

Plant Disease Detection and Treatment System for Home Gardeners Using MobileNetV2 Transfer Learning

Mohd Hyder Hussain¹, K. Balakrishna Maruthiram²

¹MCA Student, Department of Computer Science

²Assistant Professor – Guide, Department of Computer Science

University College of Engineering, Science & Technology,

Jawaharlal Nehru Technological University Hyderabad, Hyderabad, India

Abstract - Home gardeners and plant enthusiasts increasingly grow vegetables, fruit trees and ornamental plants on balconies and small plots, but rarely have access to agricultural extension services that can identify a disease at an early, treatable stage. This paper presents the Plant Disease Detection and Treatment System (PDDTS), a Flask-based web application that lets a home gardener upload a photograph of a leaf and receive an instant diagnosis together with a complete, actionable treatment plan. A lightweight convolutional neural network built by transfer learning on MobileNetV2 classifies leaf images of Apple, Potato and Tomato into seven disease/healthy classes, achieving 98.28% validation accuracy and a validation loss of 0.0527 on a held-out set of 1,393 images drawn from a 6,970-image PlantVillage subset. Rather than stopping at a class label, the system links every prediction to a structured PostgreSQL treatment-reference table — accessed through a dictionary-style RealDictCursor — that supplies severity, chemical treatment, organic treatment and prevention guidance, and augments the result with live local weather from the OpenWeatherMap API, since temperature and humidity strongly influence fungal and bacterial disease progression. The application is organised into Detect, Result, History and Dashboard views, giving the gardener a fast diagnosis as well as a running record of past scans and transparent model-performance metrics. Experimental results, confusion-matrix analysis and a walkthrough of the deployed screens demonstrate that a compact, mobile-friendly CNN combined with a well-structured treatment knowledge base can deliver rapid, trustworthy plant-care support for a non-expert, home-gardening audience.

Key Words: *Plant Disease Detection, Deep Learning, MobileNetV2, Transfer Learning, Convolutional Neural Network, Treatment Recommendation System, PostgreSQL, OpenWeatherMap API, Flask, Home Gardening.*

1. INTRODUCTION

Home gardening has expanded rapidly in urban and semi-urban India, with balconies, terraces and small backyard plots increasingly used to grow vegetables, fruit trees and ornamental plants. Unlike commercial farmers, home gardeners rarely have access to agronomists or laboratory diagnostic services, so when a leaf develops spots, discolouration or wilting, the gardener is largely left to guess — searching generic images online, misidentifying the problem, or reaching for the nearest pesticide regardless of whether it suits the actual cause.

Early, correct identification of plant disease matters because common leaf diseases such as early blight, leaf mould and apple scab spread quickly under favourable humidity and temperature and can move to neighbouring plants within days. A delay of even a week between the first symptom and the start of treatment can decide whether a plant recovers or has to be discarded. Convolutional neural networks (CNNs) have shown strong performance on image-based plant disease classification, and lightweight architectures such as MobileNetV2 make it practical to deploy such models in a web context without specialised hardware. A bare classification label, however, is of limited use to a gardener who does not know what it means, how urgent it is, or what to do next — the value of an assistive system lies as much in what happens after the prediction as in the prediction itself.

This work presents PDDTS, a web application that combines a MobileNetV2-based leaf classifier with a structured treatment and prevention knowledge base stored in PostgreSQL, and a live weather widget powered by the OpenWeatherMap API [10] that gives the gardener environmental context relevant to disease risk and treatment timing. The system is deliberately scoped for home gardeners and plant enthusiasts rather than commercial agriculture: it does not attempt field-scale crop monitoring or farm-level advisory, and instead focuses on a fast, trustworthy answer for a single plant from a single leaf photograph.

The specific objectives of this work are threefold: (i) prepare a labelled image set covering common diseases of three garden-relevant species — Apple, Potato and Tomato — and train a lightweight, transfer-learning CNN to classify a leaf image into one of seven disease/healthy classes; (ii) build a structured treatment-and-history data layer in PostgreSQL that ties every predicted class to severity, chemical and organic treatment, and prevention steps, and logs every scan a user performs; and (iii) assemble these components into a Flask web application with Detect, Result, History

and Dashboard views, so that a gardener receives a diagnosis, a confidence score, a severity rating and next-step guidance in real time, with the surrounding weather conditions shown alongside the result.

