Saniya G S, 2026, 14:5
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Crop Disease Detection and Classification

Saniya G S[1] Maaz Ahmed[2] Murali Karthik Nayaka P J[3]
Dept. Of Computer Science And Engineering (Ai & Ml
Don Bosco Institute Of Technology Bengaluru, India

Abstract—This paper presents a web-based system for classifying tomato leaf images as bacterial spot, early blight, late blight, or healthy. It uses MobileNetV2, initialized with ImageNet weights, as a frozen feature extractor, with a small trainable classification head. The model was trained on 6,644 images from a PlantVillage-based dataset. Images were resized to 128 × 128 pixels, and training images were augmented with random flips, rotations, and zoom. On a fixed, random 20% image-level validation split, the saved evaluation report shows 94.43% accuracy and a weighted F1-score of 94.32%. However, performance varied by class: early blight had the lowest recall at 79.34%. The accompanying Streamlit app allows users to upload a leaf image and view its predicted class, confidence, and available disease information. Since the validation images come from the same controlled dataset and were split randomly by image, these results do not indicate how well the system would perform on field photographs or images from independent farms..

Keywords—crop disease classification, plant leaf image, transfer learning, MobileNetV2, PlantVillage, Streamlit.

I. INTRODUCTION

Plant diseases can reduce crop quality and yield, and visible symptoms are not always easy to distinguish without experience. Image classification offers a way to make an initial screening step more accessible: a user provides a leaf photograph and a trained model returns one of the conditions represented in its training data. Prior work on the PlantVillage collection demonstrated that deep neural networks can learn disease-related patterns from leaf images, while also highlighting the importance of the dataset and image conditions used for evaluation [1].

This project turns that idea into a small, usable application. It combines a transfer-learning classifier with image preprocessing, a browser-based interface, and disease information for supported classes. The current model focuses on four tomato leaf conditions. Its contribution is the integrated training, evaluation, prediction, and user-interface workflow, rather than a new neural-network architecture.

The intended workflow is deliberately simple. A user uploads a photograph, the application prepares it in the format expected by the saved model, and the model returns class scores. The interface then shows the leading class and its score along with explanatory text when it is available. Keeping the workflow in one application makes the project easier to demonstrate and gives a non-specialist a concrete way to explore a trained image classifier.

The work addresses three practical questions: whether a pretrained image model can separate the four selected leaf conditions in a held-out subset; how its class-wise performance differs; and how the trained model can be presented through a browser interface. These questions describe a prototype evaluation. They do not test whether the classifier improves farm outcomes or replaces in-person diagnosis.

The paper is organized as follows. Section II summarizes the technical background. Section III describes the dataset and implementation. Section IV reports the available validation results. Section V discusses their limits, and Section VI concludes the paper.

1. Technical Background

PlantVillage is a public collection of healthy and diseased leaf images that has been widely used to study image-based plant disease classification. Mohanty et al. evaluated deep learning methods on this collection and reported strong performance under their experimental setup [1]. The original collection contains 54,306 images across 14 crop species and 26 diseases, as well as healthy leaves [1]. The present

© 2026 Saniya G S, This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

project uses a much smaller four-class subset, so its validation result measures this implementation on its own split and should not be compared directly with published scores from broader experiments.

Recent studies have explored models with more sophisticated feature extractors. Feng et al. combine EfficientNet convolutional features with group-wise transformer attention [2]. Li et al. introduce a convolutional self-guided transformer for recognising plant diseases across different environments [3], while Chen et al. use channel attention and lightweight upsampling for crop-disease detection [4]. Unlike image-level classification, object detection identifies and locates disease-related objects. The studies reflect different design priorities, and their results cannot be compared directly because they use different datasets, class sets, tasks, and evaluation protocols.

Other research has examined ensemble learning with explainable outputs [5], transfer learning and hyperparameter optimisation [6], and crop-specific methods, including learning-rate annealing for groundnut disease [7]. One two-stage framework first identifies the plant species and then classifies the disease [8]. Yu et al. combine K-means clustering with deep learning to diagnose corn leaf diseases [9]. These approaches show how dataset scope and system goals influence model design. In contrast, this project uses a single frozen MobileNetV2 backbone with a compact, trainable head to classify four tomato-leaf labels.

