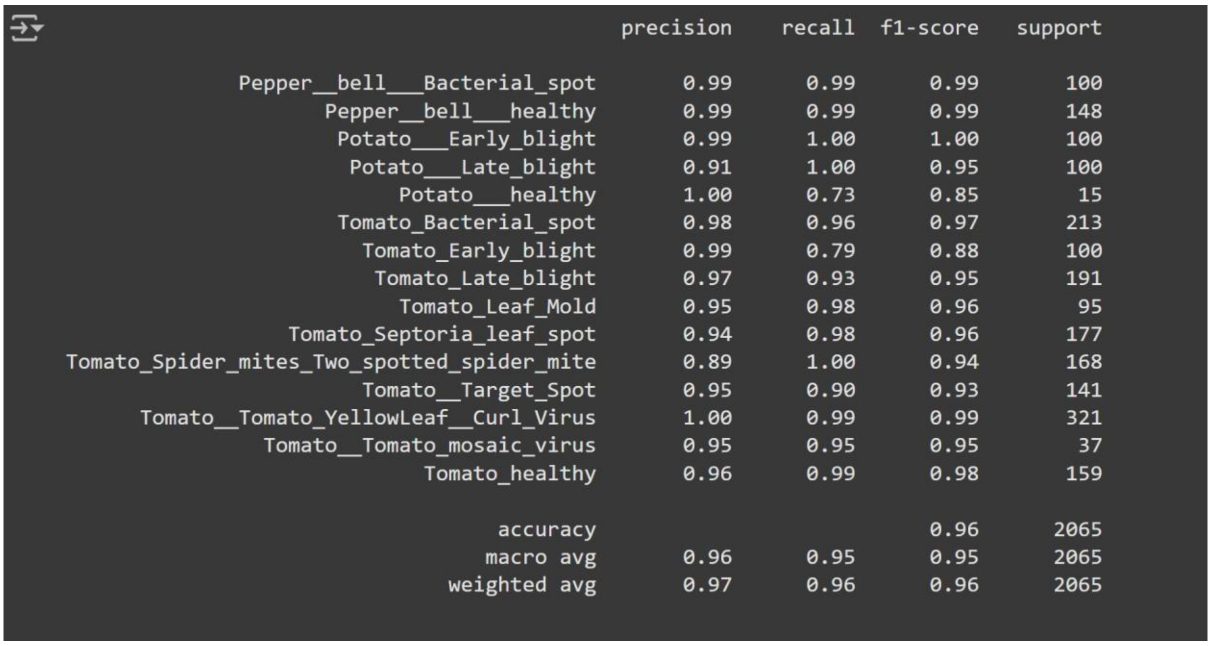
Improved Model Performance: The model achieved a high test accuracy of 96%, benefitting from the varied training examples produced through augmentation.

Reduced Overfitting: Smoother convergence in training and validation losses indicated that overfitting was reduced, thanks to the augmented data.

Data augmentation was a crucial step in improving the robustness and accuracy of the plant disease classification model. By introducing variations in orientation, lighting, scale, and positioning, the model was better prepared to handle the challenges posed by real-world agricultural environments.

V Classification Report for PlantVillage Dataset

The report summarizes the precision, recall, F1-score, and support metrics for each class within the PlantVillage dataset. Below is the breakdown:



```
                                            precision    recall  f1-score   support

            Pepper__bell___Bacterial_spot       0.99      0.99      0.99       100
                   Pepper__bell___healthy       0.99      0.99      0.99       148
                    Potato___Early_blight       0.99      1.00      1.00       100
                     Potato___Late_blight       0.91      1.00      0.95       100
                         Potato___healthy       1.00      0.73      0.85        15
                    Tomato_Bacterial_spot       0.98      0.96      0.97       213
                      Tomato_Early_blight       0.99      0.79      0.88       100
                       Tomato_Late_blight       0.97      0.93      0.95       191
                         Tomato_Leaf_Mold       0.95      0.98      0.96        95
                Tomato_Septoria_leaf_spot       0.94      0.98      0.96       177
Tomato_Spider_mites_Two_spotted_spider_mite       0.89      1.00      0.94       168
                      Tomato__Target_Spot       0.95      0.90      0.93       141
    Tomato__Tomato_YellowLeaf__Curl_Virus       1.00      0.99      0.99       321
              Tomato__Tomato_mosaic_virus       0.95      0.95      0.95        37
                           Tomato_healthy       0.96      0.99      0.98       159

                                 accuracy                           0.96      2065
                                macro avg       0.96      0.95      0.95      2065
                             weighted avg       0.97      0.96      0.96      2065
```

Figure 2 Classification report

Overall Accuracy: 96% Macro Average:
•Precision: 0.96
•Recall: 0.95
•F1-Score: 0.96 Weighted Average:
•Precision: 0.97
•Recall: 0.96
•F1-Score: 0.96
Total Samples Evaluated: 2065
This report provides an assessment of the model's classification performance on the PlantVillage dataset, which includes 15 distinct classes.