In summary, this paper makes three contributions. First, it trains and validates a MobileNetV2-based transfer-learning classifier on a deliberately small, imbalanced, home-garden-relevant subset of PlantVillage, and shows that balanced class weighting recovers strong per-class performance even for the most under-represented class. Second, it designs a structured, queryable treatment-reference schema in PostgreSQL that turns every prediction into severity, chemical, organic and preventive guidance rather than a bare label, addressing a gap that recurs across the literature reviewed in Section 2. Third, it integrates live weather context from OpenWeatherMap into the diagnostic result and packages the whole pipeline into a four-screen Flask application — Detect, Result, History and Dashboard — whose testing and walkthrough (Section 5) confirm that the system behaves as intended end to end, not only at the level of classification accuracy.

2. LITERATURE SURVEY

Mohanty et al. [1] introduced deep CNN-based classification on the PlantVillage dataset (54,306 images, 14 crop species, 26 diseases), achieving over 99% accuracy on held-out lab-condition images, but the study reports only a disease label with no treatment, severity or prevention information and relies on large architectures such as AlexNet and GoogLeNet that are not designed for lightweight deployment.

Ferentinos [2] trained CNN and VGG-based models on 87,848 leaf images across many plant and disease classes, reaching 99.53% test accuracy, and showed that accuracy degrades noticeably when lab-trained models are evaluated on field-condition images — underscoring the domain-shift problem — but again stopped at classification accuracy without any downstream advisory component.

Too et al. [3] compared fine-tuned VGG16, Inception-V4, ResNet variants and DenseNet-121 on PlantVillage, finding that DenseNet-121 improved steadily with more epochs and used comparatively fewer parameters than the alternatives, though the comparison did not include mobile-oriented backbones such as MobileNet and stopped at classification metrics.

Sandler et al. [4] introduced MobileNetV2 itself, built around inverted residual blocks with linear bottlenecks, delivering strong ImageNet accuracy at a fraction of the parameter count of VGG or ResNet — precisely the efficiency profile required for a responsive, low-latency web classifier, and the backbone adopted in this work. Hughes and Salathé [5] released the open, crowd-sourced PlantVillage repository that underlies most plant-disease CNN research, including the Apple/Potato/Tomato subset used here, though the repository supplies only labelled images with no treatment or severity content. Kamilaris and Prenafeta-Boldú [6], in a broad survey of deep learning in agriculture, observed that most model outputs are not carried through into systems that farmers or end users can act on — the same gap this project targets. Rangarajan et al. [7] fine-tuned AlexNet and VGG16 on a tomato-only PlantVillage subset and showed that a well-tuned pre-trained network reaches high accuracy even with modest species-specific data, reinforcing the case for transfer learning on the smaller, species-focused dataset used in this project, though the study is restricted to a single species and, like the others, reports accuracy alone.

Beyond plant pathology, CNN-based detection and classification pipelines have proven effective across a wide range of vision domains, which further supports the choice of an efficient, transfer-learning backbone for a responsive web application. Maruthiram and Keerthana [12] used multi-intensity image fusion to improve deep-learning-based human face detection, while Maruthiram and Reddy [13]–[15] developed and progressively refined CNN and YOLO-based object detection pipelines for autonomous vehicles operating under adverse weather conditions, and Maruthiram and Reddy [16] surveyed object detection and localisation methods in image processing more broadly. Read together with the plant-disease literature, this body of work confirms that well-designed CNN pipelines generalise across very different visual domains provided the surrounding system — pre-processing, class-imbalance handling, and, in this case, a treatment-and-context layer — is designed for the target use case, which is exactly the gap addressed by PDDTS for the home-gardening audience.

3. SYSTEM ANALYSIS

3.1 Existing Approaches and Their Limitations

Home gardeners today mostly rely on manual, search-based identification — comparing symptoms against images found through web search, gardening forums or printed references — which is slow and prone to misdiagnosis, since diseases such as early blight and leaf mould can look superficially similar to an untrained eye. Generic plant-identification mobile apps cover thousands of species but their disease detection is typically shallow, offering a label with no severity rating and no distinction between chemical and organic remedies. Academic classifiers (Section 2) demonstrate that CNN-

based diagnosis is technically feasible but are rarely packaged into a deployable, user-facing application, and even when they are, they stop at the label without addressing what the user should do next. Across all three approaches, three gaps recur consistently: label-only output with no severity, treatment or prevention guidance; no linkage between diagnosis and local environmental conditions, even though humidity and temperature strongly affect fungal and bacterial disease progression; and a design orientation toward commercial-scale farming rather than the few-plant, balcony-or-backyard setting typical of a home gardener.

