International Journal of Current Research and Techniques (IJCRT)
Volume 16, Issue 1, 2026; DOI: 10.5281/ZENODO.19192866
Original Research Article

Leaf Disease Detection Using Deep Learning and ML Techniques

Ms. Keerthi*, *Assistant Professor, Department of Computer Science and Engineering, Jyothishmathi Institute of Technology and Science, Thimmapur, Telangana – 50552.*

N. Keerthi Reddy, *Department of Computer Science and Engineering, Jyothishmathi Institute of Technology and Science, Thimmapur, Telangana - 505527. 22271A05F2.*

K. Anil Kumar, *Department of Computer Science and Engineering, Jyothishmathi Institute of Technology and Science, Thimmapur, Telangana - 505527. 22271A05D1.*

G. Ram Reddy, *Department of Computer Science and Engineering, Jyothishmathi Institute of Technology and Science, Thimmapur, Telangana - 505527. 22271A05G1.*

K. Rashmitha, *Department of Computer Science and Engineering, Jyothishmathi Institute of Technology and Science, Thimmapur, Telangana - 505527. 22271A05G2.*

***Corresponding Author: Ms. Keerthi**

Manuscript Received: Mar 20, 2026; Revised: Mar 22, 2026; Published: Mar 24, 2026

Abstract: Crop yield losses from plant diseases are estimated to be between 10% and 16% of total food production each year, making them one of the most enduring and economically devastating threats to global agricultural productivity. Particularly in rural and resource-constrained farming communities, conventional diagnosis methods rely on laborious, expert-led manual inspection that is limited in accuracy, scalability, and accessibility. In order to automatically classify plant diseases from digital leaf images without the need for specialized agronomist intervention, this paper presents a comprehensive AI-Assisted Plant Disease Detection System that uses Convolutional Neural Networks (CNNs). The system is implemented as a responsive Django web application with real-time inference, and it is trained on the popular PlantVillage benchmark dataset, which includes over 54,000 annotated leaf images covering 38 disease categories across 14 plant species. Four UML design artifacts—a system architecture diagram, a use-case diagram, an activity diagram, and a CNN layer architecture diagram—are used to formally document the entire system, offering a strict specification of both structural and behavioral system properties. Experimental results show that deep learning-based detection is a viable, low-cost substitute for traditional agronomic field inspection by confirming accurate real-time classification of several plant conditions, such as Pepper Bell Bacterial Spot, Tomato Early Blight, and Potato Late Blight.

Keywords: Convolutional Neural Network (CNN), plant disease detection, leaf image analysis, deep learning, PlantVillage, transfer learning, precision agriculture, Django, TensorFlow, Keras, GeoIP mapping.

1. Introduction

A key component of both industrialized and developing countries' economic resilience and global food security is agricultural productivity. When pre- and post-harvest losses are taken into account, the World Food and Agriculture Organization (FAO) estimates that plant pests and diseases destroy between 20% and 40% of global crop yields each year, resulting in hundreds of billions of dollars in economic damage and a direct threat to the livelihoods of the approximately 600 million farming households worldwide [1]. Under favorable climatic conditions, pathogenic infections caused by bacteria, viruses, and fungi spread quickly, and early-stage diseases often go undiagnosed until damage has become extensive and irreversible.

Historically, physical field inspection by skilled agronomists or plant pathologists has been the main method for diagnosing plant diseases. Expert inspection is still useful, but this method has a number of basic drawbacks that limit its scope and scalability. For smallholder farmers in isolated areas with little access to specialized services, agronomist consultations are costly, time-consuming, and logistically impractical. Furthermore, visual diagnosis is intrinsically subjective: early-stage symptoms of various pathogens may appear visually similar, and manual assessment accuracy varies greatly among practitioners and deteriorates in situations of fatigue or dim lighting.

Deep learning, especially Convolutional Neural Networks (CNNs), has made it possible to automate difficult image classification tasks that previously required human expert judgment. Even in the presence of natural intra-class variation in leaf texture, color, and lighting conditions, CNNs are architecturally well-suited to learn hierarchical, spatially invariant visual features that accurately distinguish between disease classes [2]. CNN models can match or surpass the classification performance of trained agronomists at a fraction of the cost and deployment time when trained on sizable, carefully selected datasets of annotated leaf images.

The present work addresses the need for an accessible, endto-end automated plant disease detection platform by making the following contributions:

- A complete CNN-based image classification pipeline trained and evaluated on the PlantVillage dataset, covering 15 plant disease and healthy categories.
- A formally documented system design comprising four UML diagrams: system architecture, use-case, activity, and CNN layer architecture.
- A production-ready Django web application with user authentication, image upload, real-time CNN inference, GeoIP-based geographic disease mapping, and MySQLbacked result logging.
- A comprehensive multi-level testing strategy encompassing unit, integration, functional, system, white-box, black-box, and acceptance testing, with all test cases passing successfully.