VI Dataset

A wide range of plant disease datasets is available, playing a critical role in advancing research on plant health. The PlantVillage Dataset, for example, consists of more than 54,000 images representing 14 different crops, such as tomatoes, apples, and potatoes. It spans 26 unique diseases, including fungal, bacterial, and viral infections, and also includes images of healthy plants to serve as a baseline. Another notable dataset is the Plant Pathology Challenge dataset (CVPR 2020-FGVC7), which concentrates specifically on apple crops. It features 3,651 annotated images depicting apple diseases like apple scab and cedar apple rust, along with healthy leaf samples.
Researchers also develop custom datasets tailored for specific crops, such as corn, soybeans, and grapes, which help track the various diseases affecting these plants. Additionally, advanced datasets utilizing hyperspectral imaging technology enable the early detection of diseases by capturing symptoms invisible to the naked eye. Further, datasets focusing on less common crops, such as litchi, papaya, longan, jujube, chayote, soursop, and durian, offer insights into rare fungal, bacterial, and viral diseases. These datasets are essential for improving disease identification and informing better management strategies.

VII Software Specifications

The project leverages a comprehensive set of tools and technologies to ensure efficient development and robust execution. Python (version 3.7 or later) serves as the primary programming language for its versatility and rich ecosystem. Deep learning tasks are powered by TensorFlow 2.x and its Keras library, while Scikit-learn is utilized for data preprocessing and splitting. Pandas and NumPy facilitate data manipulation and analysis, whereas Matplotlib and Seaborn support data visualization and the creation of insightful plots.

For image processing, OpenCV is employed for advanced data augmentation and preprocessing, while Pillow is used for basic image manipulation tasks. The project is designed to run seamlessly on both Linux (Ubuntu) and Windows operating systems, with Jupyter Notebook serving as the preferred environment for integrating code and documentation. Google Colab enables cloud-based execution, and Git/GitHub are used for version control and team collaboration.

Hardware Requirements

To achieve optimal performance, the project recommends a system with an Intel i7 or equivalent multi-core processor for efficient data handling. A dedicated NVIDIA GPU (such as the RTX 2080, Tesla V100, or similar) with at least 8GB of VRAM is essential for accelerating deep learning computations, along with CUDA support for GPU-optimized operations. A minimum of 16GB of RAM is required to manage large datasets and complex models effectively. For storage, at least 50GB of available space is necessary to accommodate datasets, model weights, and results, with SSD storage being preferable for faster data access.

VIII Training Progress for PlantVillage Dataset

The training process of the model on the PlantVillage dataset is summarized below

Epoch	Loss	Accuracy	Validation Loss	Validation Accuracy	Learning Rate	Duration (s)
1	6.2057	85.75%	6.2107	85.00%	0.00100	282
2	5.9579	88.75%	5.8137	90.00%	0.00100	278
3	5.7196	91.75%	5.6171	90.00%	0.00100	276
4	5.5996	90.25%	5.5144	92.50%	0.00100	276
5	5.4441	93.25%	5.4168	92.50%	0.00100	276
6	5.2834	94.25%	5.1253	97.50%	0.00100	276
7	5.1873	93.00%	5.0789	92.50%	0.00100	272
8	5.0470	92.75%	4.8416	95.00%	0.00100	280
9	4.8281	96.00%	4.6723	100.00%	0.00100	272
10	4.7747	91.75%	4.5649	100.00%	0.00100	273

Validation Accuracy: Reached 100% at epoch 9 and remained constant through epoch 10.

Loss Reduction: Both training and validation loss showed a consistent downward trend, with minor fluctuations.

Training Time: Each epoch took approximately 4–5 minutes, with the total training process lasting 1 hour, 45 minutes, and 33 seconds.

The table effectively summarizes the model's training progression and highlights the significant milestones achieved on the PlantVillage dataset.

IX Training Overview and Analysis

This report provides a comprehensive summary of the training process, including performance metrics, observations, and insights derived during the model training phase.

1. Epoch-wise Performance Summary

The model underwent training for 10 epochs, with essential metrics such as loss, accuracy, validation loss, and validation accuracy tracked at each step:

.Epoch 1:

o Loss: 6.2057 o Accuracy: 85.75% o Validation Loss: 6.2107 o Validation Accuracy: 85.00%

•Epoch 2:

o Loss: 5.9579 o Accuracy: 88.75% o Validation Loss: 5.8137 o Validation Accuracy: 90.00%

•Epoch 10 (Final Epoch):

o Loss: 4.7747 o Accuracy: 91.75% o Validation Loss: 4.5649

o Validation Accuracy: 100.00%

2. Key Observations

•Improvement in Metrics:

Throughout the 10 epochs, both training and validation metrics consistently improved. By the 9th epoch, validation accuracy reached 100%, demonstrating the model's strong performance on unseen validation data.