3.2 Feasibility

The proposed system is technically feasible: it is built entirely on mature, open-source components — Python, TensorFlow/Keras [8], Flask [11] and PostgreSQL [9] — and MobileNetV2's roughly 2.6 million final-model parameters mean training is practical on a single Colab T4 GPU while inference runs comfortably on CPU-only hosting. It is economically feasible because every core component is free and open source and the only recurring cost is OpenWeatherMap API usage, which offers a free tier sufficient for individual or small-scale deployment. Operationally, the system requires no training to use: a photograph, a few seconds of processing, and a tabbed Result screen are enough for a non-technical gardener, while the optional Dashboard view serves the more curious or technical user without being necessary for typical use.

Table -1: Existing Systems Versus the Proposed System

Aspect	Existing Systems	Proposed PDDTS
Output	Disease label only	Label + severity + treatment
Treatment data	Absent / static text	Structured PostgreSQL table
Environmental context	None	Live OpenWeatherMap widget
Target audience	Commercial farms	Home gardeners
Scan history	Not tracked	Logged and filterable

4. PROPOSED SYSTEM AND METHODOLOGY

The proposed system is built around three cooperating components: a MobileNetV2-based leaf classifier, a PostgreSQL-backed treatment and history data layer, and a live OpenWeatherMap widget, all served by a Flask web application designed for home gardeners rather than commercial farm operations.

4.1 System Architecture

A leaf photograph uploaded by the gardener is resized to 224×224 and normalised before being passed to the frozen MobileNetV2 backbone with a lightweight custom classification head, which predicts the disease class and confidence score. The predicted class is used as a key into the treatment_reference table in PostgreSQL to retrieve severity, chemical treatment, organic treatment and prevention steps, while the Flask backend concurrently queries the OpenWeatherMap API for the current temperature, humidity, wind speed and conditions at the user's location. Every prediction, together with its treatment context, is written to a prediction_history table before the combined result is rendered on the Result screen. Fig -1 summarises this pipeline.

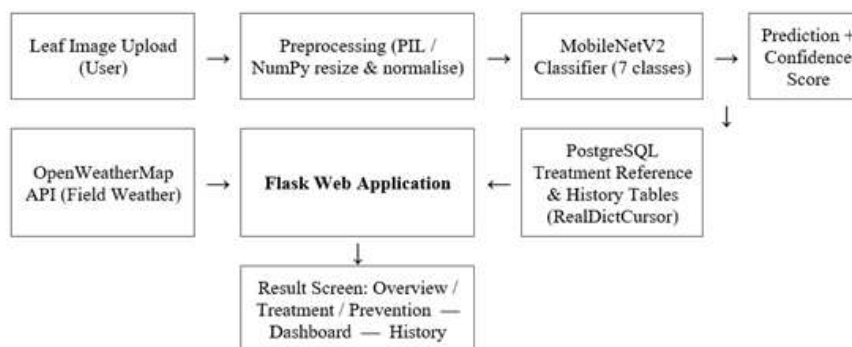


Fig -1: Architecture of the Plant Disease Detection and Treatment System

4.2 Dataset Description

The model is trained on a filtered, seven-class subset of the publicly available PlantVillage dataset covering Apple, Potato and Tomato leaves in healthy and diseased states, totalling 6,970 images (Table -2). The class distribution is

imbalanced — Potato__healthy contributes only 152 images against 1,645 for Apple__healthy — which was corrected during training using balanced class weighting rather than by discarding data. An approximately 80/20 stratified split (fixed seed = 42) produced 5,577 training and 1,393 validation images. Training images were augmented with rotation ($\pm 30^\circ$), width/height shifts ($\pm 20\%$), shear ($\pm 20\%$), zoom ($\pm 20\%$), horizontal flipping and brightness variation ($0.8-1.2\times$); validation images were rescaled only, to give an unbiased performance estimate.