A plant disease image classifier learns visual associations between an image and the label assigned to it. In a leaf photograph, potentially useful signals include changes in color, texture, shape, and the distribution of spots or lesions. Those signals can overlap across diseases, and a photograph can contain shadows, soil, or other background objects. A model may therefore learn both disease-related appearance and accidental patterns in its training images. Evaluation on a held-out subset is necessary, but a split drawn from the same image collection cannot by itself show that those patterns transfer to another setting.

MobileNetV2 was designed as a compact vision architecture using inverted residual blocks and linear bottlenecks [10]. In this project, the pretrained feature extractor is kept frozen and a small classifier is trained on top of it. This transfer-learning setup reuses visual features learned from ImageNet and limits training to the new classification layers. The implementation and supported classes are available in the project repository [11].

In an inverted residual block, feature maps are expanded before applying depthwise convolution and then projected into a narrower representation [10]. This structure helps the network reuse useful visual transformations with a more efficient computation pattern than a conventional wide convolution at every stage. The project adopts the published backbone through Keras rather than reimplementing those blocks. The experimental question here is whether a frozen pretrained representation plus a compact head can distinguish the selected labels, not whether the backbone itself improves on other architectures.

The choice of a pretrained backbone is practical for a student project: it avoids training every convolutional layer from random initialization and keeps the model architecture manageable. Freezing the backbone also reduces the number of trainable parameters. The tradeoff is that the visual representation is not adapted through fine-tuning to the leaf dataset. The trainable classification head must learn to separate the four labels using the features already supplied by the frozen network.

II. DATA AND METHODS

1.Dataset And Split

The working dataset contains 6,644 images across four classes. The class names and counts are shown in Table I. The dataset documentation identifies the PlantVillage repository as the source of the added color images; it also records 31 retained project images and removal of 14 exact duplicate copies. Images are loaded from class-specific folders, so each folder name supplies the target label.

The classes are not equally represented. Bacterial spot is the largest group, with 2,134 images, while early blight is the smallest, with 1,009. Late blight and healthy leaves contribute 1,908 and 1,593 images, respectively. The largest class is a little more than twice the size of the smallest. This difference does not prove that class imbalance caused the weaker early-blight

recall, but it is relevant when interpreting an overall score and planning further data collection.

A class label comes from the image's directory name, which keeps training labels easy to inspect and extend. The training

process writes the class-name order alongside the model, and the evaluation and prediction code reuse that order when interpreting the output vector. Preserving this mapping is essential: a correct probability vector paired with a different class order would produce a plausible-looking but incorrect disease name.

Table I. Dataset Composition And Validation Support

Class	All images	Validation
Bacterial spot	2,134	394
Early blight	1,009	213
Late blight	1,908	379
Healthy	1,593	342
Total	6,644	1,328

Training uses an 80/20 random image-level split with seed 42. This produces 5,316 training images and 1,328 validation images. Because the split is made at the image level, closely related images could appear on both sides; the validation set is not an independent, field-collected test set.

The same split settings are used by the evaluation script to rebuild the validation subset. The validation labels are read in the same pass as model predictions so that labels and outputs remain aligned when the data loader shuffles examples. This makes the saved report reproducible for the current data folders and seed, provided the image collection and class names have not changed since the model was trained.

A random image split is useful for checking whether the model can classify images drawn from the same source collection. It is a narrower question than generalization to new farms, cameras, or growing seasons. A stronger follow-up study would reserve images by plant, acquisition session, or source location before training, and would collect an additional test set in field conditions. Such grouping reduces the chance that near-identical views of a leaf influence both training and evaluation.

1. Image Preparation And Augmentation

Images are resized to 128 × 128 pixels and represented as three-channel RGB inputs. The model rescales pixel values by 1/255. During training, random horizontal flips, rotations, and zooms introduce modest visual variation. Validation images are not augmented.