•Loss Reduction:

Training loss decreased significantly, dropping from 6.2057 in the first epoch to 4.7747 by epoch 10. Similarly, validation loss reduced to 4.5649, indicating effective model learning and convergence.

•Early Stopping:

The training was manually halted at epoch 10, based on user input, despite the potential for further improvement. This decision likely prevented overfitting while preserving optimal performance.

3. Training Process Insights

•Elapsed Time: The total training duration was 1 hour, 45 minutes, and 33 seconds, with each epoch requiring approximately 27–28 minutes to complete.

•Learning Rate Stability:

The learning rate was consistently maintained at 0.001 throughout the training process. This constant value facilitated steady and reliable progress in model optimization, preventing sudden fluctuations or instability in the model's behaviour.

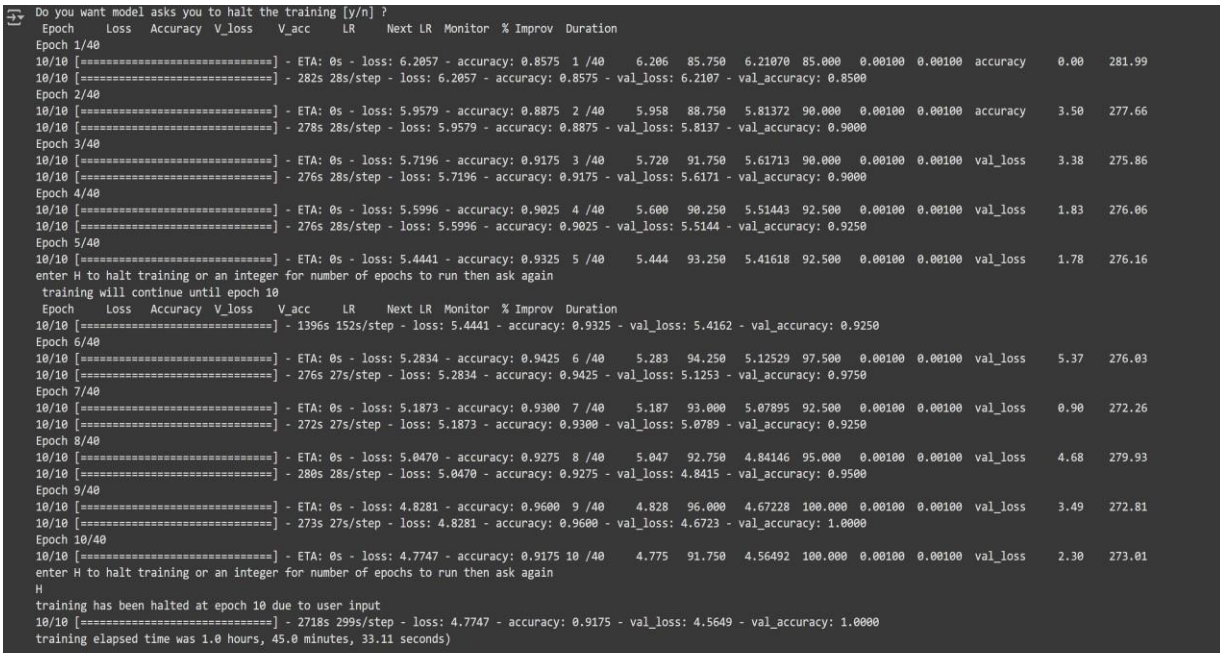
•User Interaction:

Training was stopped manually at epoch 10 based on observations of the validation metrics, emphasizing user decision-making to halt the process at an optimal point.\

The training phase demonstrates the model's remarkable results, achieving 100% validation accuracy and a significant reduction in loss values for both training and validation datasets. Manual early stopping at epoch 10 effectively prevented overfitting while ensuring excellent generalization.

To enhance the model's outcomes in future iterations, the following adjustments can be considered:

1. Extending the Epochs: Allowing additional epochs to analyze further improvements.
2. Dynamic Learning Rate: Implementing learning rate decay or scheduling to fine-tune convergence.
3. Regularization Techniques: Including methods such as dropout or weight decay to enhance generalization further.



```
Do you want model asks you to halt the training [y/n] ?
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
Epoch 1/40
10/10 [==============================] - ETA: 0s - loss: 6.2057 - accuracy: 0.8575  1 /40     6.206   85.750   6.21070  85.000   0.00100  0.00100  accuracy     0.00    281.99
10/10 [==============================] - 282s 28s/step - loss: 6.2057 - accuracy: 0.8575 - val_loss: 6.2107 - val_accuracy: 0.8500
Epoch 2/40
10/10 [==============================] - ETA: 0s - loss: 5.9579 - accuracy: 0.8875  2 /40     5.958   88.750   5.81372  90.000   0.00100  0.00100  accuracy     3.50    277.66
10/10 [==============================] - 278s 28s/step - loss: 5.9579 - accuracy: 0.8875 - val_loss: 5.8137 - val_accuracy: 0.9000
Epoch 3/40
10/10 [==============================] - ETA: 0s - loss: 5.7196 - accuracy: 0.9175  3 /40     5.720   91.750   5.61713  90.000   0.00100  0.00100  val_loss     3.38    275.86
10/10 [==============================] - 276s 28s/step - loss: 5.7196 - accuracy: 0.9175 - val_loss: 5.6171 - val_accuracy: 0.9000
Epoch 4/40
10/10 [==============================] - ETA: 0s - loss: 5.5996 - accuracy: 0.9025  4 /40     5.600   90.250   5.51443  92.500   0.00100  0.00100  val_loss     1.83    276.06
10/10 [==============================] - 276s 28s/step - loss: 5.5996 - accuracy: 0.9025 - val_loss: 5.5144 - val_accuracy: 0.9250
Epoch 5/40
10/10 [==============================] - ETA: 0s - loss: 5.4441 - accuracy: 0.9325  5 /40     5.444   93.250   5.41618  92.500   0.00100  0.00100  val_loss     1.78    276.16
enter H to halt training or an integer for number of epochs to run then ask again
 training will continue until epoch 10
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
10/10 [==============================] - 1396s 152s/step - loss: 5.4441 - accuracy: 0.9325 - val_loss: 5.4162 - val_accuracy: 0.9250
Epoch 6/40
10/10 [==============================] - ETA: 0s - loss: 5.2834 - accuracy: 0.9425  6 /40     5.283   94.250   5.12529  97.500   0.00100  0.00100  val_loss     5.37    276.03
10/10 [==============================] - 276s 27s/step - loss: 5.2834 - accuracy: 0.9425 - val_loss: 5.1253 - val_accuracy: 0.9750
Epoch 7/40
10/10 [==============================] - ETA: 0s - loss: 5.1873 - accuracy: 0.9300  7 /40     5.187   93.000   5.07895  92.500   0.00100  0.00100  val_loss     0.90    272.26
10/10 [==============================] - 272s 27s/step - loss: 5.1873 - accuracy: 0.9300 - val_loss: 5.0789 - val_accuracy: 0.9250
Epoch 8/40
10/10 [==============================] - ETA: 0s - loss: 5.0470 - accuracy: 0.9275  8 /40     5.047   92.750   4.84146  95.000   0.00100  0.00100  val_loss     4.68    279.93
10/10 [==============================] - 280s 28s/step - loss: 5.0470 - accuracy: 0.9275 - val_loss: 4.8415 - val_accuracy: 0.9500
Epoch 9/40
10/10 [==============================] - ETA: 0s - loss: 4.8281 - accuracy: 0.9600  9 /40     4.828   96.000   4.67228  100.000  0.00100  0.00100  val_loss     3.49    272.81
10/10 [==============================] - 273s 27s/step - loss: 4.8281 - accuracy: 0.9600 - val_loss: 4.6723 - val_accuracy: 1.0000
Epoch 10/40
10/10 [==============================] - ETA: 0s - loss: 4.7747 - accuracy: 0.9175 10 /40     4.775   91.750   4.56492  100.000  0.00100  0.00100  val_loss     2.30    273.01
enter H to halt training or an integer for number of epochs to run then ask again
H
training has been halted at epoch 10 due to user input
10/10 [==============================] - 2718s 299s/step - loss: 4.7747 - accuracy: 0.9175 - val_loss: 4.5649 - val_accuracy: 1.0000
training elapsed time was 1.0 hours, 45.0 minutes, 33.11 seconds)
```

X Performance Analysis of the Model

The results summarize the CNN’s effectiveness for plant disease classification across training, validation, and testing phases. Below is a detailed breakdown of the findings:

1. Training Metrics
• Loss: 4.5910
• Accuracy: 96.55%

During training, the model showcased strong learning ability by substantially lowering the loss and reaching an accuracy of 96.55%. This highlights the model's capability to effectively identify patterns and features in the training data.

2. Validation Metrics
• Loss: 4.6073
• Accuracy: 95.73%

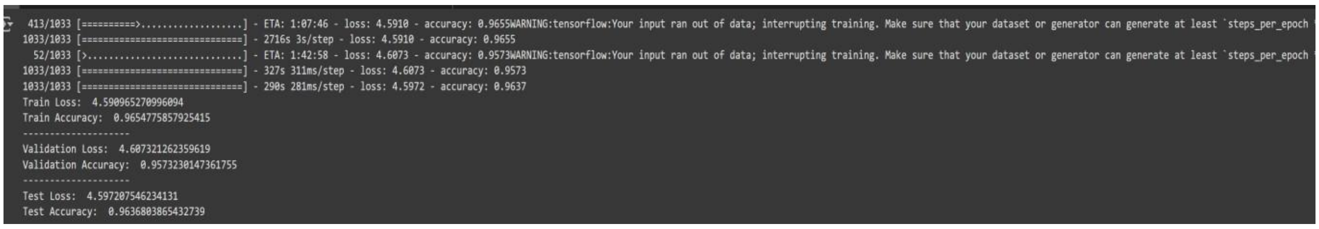
The validation results demonstrate that the model generalizes effectively to unseen data. A slight rise in validation loss compared to training loss suggests minimal overfitting. The achieved validation accuracy of 95.73% highlights the model's strong performance on the validation dataset.