Table -2: Class-Wise Image Count (N = 6,970)

Class	Plant	Status	Images
Apple__Apple_scab	Apple	Disease	630
Apple__healthy	Apple	Healthy	1,645
Potato__Early_blight	Potato	Disease	1,000
Potato__healthy	Potato	Healthy	152
Tomato__Early_blight	Tomato	Disease	1,000
Tomato__Leaf_Mold	Tomato	Disease	952
Tomato__healthy	Tomato	Healthy	1,591

4.3 Model Architecture

A frozen MobileNetV2 base (ImageNet weights, top layer excluded, input $224\times 224\times 3$) is followed by a custom classification head consisting of global average pooling, batch normalisation, two dense layers (256 and 128 units, ReLU) each followed by dropout (0.4 and 0.3), and a final 7-way softmax layer (Table -3). Of 2,624,839 total parameters, only 364,295 in the custom head are trainable, keeping training fast and limiting overfitting risk on a comparatively small dataset.

Table -3: Model Architecture Summary

Layer	Output Shape	Params
MobileNetV2 base (frozen)	(7,7,1280)	2,257,984
GlobalAveragePooling2D	(1280)	0
BatchNormalization	(1280)	5,120
Dense (256, ReLU) + Dropout 0.4	(256)	327,936
Dense (128, ReLU) + Dropout 0.3	(128)	32,896
Dense (7, Softmax)	(7)	903

4.4 Training Configuration

The model was compiled with the Adam optimiser (initial learning rate 0.001) against categorical cross-entropy loss and trained for up to 15 epochs under EarlyStopping (patience = 4, monitoring validation loss, restoring best weights), complemented by ReduceLROnPlateau (factor 0.3, patience 2) and a ModelCheckpoint saving the best model by validation accuracy. Balanced class weights (e.g., 6.53 for Potato__healthy versus 0.61 for Apple__healthy) were applied throughout to offset the class imbalance in Table -2. Training used a batch size of 32 and ran on a Google Colab T4 GPU using TensorFlow 2.20 / Keras.

4.5 Treatment, History and Weather Integration

No treatment text is hard-coded into the application. Each of the seven output classes maps to one row of a treatment_reference table (class label, plant, disease name, severity, chemical treatment, organic treatment, prevention steps), retrieved through a dictionary-style RealDictCursor so that Flask can render it directly into Jinja2 templates and JSON responses. Every scan is logged to a prediction_history table (predicted class, confidence, severity, filename, timestamp), which drives the History screen's filters and the Dashboard's live usage statistics. In parallel, a “Field Weather” widget on the Detect page calls the OpenWeatherMap REST API for the current temperature, “feels like” temperature, humidity, wind speed and short condition description, giving the gardener environmental context — for example, that high humidity currently favours further spread of a fungal infection — alongside the diagnosis.

4.6 Error Handling and Edge Cases

A system built around a live model, a database and a third-party API needs defined behaviour for the unhappy path, not only the happy one. The /predict route rejects unsupported file types and uploads over 10 MB with a clear message

before the file ever reaches the model, so a malformed upload never triggers an unnecessary inference or database write. The weather fetch is treated as an unreliable external dependency: if the OpenWeatherMap call fails, times out or cannot resolve a location, the handler catches the exception and returns generic placeholder content rather than surfacing the error, since a gardener cares about the diagnosis, not the failure mode of a secondary widget. A predicted class with no matching treatment_reference row — which should not occur in normal operation but could arise from a future data-entry error — falls back to a generic “consult a local expert” message instead of crashing or rendering an empty tab. Finally, low-confidence predictions are never disguised as certain: the Result page always shows the raw confidence percentage next to the diagnosis, so a 52.9% result is visibly less trustworthy than a 99.8% one and the gardener can decide whether to retake the photograph before acting on it.

5. EXPERIMENTAL RESULTS

5.1 Training and Validation Performance

Under the EarlyStopping/ReduceLROnPlateau schedule, training ran the full 15 epochs, with training accuracy rising from 80.6% to 96.16% and validation accuracy rising from 92.3% to a best of 98.28%, while validation loss fell from 0.2134 to 0.0527 (Table -4, Fig -2). Validation accuracy sits at or above training accuracy through the later epochs rather than falling behind it — the opposite of an overfitting signature — which is consistent with freezing the MobileNetV2 base and training only a small head on a dataset of this size.