Resizing provides a consistent input tensor even when users upload photographs with different dimensions. Converting to RGB also gives the model the three channels used during training. These steps standardize image shape and color representation; they do not correct blur, glare, poor framing, or a leaf that occupies only a small portion of the photograph. Input quality remains a factor in the predicted class and confidence.

Augmentation is performed only on the training batches, so the validation report reflects the original held-out images. Horizontal flipping can vary leaf orientation, while small rotations and zoom changes vary its position and scale. These transformations increase variation in the examples shown during training, but they cannot reproduce every field condition. They do not add new disease symptoms, backgrounds, lighting patterns, or camera characteristics that are absent from the source images.

2. Classifier And Training

The model starts with MobileNetV2 without its original classification layer and loads ImageNet weights. Its base layers are frozen. Global average pooling reduces the feature maps to a vector, followed by dropout (0.4), a 128-unit ReLU layer, a second dropout layer (0.3), and a four-unit softmax output. The model is optimized with Adam and sparse categorical cross-entropy. Training is configured for up to 40 epochs with early stopping on validation loss (patience 8) and learning-rate reduction when validation loss plateaus (patience 3).

The principal training settings are summarized in Table II to make the experiment easier to reproduce. The selected input size reduces image detail compared with the original photographs, but it keeps the input shape consistent between training and prediction. Batch size 16 controls how many images contribute to each parameter update. The fixed random seed is used

when forming the directory-based split; it does not remove the limitations of an image-level split.

Table II. Model And Training Configuration

Setting	Value
Backbone	MobileNetV2, ImageNet weights
Backbone training	Frozen
Input	128 × 128 × 3 RGB
Classifier head	GAP, dropout 0.4, Dense 128, dropout 0.3, softmax 4
Optimizer and loss	Adam; sparse categorical cross-entropy
Batch size / maximum epochs	16 / 40
Callbacks	Early stopping: 8; LR reduction: 3
Split and seed	80/20 image-level; seed 42

The final softmax layer assigns one score to each of the four dataset labels. During training, sparse categorical cross-entropy compares those scores with the integer class label. Adam updates the trainable classification head to reduce that loss. Dropout randomly masks some activations during training and is disabled during inference. Early stopping restores the best weights observed on validation loss, while the learning-rate callback lowers the step size after a plateau. These choices aim to stabilize learning on a dataset that is smaller than the broad image collection used to pretrain the backbone.

The training script saves the Keras model and class-name list under the models directory. It also creates a feature reference used by the application's input check and a graph of the training history. The separate evaluation script loads the saved model and writes the classification report. In this paper, numerical results are taken from that report rather than from the training-curve plot, so the claims match the metric artifact used for evaluation.

3. Prediction Application

The Streamlit interface accepts JPG, JPEG, and PNG uploads. An image is converted to RGB, resized to the model input size, and passed to the saved classifier. The interface presents the highest-scoring class and its probability as a confidence percentage, together with a healthy or diseased status and any matching description, symptoms, or general prevention guidance. A feature-distance check flags inputs unlike the supported leaf classes; it is an input-screening aid, not a separately validated disease classifier.

From the user's perspective, the interaction has six stages: (1) select a leaf photograph; (2) validate and decode the image; (3) resize and normalize it; (4) compute class scores with the saved model; (5) check whether its feature representation is sufficiently similar to supported examples; and (6) display a result with class information. The model is loaded once and reused by the application. This avoids the delay and resource cost of training during an upload session.

The feature-distance screen is intended to reduce confident predictions for images that differ substantially from the supported leaf classes. It compares an image representation with stored reference information and can cause the app to mark an input as unsupported. Its threshold is calibrated from held-out examples in the training collection. That setup does not establish how often it rejects real-world distractors or how often it rejects valid leaves photographed under new conditions; these rates need a separately designed test set.

The software is organized around separate training, evaluation, and prediction steps. Training is run when the dataset or model needs to change; the application loads the saved model and does not retrain it when a person opens the web page. This keeps model development separate from everyday inference. The command-line prediction script provides another way to submit a single image, while the Streamlit interface is intended for interactive use.