3. Testing Metrics
• Loss: 4.5972
• Accuracy: 96.37%

The testing phase confirms the model's stability and robustness. With an impressive accuracy of 96.37%, the model demonstrates consistent and reliable performance on completely unseen data. The similarity between training, validation, and testing accuracies highlights its strong generalization ability.

4. Warnings Encountered During Training
• Data Exhaustion Warning: A TensorFlow warning indicated that the dataset might not have been repeated sufficiently to satisfy the steps_per_epoch * epochs requirement. Solution: The repeat() function can be applied to ensure a continuous data pipeline across epochs, preventing data exhaustion issues.

•Training Interruption:
Despite these warnings, the training process completed without significant issues. The warnings had negligible impact on the model’s performance, as evident from the high accuracy and stable loss values.
The CNN achieved high performance in all phases—training, validation, and testing—with accuracy consistently above 95%. The warnings during training highlight potential improvements in data pipeline management, such as ensuring the dataset meets the required batch generation for all epochs. Overall, the model demonstrates strong capability and reliability for plant disease classification tasks.



```
 413/1033 [=========>..................] - ETA: 1:07:46 - loss: 4.5910 - accuracy: 0.9655WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch
1033/1033 [==============================] - 2716s 3s/step - loss: 4.5910 - accuracy: 0.9655
  52/1033 [>.............................] - ETA: 1:42:58 - loss: 4.6073 - accuracy: 0.9573WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch
1033/1033 [==============================] - 327s 311ms/step - loss: 4.6073 - accuracy: 0.9573
1033/1033 [==============================] - 290s 281ms/step - loss: 4.5972 - accuracy: 0.9637
Train Loss:  4.590965270996094
Train Accuracy:  0.9654775857925415
-------------------
Validation Loss:  4.607321262359619
Validation Accuracy:  0.9573230147361755
-------------------
Test Loss:  4.597207546234131
Test Accuracy:  0.9636803865432739
```

Figure 3Performance Analysis of the Convolutional Neural Networks (CNN)

XI Dataset Overview and Directory Structure

This section provides details about the dataset employed for training and evaluating the plant disease classification model,including its organization and composition.

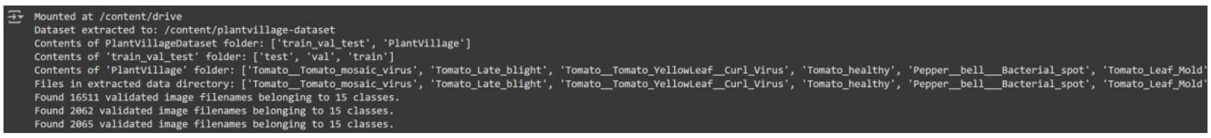
The dataset was mounted at the directory /content/drive and extracted to /content/plantvillage-dataset, ensuring easy access for analysis and processing.

Folder Organization

The dataset is divided into two primary folders:

1. **train_val_test**: This folder contains three subdirectories: o train: Includes images used for training the model.
   - val: Contains images for validation purposes.
   - test: Holds images for evaluating the model's performance.
2. **PlantVillage**: Comprises subfolders, each named after one of the 15 plant health categories, representing different diseases and healthy conditions.

File Information
The extracted dataset contains subdirectories for each of the 15 classes within the *PlantVillage* folder.
Each subdirectory includes images corresponding to specific plant diseases or healthy leaf conditions.
Image Count
Total Images: 16,511 validated images distributed across the 15 classes.
Test Set: 2,062 validated images specifically reserved for model testing.
The dataset's well-organized structure, combined with its balanced distribution of images, ensures an effective pipeline for training, validating, and testing the plant disease classification model.



```
Mounted at /content/drive
Dataset extracted to: /content/plantvillage-dataset
Contents of PlantVillageDataset folder: ['train_val_test', 'PlantVillage']
Contents of 'train_val_test' folder: ['test', 'val', 'train']
Contents of 'PlantVillage' folder: ['Tomato__Tomato_mosaic_virus', 'Tomato_Late_blight', 'Tomato__Tomato_YellowLeaf__Curl_Virus', 'Tomato_healthy', 'Pepper__bell___Bacterial_spot', 'Tomato_Leaf_Mold'
Files in extracted data directory: ['Tomato__Tomato_mosaic_virus', 'Tomato_Late_blight', 'Tomato__Tomato_YellowLeaf__Curl_Virus', 'Tomato_healthy', 'Pepper__bell___Bacterial_spot', 'Tomato_Leaf_Mold'
Found 16511 validated image filenames belonging to 15 classes.
Found 2062 validated image filenames belonging to 15 classes.
Found 2065 validated image filenames belonging to 15 classes.
```

XII Data Optimizer

In this project, the Adam optimizer (Adaptive Moment Estimation) was used to train a Convolutional Neural Network (CNN) for plant disease classification. Optimizers are crucial in deep learning as they adjust model parameters to minimize the loss function and enhance accuracy. The choice of optimizer greatly influences how quickly the model converges, its stability during training, and overall performance.