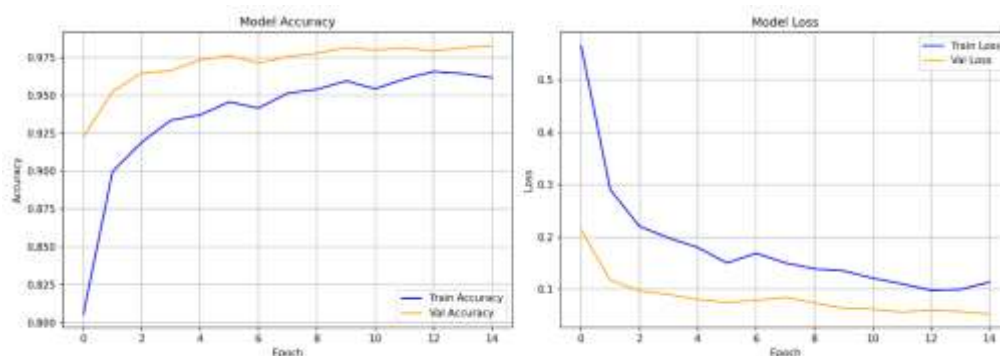


Fig -2: Training/Validation Accuracy and Loss Curves Over 15 Epochs

Table -4: Overall Model Performance Summary

Metric	Value
Best Validation Accuracy	98.28% (epoch 15)
Best Validation Loss	0.0527 (epoch 15)
Final Training Accuracy	96.16%
Training / Validation Images	5,577 / 1,393
Epochs Run	15 (EarlyStopping, patience = 4)

5.2 Per-Class Classification Performance

Table -5 reports precision, recall and F1-score on the full 1,393-image validation set. Every class reaches an F1-score of 0.96 or higher despite the dataset being both small and imbalanced. Potato_healthy, the most under-represented class with only 152 images in the dataset (about 122 after the 80/20 training split), still achieves a 0.98 F1-score — direct evidence that balanced class weighting worked as intended rather than being a purely theoretical safeguard. Tomato_Early_blight shows the lowest recall (0.94) in the table.

Table -5: Classification Performance Report (Per-Class)

Class	Prec.	Recall	F1
Apple__Apple_scab	0.98	0.98	0.98
Apple__healthy	0.99	0.99	0.99
Potato__Early_blight	0.99	0.99	0.99
Potato__healthy	1.00	0.97	0.98
Tomato__Early_blight	0.98	0.94	0.96

Tomato__Leaf_Mold	0.96	0.98	0.97
Tomato__healthy	0.98	1.00	0.99
Macro / Weighted Avg.	0.98	0.98	0.98

5.3 Confusion Matrix Analysis

The 7×7 confusion matrix over the full validation set (Fig -3) is overwhelmingly diagonal: 1,369 of 1,393 images (98.3%) are classified correctly, consistent with Table -4. The small number of off-diagonal cells identify the specific class pairs — chiefly among the visually similar tomato disease classes — that the model finds hardest to separate, which is where future data collection or fine-grained features would have the most impact.

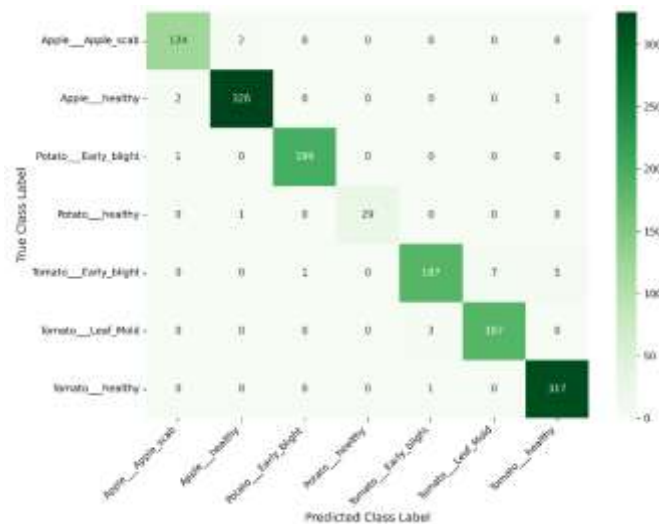


Fig -3: Confusion Matrix on the 1,393-Image Validation Set

5.4 System Testing

Unit tests verified image pre-processing for varied sizes, aspect ratios and colour modes; that exactly one treatment_reference row is returned per class, including correct maintenance guidance for the healthy classes; that the weather-fetch function degrades gracefully on an invalid location or network failure; and that History filtering by plant and severity returns consistent results. Integration testing exercised the full Detect → classify → treatment-lookup → weather → history-log → Result pipeline on held-out images from every class, and cross-checked the History and Dashboard “live” statistics directly against the database. A representative subset of the thirteen system test cases is shown in Table -6; all cases passed on the final build.