The interface also handles common input and setup problems, including a missing model, an unsupported file, or an unreadable image. Disease descriptions are stored separately from the model and can be edited without changing the classifier. If there is no entry for a predicted class, the application can present a fallback message. This separation makes the display content easier to maintain, but it does not make the disease guidance a clinical or agricultural diagnosis.

The application keeps inference in the local Python process that serves the Streamlit page. Uploaded bytes are decoded and passed through the shared preprocessing helper, which reduces the chance that command-line and web predictions silently use

different image sizes or color handling. The model is cached for reuse during the session. This simple design suits a project demonstration; a public service would also need explicit decisions about image retention, access control, operational monitoring, and how users can report an incorrect prediction.

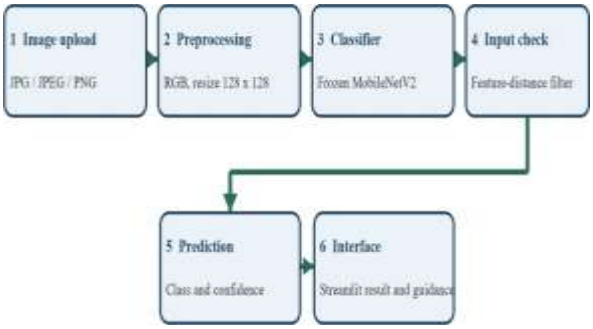


Fig. 1. Image-processing and prediction workflow implemented in the application.

IV. EXPERIMENTAL RESULTS

The results below are taken from the project's saved classification report, which evaluates the fixed validation split described in Section III. The report contains 1,328 validation images and gives an accuracy of 94.43%. Table III lists class-level precision, recall, and F1-score. Macro F1 is 93.33%, while weighted F1 is 94.32%.

For a four-class problem, accuracy is the fraction of validation images whose predicted label matches the reference label. For an individual class, precision measures how many predictions of that class are correct, and recall measures how many reference examples of that class are found. The F1-score is the harmonic mean of precision and recall. Macro averages give each class equal weight; weighted averages give more influence to classes with more validation examples. Reporting both averages makes the effect of the class distribution visible.

Using the usual notation, accuracy is the number of correct predictions divided by the total number of validation images. For class c, precision is TP_c/(TP_c + FP_c), recall is TP_c/(TP_c + FN_c), and F1_c is 2 × precision_c × recall_c/(precision_c + recall_c). Here TP, FP, and FN denote true positives, false positives, and false negatives for that class. These definitions make clear why precision and recall answer different questions: a model can make few incorrect positive predictions while still missing a substantial share of the true examples.

Table III. Performance On The Held-Out Image Validation Split

Class	Precision	Recall	F1-score
Bacterial spot	96.47%	97.21%	96.84%
Early blight	90.86%	79.34%	84.71%
Late blight	91.71%	96.31%	93.95%
Healthy	97.12%	98.54%	97.82%
Macro average	94.04%	92.85%	93.33%
Weighted average	94.38%	94.43%	94.32%

The largest class-level weakness is early blight recall: the model identifies 79.34% of validation examples from that class. Its precision is higher (90.86%), which means predictions labeled early blight are more reliable than the recall alone suggests, but some true early-blight images are assigned another class. The healthy class has the strongest recall at 98.54%. These figures describe the selected validation images only; the report does not provide an independent field evaluation.

Bacterial spot also has high precision and recall, at 96.47% and 97.21%. Late blight recall is 96.31%, with an F1-score of 93.95%. The differences between class scores matter in practical interpretation: a single accuracy value can hide errors concentrated in one category. In this run, early blight is the class most likely to be missed among its true validation examples, so it would be a priority for targeted review and additional testing.

The weighted F1-score exceeds macro F1 by 0.99 percentage points. The difference is modest, but it reflects that the larger classes contribute more to the weighted measure. The early-blight class has fewer validation examples and a lower recall, which weighs more heavily in the macro average. Neither average should be treated as an estimate of performance on an arbitrary field population, since the validation set preserves the source dataset's capture conditions and class distribution.