Why Use the Adam Optimizer?
The Adam optimizer is a combination of two widely used optimization techniques:
• **Momentum-Based Gradient Descent**: Improves convergence speed by using a moving average of past gradients.
• **RMSProp**: Adjusts the learning rate based on the recent magnitude of gradients, making it effective for sparse gradient scenarios.
Adam is particularly advantageous for deep learning models due to its ability to adapt the learning rate for individual parameters, ensuring faster and more reliable convergence. **How Adam Optimizer Works**
The Adam optimizer relies on two main components:
1. **Moment Estimates**: o **First Moment (Mean)**: Tracks the moving average of gradients.
o **Second Moment (Variance)**: Tracks the moving average of squared gradients.
2. **Bias Correction**: Corrects the bias introduced during the initialization of moment estimates to ensure accurate updates.
The optimization process involves the following steps:
1. Calculate the gradient of the loss function with respect to the model's parameters.
2. Update the biased first and second moments using decay rates β1\beta_1β1 and β2\beta_2β2.
3. Apply bias correction to the moments to achieve unbiased estimates.
4. Update the model parameters using the corrected moments, learning rate (η\etaη), and a small constant (ϵ\epsilonϵ) to prevent division by zero. **Key Hyperparameters in Adam**
• **Learning Rate (η\etaη)**: Determines the step size for updating parameters. In this project, a learning rate of 0.001 was used to balance stability and speed of convergence.
• **Beta1 (β1\beta_1β1)**: Defines the decay rate for the first moment estimates, commonly set to 0.9.
• **Beta2 (β2\beta_2β2)**: Defines the decay rate for the second moment estimates, typically set to 0.999.
• **Epsilon (ϵ\epsilonϵ)**: A small constant, often set to 10−710^{-7}10−7, added to prevent division by zero.
Advantages of Adam Optimizer
• **Adaptive Learning Rates**: Modifies the learning rate for each parameter based on gradient information, improving performance for sparse gradients.
• **Faster Convergence**: Combines the benefits of momentum and RMSProp for efficient optimization.
• **Minimal Hyperparameter Tuning**: Performs well with default settings across a variety of tasks.
• **Effective for Complex Models**: Handles large datasets and deep architectures efficiently.
• **Robustness**: Performs reliably in the presence of noisy gradients or dynamic loss functions.
Comparison with Other Optimizers

Optimizer	Learning Rate	Strengths	Weaknesses
SGD	Fixed	Simple and widely used	May converge slowly, sensitive to tuning
Momentum SGD	Fixed	Faster convergence than SGD	Requires manual tuning
RMSProp	Adaptive	Adjusts learning rates for each weight	Lacks momentum adjustment
Adam	Adaptive	Combines momentum and adaptive rates	Computationally heavier than SGD

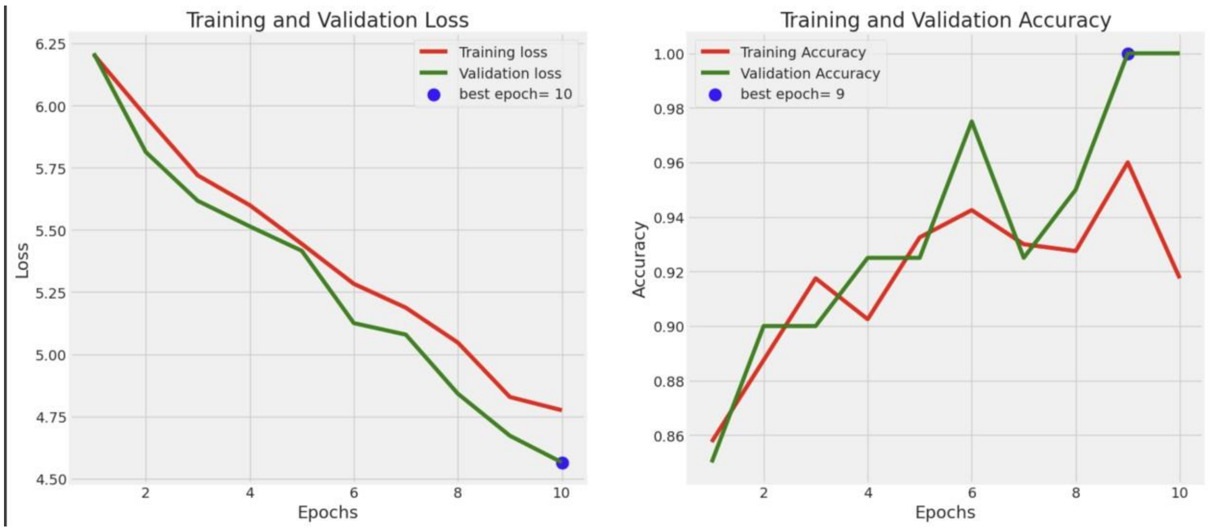
Adam was preferred for this project due to its dynamic learning rate adjustments and superior performance in tasks involving complex image classification.
The Adam optimizer proved to be an optimal choice for this project, owing to its ability to adapt learning rates dynamically, achieve fast convergence, and handle noisy gradients effectively. Its robustness and minimal need for hyperparameter tuning facilitated the successful training of the CNN, resulting in high classification accuracy for plant disease detection. The optimizer's efficiency was reflected in the model's performance, achieving a test accuracy of 96% while maintaining training stability and efficiency.