Table -6: Representative System Test Cases

ID	Description	Result
TC01	Upload a valid JPG leaf image	Pass
TC03	Upload an oversized image (>10 MB)	Pass
TC06	Treatment lookup for a disease class	Pass
TC09	Weather API unreachable / timeout	Pass
TC13	Dashboard live statistics refresh	Pass

5.5 Application Walkthrough

The deployed application exposes four views. The Detect page (Fig -4) accepts a drag-and-dropped leaf image, states the seven supported classes and the model's headline validation accuracy up front, and shows the live Field Weather widget alongside the upload panel.

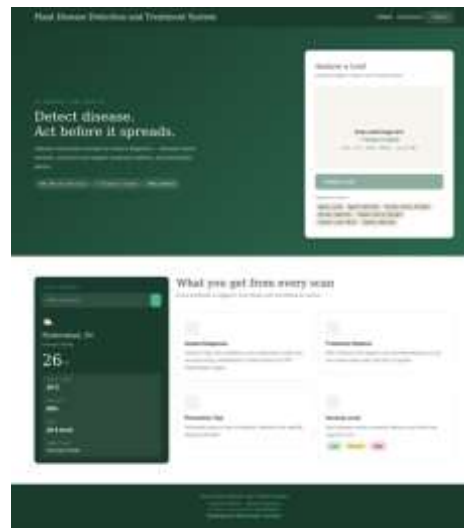


Fig -4: Detect Screen with the Field Weather Widget

The Result page (Fig -5) presents Overview, Treatment and Prevention tabs for the predicted class, always alongside the raw confidence percentage so a gardener can judge how much to trust a borderline result; a healthy prediction surfaces plain maintenance guidance rather than an empty treatment tab.

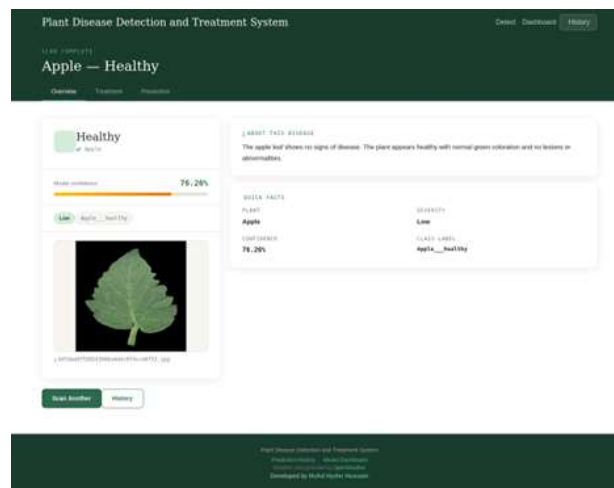


Fig -5: Result Screen — Overview Tab for a Healthy Prediction

The History page (Fig -6) lists past scans in reverse-chronological order with summary cards (total scans, diseases found, average confidence, most-scanned plant) and drop-down filters for plant species and severity, letting a gardener track whether a given plant is improving over repeated visits.