The current metrics describe classification among the four known labels. They do not score the feature-distance screen, image-quality handling, response time, or user understanding. They also do not report performance separately by camera, lighting, leaf age, or symptom severity. Such breakdowns would help identify where the model is dependable and where it needs more examples. The saved report is therefore a useful baseline for the prototype, rather than a complete system-level evaluation.

1. Discussion

The evaluation indicates that transfer learning can support a useful four-class demonstration with a moderate dataset. The web interface makes the trained model easier to try than a command-line-only workflow, and the class-specific information panel gives users context for a prediction. The confidence score is the model's highest softmax probability; it is not a calibrated probability that the diagnosis is correct.

The displayed confidence is useful as a summary of how the model ranked its own four outputs. It does not measure whether the photograph is from a supported crop, whether the leaf is healthy in a broader biological sense, or whether the suggested label is true. A model can be confidently wrong, especially when an image differs from the training distribution. The feature-distance check is intended to flag some unfamiliar inputs, but it requires a separate evaluation with supported leaves and non-leaf photographs before its rejection behavior can be quantified.

Several limitations matter for interpretation. First, PlantVillage images were collected under controlled conditions, while field photographs vary in lighting, background, camera quality, leaf age, and symptom severity [1]. Second, the random image-level split may overstate generalization if related images occur in both subsets. Third, the classes cover only four conditions; the classifier cannot identify unsupported conditions with dependable accuracy. The available numerical report does not provide class-to-class error counts, so the results are interpreted through the reported precision, recall, F1-scores, and class supports.

There are also limits in the current set of measured outcomes. The available report gives aggregate classification metrics and class supports, but it does not provide a field-based test, confidence calibration, or a measured rejection rate for unsupported images. Since the displayed score is the highest model output, users may interpret it as stronger evidence than it is. The interface should be understood as a preliminary classifier that can support inspection, with expert review needed before crop treatment decisions.

The dataset is a selected subset rather than a complete inventory of tomato diseases. A real plant may also show multiple symptoms, nutrient stress, insect damage, or an unfamiliar condition. The current output forces a choice among the trained labels unless the input check flags the image. It does not segment diseased regions, estimate severity, diagnose a pathogen in a laboratory sense, or recommend a treatment tailored to local conditions. Those capabilities would require additional data, validation, and expert review.

A stronger evaluation would use a new set of field photographs, preserve separation by plant or source where

possible, and report per-class confusion counts. Future development can expand the class set, improve early-blight recall, assess input screening separately, and measure inference time and resource use on the intended device. Any disease-management action should be checked with an agricultural expert.

A practical next experiment would be to collect examples of early blight across multiple farms and lighting conditions, then test a model trained without those images. Researchers could compare the current frozen-backbone classifier with a fine-tuned version, class-weighted training, or additional augmentation. The comparison should use the same fixed test set and report per-class precision, recall, F1, and confusion counts. A separate out-of-distribution evaluation should include unrelated plants and non-leaf images, and should measure both false rejection of valid leaves and acceptance of unsupported inputs.

For deployment, the application would also benefit from checking inference latency on the target computer or phone, assessing behavior with low-resolution and partially occluded leaves, and gathering

feedback from intended users. Clear explanations of supported classes and confidence would help prevent a preliminary model output from being interpreted as a confirmed diagnosis. The current system is therefore best used as an educational demonstration and a starting point for more rigorous field testing.

An evaluation plan for a later version can be divided into three stages. First, create an external test set with images collected from plants and sites not used during training, and document crop, device, lighting, and image-quality conditions. Second, evaluate classification and input rejection separately: report per-class precision, recall, F1, confusion counts, and the rate at which valid leaves or unrelated images are rejected. Third, ask agricultural specialists to review a sample of predictions and record whether the displayed explanation is understandable and appropriately cautious. The separation of these stages helps distinguish model accuracy from interface and screening behavior.

The next model iteration should focus on the observed early-blight recall gap. Additional verified examples could be collected for that class under varied conditions. Training alternatives such as fine-tuning selected backbone layers, class weighting, and stronger augmentation could then be compared using a fixed validation protocol. Each change should be evaluated against the same untouched test set once one is available; repeatedly choosing among models on the final test set would make the reported score optimistic.