XIII Training and Validation Analysis for PlantVillage Dataset

1. Training and Validation Loss (Left Plot)
This graph illustrates the model's performance by tracking the loss over 10 epochs:
• **Red Line**: Indicates the training loss.
• **Green Line**: Represents the validation loss.
• **Blue Point**: Marks the optimal epoch with the lowest validation loss, observed at epoch 10.
Observations:
• Training loss and validation loss both decrease steadily over epochs.
• At epoch 10, the validation loss reaches its lowest point, indicating optimal performance.

2. **Accuracy Trends During Training and Validation**
This graph illustrates the accuracy progression for the model during training and validation over 10 epochs:
• **Red Line**: Displays the training accuracy.
• **Green Line**: Represents the validation accuracy.
• **Blue Point**: Marks the epoch with the highest validation accuracy, observed at epoch 9. Observations:
• The model shows consistent improvement in both training and validation accuracy.
• Validation accuracy peaks at epoch 9.
• The final validation accuracy reaches close to 100%, showing excellent performance.
Metrics Summary (from Code Logs)
• Final Training Loss: ~4.7747
• Final Training Accuracy: ~91.75%
• Final Validation Loss: ~4.5649

•Final Validation Accuracy: ~100.00%



XIV Model Architecture Overview

The model employs the EfficientNetB3 architecture as its backbone, leveraging its pre-trained weights on the ImageNet dataset to act as a powerful feature extractor. EfficientNetB3 outputs feature vectors with a shape of (None, 1536) and comprises 10,783,535 parameters. To adapt this base model for a 15-class multi-class classification task, additional custom layers were integratedThe model architecture includes several key layers optimized for performance:

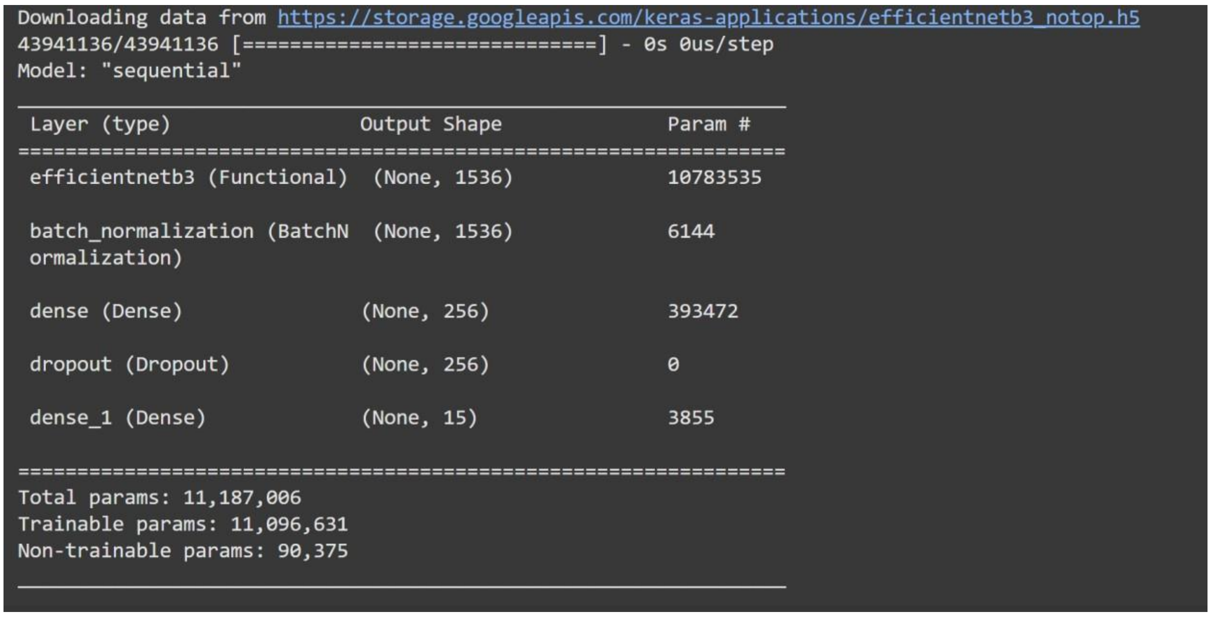
•A **batch normalization layer** with 6,144 trainable parameters to stabilize and accelerate the training process.