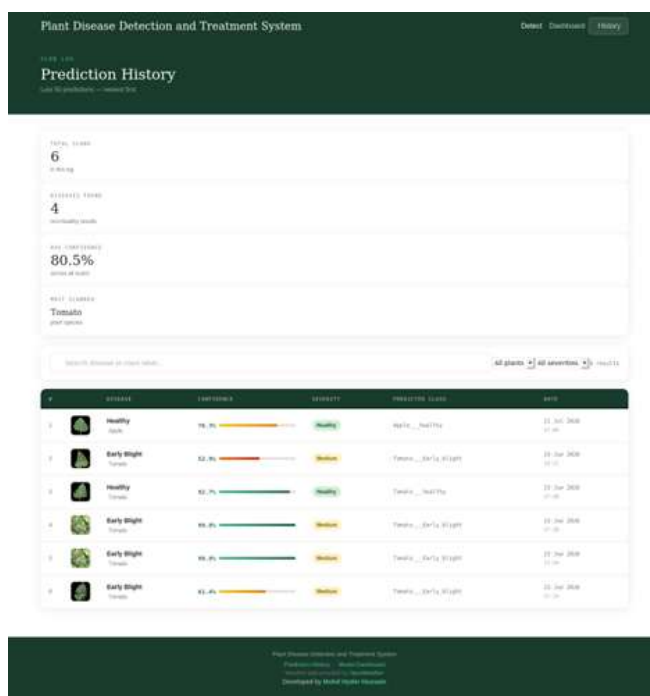


Fig -6: History Screen — Prediction Log with Filters

The Dashboard (Fig -7) exposes the same validation accuracy and loss figures as Table -4, together with the training/validation curves and confusion matrix, for a curious or technical user who wants to look past a single diagnosis and inspect the model itself — without this screen being necessary for the everyday Detect → Result workflow.

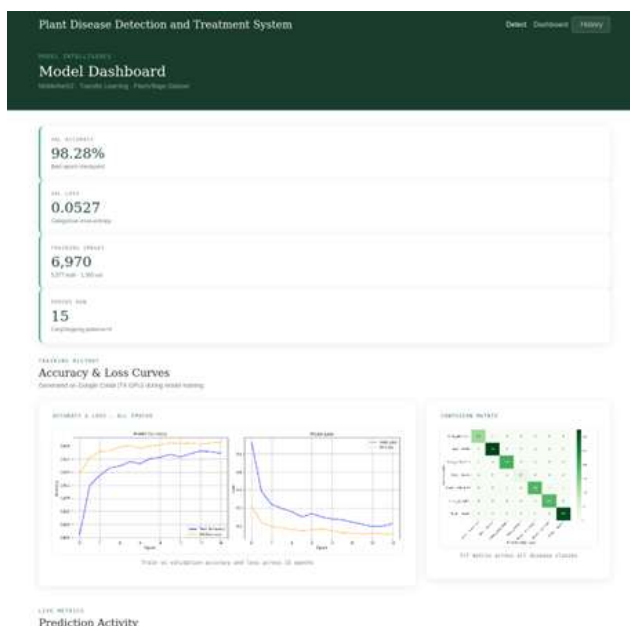


Fig -7: Dashboard Screen — Model Metrics and Training Curves

5.6 Discussion

Table -7 situates the achieved validation accuracy against the plant-disease literature reviewed in Section 2. The reported figures are not directly comparable class-for-class, since dataset size, species coverage and lab- versus field-condition images differ across studies; the comparison is included to show that a MobileNetV2-based head trained on a much smaller, imbalanced, three-species subset (6,970 images versus 54,306 or 87,848 in [1] and [2]) still reaches accuracy in the same band as larger backbones trained on far larger datasets, while using a fraction of the parameters and, unlike any of the reviewed systems, carrying every prediction through to a structured, actionable treatment plan.

Table -7: Validation Accuracy Across Related Studies

Study	Backbone	Val. Acc.
Mohanty et al. [1]	AlexNet / GoogLeNet	>99%
Ferentinos [2]	VGG (best)	99.53%
Rangarajan et al. [7]	VGG16 / AlexNet	97.29%
This work	MobileNetV2 (transfer)	98.28%

6. CONCLUSION

This paper presented PDDTS, a treatment-first plant disease detection system aimed squarely at home gardeners and plant enthusiasts, a user group that existing plant-disease research largely overlooks in favour of commercial-scale agriculture and bare classification labels. A MobileNetV2-based transfer-learning classifier, trained on a small, imbalanced seven-class subset of PlantVillage with balanced class weighting, reached 98.28% validation accuracy and a per-class F1-score of at least 0.96 while remaining light enough (2.6 million parameters) for interactive web inference without a dedicated GPU. More importantly, every prediction is carried through to a structured PostgreSQL treatment-reference table — covering severity, chemical treatment, organic treatment and prevention — and paired with live OpenWeatherMap data, so that a diagnosis always arrives with an actionable next step and relevant environmental context, delivered through Detect, Result, History and Dashboard views in a Flask web application. Confusion-matrix analysis and an end-to-end walkthrough of the deployed screens support the conclusion that a compact CNN combined with a well-structured treatment knowledge base can give home gardeners fast, trustworthy, near-actionable plant-care support.