If the project adds more crops or disease categories, the interface and data pipeline should be updated together. Class names, disease descriptions, and model outputs need to stay synchronized, and the application should state which crop and conditions are supported. A future explanation view could highlight image regions that influenced a prediction, but any such visualization would need its own review: an attractive heatmap does not by itself prove that the model used biologically meaningful evidence.

V. CONCLUSION

This paper presented a crop disease classification application built around a frozen MobileNetV2 feature extractor and a small trainable classification head. The system trains on 6,644 images in four tomato leaf-condition classes and provides image upload, prediction, confidence display, and disease information through Streamlit. The saved image-level validation report shows 94.43% accuracy and 94.32% weighted F1-score, with lower recall for early blight than for the other classes. These results support the project as an educational prototype; testing on independent field images is needed before making claims about practical farm use.

REFERENCES

1. S. P. Mohanty, D. P. Hughes, and M. Salathé, “Using deep learning for image-based plant disease detection,” Frontiers in Plant Science, vol. 7, Art. no. 1419, 2016, doi: 10.3389/fpls.2016.01419.
2. J. Feng, W. E. Ong, W. C. Teh, and R. Zhang, “Enhanced crop disease detection with EfficientNet convolutional group-wise transformer,” IEEE Access, vol. 12, pp. 44147–44162, 2024, doi: 10.1109/ACCESS.2024.3379303.
3. H. Li, N. Li, W. Wang, C. Yang, N. Chen, and F. Deng, “Convolution self-guided transformer for diagnosis and recognition of crop disease in different environments,” IEEE Access, vol. 12, pp. 165903–165917, 2024, doi: 10.1109/ACCESS.2024.3495529.
4. W. Chen, L. Zheng, and J. Xiong, “Algorithm for crop disease detection based on channel attention mechanism and lightweight up-sampling operator,” IEEE Access, vol. 12, pp. 109886–109899, 2024, doi: 10.1109/ACCESS.2024.3439410.
5. A. Oad, S. S. Abbas, A. Zafar, B. A. Akram, F. Dong, M. S. H. Talpur, and M. Uddin, “Plant leaf disease detection using ensemble learning and explainable AI,” IEEE Access, vol. 12, pp. 156038–156049, 2024, doi: 10.1109/ACCESS.2024.3484574.
6. R. M. Prakash, M. Vimala, K. Ramalakshmi, M. B. Prakash, A. Krishnamoorthi, and R. S. S. Kumari, “Crop disease detection and classification with transfer learning and hyper-parameters optimized convolutional neural network,” in Proc. 3rd Int. Conf. Intelligent Computing, Instrumentation and Control Technologies (ICICICT), 2022, pp. 1608–1613, doi: 10.1109/ICICICT54557.2022.9917901.
7. B. B. S. H. Vardhan, H. Paluvadi, V. K. S. Srihari, and R. C. B., “Groundnut crop disease detection using learning rate annealing methods,” in Proc. 2024

IEEE North Karnataka Subsection Flagship Int. Conf. (NKCon), 2024, pp. 1–7, doi: 10.1109/NKCon62728.2024.10774913.

8. D. Abhiram, K. Anuradha, V. Srilakshmi, and K. Adilakshmi, "A novel two-stage deep learning framework for plant disease detection," in Proc. 5th IEEE Global Conf. for Advancement in Technology (GCAT), 2024, doi: 10.1109/GCAT62922.2024.10924005.
9. H. Yu, J. Liu, C. Chen, A. A. Heidari, Q. Zhang, H. Chen, M. Mafarja, and H. Turabieh, "Corn leaf diseases diagnosis based on K-means clustering and deep learning," IEEE Access, vol. 9, pp. 143824–143835, 2021, doi: 10.1109/ACCESS.2021.3120379.
10. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510–4520.
11. S. G. Saniya et al., "Crop disease detection and classification," GitHub repository, 2026. [Online]. Available: https://github.com/sanigs1122-ship-it/crop-disease-detection. Accessed: Oct. 22, 2026.