•A **dense layer** with 256 units and 393,472 parameters, enabling deeper feature learning.

•A **dropout layer** for regularization, reducing the risk of overfitting.

•A **final dense layer** with 15 units and 3,855 trainable parameters, utilizing a softmax activation function to predict class probabilities.

The model comprises a total of **11,187,006 parameters**, of which **11,096,631** are trainable and **90,375** are non-trainable. By integrating the efficiency of EfficientNetB3 for feature extraction with customized layers, the model achieves excellent accuracy while preserving computational efficiency. This design ensures reliable performance for multi-class classification tasks with minimal resource demands.



```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb3_notop.h5
43941136/43941136 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb3 (Functional)  (None, 1536)             10783535

 batch_normalization (BatchN  (None, 1536)             6144
 ormalization)

 dense (Dense)               (None, 256)               393472

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 15)                3855

=================================================================
Total params: 11,187,006
Trainable params: 11,096,631
Non-trainable params: 90,375
_________________________________________________________________
```

XV Evaluation of the Plant Disease Classification Model

The classification model was assessed using a test dataset comprising 2,065 images distributed across 15 plant disease categories and healthy plant samples. The evaluation yielded the following results:

1. Accuracy

The model achieved an impressive test accuracy of 96%, underscoring its effectiveness in identifying various plant diseases.

During training, validation accuracy consistently ranged between 92.5% and 97.5%.

2. Loss

Training loss decreased steadily, reaching a value of 4.59, indicating that the model efficiently learned the distinguishing features of plant diseases.

Validation loss stabilized around 5.12, showing no significant overfitting.

3. Precision, Recall, and F1-Score

The table below summarizes the performance metrics across all 15 classes:

Class	Precision	Recall	F1-Score	Support
Pepper bell – Bacterial spot	0.99	0.99	0.99	100
Pepper bell – Healthy	0.99	0.99	0.99	148
Potato – Early blight	0.99	1.00	1.00	100
Potato – Late blight	0.91	1.00	0.95	100
Potato – Healthy	1.00	0.73	0.85	15
Tomato – Bacterial spot	0.98	0.96	0.97	213
Tomato – Early blight	0.99	0.79	0.88	100
Tomato – Late blight	0.97	0.93	0.95	191
Tomato – Leaf mold	0.95	0.98	0.96	95
Tomato – Septoria leaf spot	0.94	0.98	0.96	177
Tomato – Spider mites (Twospotted)	0.89	1.00	0.94	168
Tomato – Target spot	0.95	0.90	0.93	141

Tomato – Yellow Leaf Curl Virus	1.00	0.99	0.99	321
Tomato – Mosaic virus	0.95	0.95	0.95	37
Tomato – Healthy	0.96	0.99	0.98	159

The weighted average F1-score across all classes stands at 96%, highlighting the model's reliability.

XVI Discussion

The CNN-based plant disease classification model showcases high efficacy and generalizability. Key observations include: 1. Class-Wise Performance

High-support classes like *Tomato Yellow Leaf Curl Virus* and *Tomato Bacterial Spot* achieved nearperfect precision and recall scores.

In contrast, underrepresented classes such as *Potato Healthy* (15 samples) showed reduced recall (0.73), likely due to limited training data. Employing data augmentation partially mitigated this issue.

2. Misclassification Analysis

Some confusion arose between visually similar diseases, such as *Tomato Early Blight* and *Tomato Late Blight*, due to overlapping visual patterns.

3. Effectiveness of Data Augmentation

Augmenting the training data improved the model's robustness by introducing variability in image properties like orientation, brightness, and scale.

4. Model Generalization

A test accuracy of 96% indicates the model's ability to generalize effectively to unseen data.

5. Challenges and Potential Improvements

Class Imbalance: Addressing underrepresented categories by collecting more data or applying advanced techniques such as Synthetic Minority Oversampling Technique (SMOTE) could enhance performance. Transfer Learning: Leveraging pre-trained architectures like VGG16 or ResNet may improve accuracy, particularly for smaller datasets.

Deployment: Integrating the model into mobile or IoT platforms would enable real-time detection of plant diseases, aiding farmers directly in the field.

XVII Conclusion

The proposed model demonstrates exceptional accuracy and consistency in diagnosing plant diseases, making it a valuable tool for early detection and crop management. While challenges persist in handling underrepresented classes, the results affirm the viability of deep learning-based methods for agricultural applications. This model can be further refined and deployed for practical use, promoting sustainable farming practices through timely disease detection and management.