Limitations: Two limitations are worth stating plainly. First, like most PlantVillage-derived work, the training images are largely lab-condition photographs against clean backgrounds, and Section 2 notes that field-condition accuracy typically drops relative to lab-condition accuracy [2]; performance on genuinely cluttered home-garden photographs (uneven lighting, soil or pot in frame, partially occluded leaves) has not been separately measured. Second, the species and disease coverage is deliberately narrow — three plants and seven classes — so a gardener growing a species outside this set receives no diagnosis at all rather than a degraded one. Both limitations are addressed directly in the future work below.

7. FUTURE SCOPE

Planned extensions include broadening species and disease coverage beyond Apple, Potato and Tomato, fine-tuning deeper layers of the MobileNetV2 backbone once more field-condition (rather than lab-condition) images are available, and adding on-device or edge inference for offline use. Given the accuracy drop typically observed when lab-trained plant-disease models are evaluated on field images [2], collecting real home-garden photographs — including under adverse lighting and background clutter — for fine-tuning is a priority. Multilingual treatment content, push notifications tied to weather-driven disease risk, and a mobile companion app are also planned to widen the system's reach among home-gardening communities.

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. 1419, 2016.
- [2] K. P. Ferentinos, “Deep Learning Models for Plant Disease Detection and Diagnosis,” *Computers and Electronics in Agriculture*, vol. 145, pp. 311–318, 2018.
- [3] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, “A Comparative Study of Fine-Tuning Deep Learning Models for Plant Disease Identification,” *Computers and Electronics in Agriculture*, vol. 161, pp. 272–279, 2019.
- [4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in *Proc. IEEE CVPR*, pp. 4510–4520, 2018.
- [5] D. P. Hughes and M. Salathé, “An Open Access Repository of Images on Plant Health to Enable the Development of Mobile Disease Diagnostics,” *arXiv preprint arXiv:1511.08060*, 2015.
- [6] A. Kamilaris and F. X. Prenafeta-Boldú, “Deep Learning in Agriculture: A Survey,” *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.
- [7] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, “Tomato Crop Disease Classification Using Pre-Trained Deep Learning Algorithm,” *Procedia Computer Science*, vol. 133, pp. 1040–1047, 2018.
- [8] F. Chollet et al., “Keras,” <https://keras.io>, 2015.

- [9] PostgreSQL Global Development Group, “PostgreSQL Documentation,” <https://www.postgresql.org/docs/>.
- [10] OpenWeather, “OpenWeatherMap Current Weather Data API Documentation,” <https://openweathermap.org/current>.
- [11] Pallets Projects, “Flask Documentation,” <https://flask.palletsprojects.com/>.
- [12] K. B. Maruthiram and V. Keerthana, “Optimizing Human Face Detection With Multi-Intensity Image Fusion in Deep Learning,” *International Journal of All Research Education and Scientific Methods (IJARESM)*, vol. 12, no. 8, pp. 1671–1678, 2024.
- [13] K. B. Maruthiram and G. V. R. Reddy, “Improving Object Detection and Classification for Autonomous Vehicle in Adverse Weather Conditions,” in *Proc. 2024 2nd International Conference on Artificial Intelligence Trends and Pattern Recognition (ICAITPR)*, 2024, pp. 1–6.
- [14] K. B. Maruthiram, G. V. R. Reddy, and S. Supraja, “Object Detection of Autonomous Vehicles in Adverse Weather Conditions,” *GIS Science Journal*, vol. 12, no. 7, pp. 557–566, 2025.
- [15] K. B. Maruthiram, G. V. R. Reddy, and B. Pradeep, “Object Detection in Autonomous Vehicles in Adverse Weather Conditions Using YOLO,” *Journal of Systems Engineering and Electronics*, vol. 35, no. 7, pp. 193–205, 2025.
- [16] K. B. Maruthiram and G. V. R. Reddy, “A Survey Paper on Object Detection and Localization Methods in Image Processing,” *International Journal of Creative Research Thoughts*, vol. 13, no. 6, 2024.