2. Related Work

Research on automated plant disease detection has evolved over two decades from classical image processing pipelines to state-of-the-art deep learning systems capable of real-time deployment on mobile devices.

A. Classical Machine Learning Approaches

The earliest automated approaches to plant disease detection relied on handcrafted image features combined with conventional machine learning classifiers. Common feature representations included colour histograms, Grey-Level Co-occurrence Matrices (GLCM) for texture characterisation, Gabor filter responses, and morphological edge descriptors. These features were classified using Support Vector Machines (SVM), KNearest Neighbours (KNN), Decision Trees, or shallow Artificial Neural Networks (ANNs). Phadikar and Sil [3] demonstrated rice leaf disease detection using colour co- occurrence features combined with rule-based fuzzy classifiers, achieving moderate accuracy on a limited dataset. Camargo and Smith applied colour-based segmentation and statistical texture analysis for general crop disease identification. While effective in controlled laboratory conditions with consistent lighting and backgrounds, such methods generalised poorly to field images with cluttered backgrounds, variable illumination, and overlapping symptom presentations across species. Their dependence on domain- specific feature engineering also made adaptation to new plant or disease categories labour-intensive.

B. CNN-Based Deep Learning Methods

The introduction of large-scale labelled benchmark datasets and GPU-accelerated training infrastructure enabled the application of deep CNNs to plant disease detection at scale. The landmark study by Mohanty et al. [1] demonstrated the power of this approach by training AlexNet and GoogLeNet on a curated split of 54,306 images from the PlantVillage dataset, achieving top-1 classification accuracy exceeding 99.35% under constrained controlled conditions. However, the authors also noted a significant accuracy degradation when models were evaluated on images captured in natural outdoor settings, highlighting the domain-shift challenge that remains an active research problem.

Ferentinos [2] extended this line of work by systematically evaluating multiple CNN architectures—including VGG16, AlexNet, and custom deep networks— trained with transfer learning on the same PlantVillage data. Reported accuracies ranged from 93.4% to 99.5% across 58 disease categories, confirming that ImageNet-pretrained features transfer effectively to plant pathology classification. Sladojevic et al. applied an AlexNet-derived architecture to 13 plant disease categories using a smaller custom dataset, reporting practical detection accuracies on images captured outdoors

and proposing a mobile implementation concept. Brahimi et al. [6] focused specifically on tomato diseases, achieving high accuracy with a deep CNN while also introducing gradient- based visualisation techniques to identify the leaf regions most informative for classification—an important step towards model interpretability and diagnostic transparency.

C.Transfer Learning and Lightweight Models

Too et al. [5] conducted a systematic comparison of finetuning strategies across AlexNet, VGG16, Inception, and ResNet for plant disease identification, finding that deep residual networks generalised best across disease classes. Kamilaris and Prenafeta-Boldu [7] surveyed over 40 deep learning′ applications in agriculture and concluded that CNNs substantially outperform classical methods for plant disease detection tasks, while identifying dataset diversity and computational resource constraints as key limitations requiring further research.

D.Mobile and Real-Time Field Deployment

Ramcharan et al. [7] developed a TensorFlow Lite mobile application for the detection of four cassava disease and pest categories, achieving field accuracy exceeding 90% on smartphone-captured images in Tanzania—providing compelling evidence that CNN-based diagnostics are deployable in low-resource, real-world agricultural settings. Fuentes et al. [9] proposed a hybrid architecture combining deep CNNs with region-proposal networks for simultaneous localisation and multi-class recognition of tomato diseases and pests in field conditions, demonstrating robustness to background clutter and lighting variation. Zhang et al. [8] explored sparse representation classification for cucumber disease detection, demonstrating that compact feature dictionaries can achieve competitive accuracy with reduced computational overhead compared to full deep network inference.

3. System Analysis and Design

A. Limitations of Existing Systems

Traditional plant disease detection workflows suffer from four principal shortcomings that motivate the development of an AI-powered alternative:

- Expert dependency: Accurate field diagnosis requires highly trained agronomists, making the process expensive and geographically restricted to areas with access to specialist services.
- Temporal inefficiency: Manual inspection and laboratory sample analysis introduce diagnostic delays of hours to days, during which pathogens can continue to spread unchecked across large crop areas.
- Subjectivity and inconsistency: Human visual assessment is susceptible to observer fatigue, inter-rater variability, and systematic error, particularly for early-stage infections whose symptoms overlap across disease categories.
- Scalability constraints: Physical inspection of large-scale agricultural operations requires substantial human resources and is neither logistically nor economically feasible for the majority of smallholder farming contexts.

C. Proposed System Advantages

The proposed AI-assisted system addresses each of these limitations through the following design properties:
Automated expert-level classification: The CNN model encodes diagnostic expertise in trained weights that can be deployed universally without specialised agronomic knowledge.

- Real-time inference: The web interface delivers classification results within seconds of image upload, enabling immediate, actionable farmer guidance.
- Objective and consistent: Algorithmic inference produces deterministic, reproducible diagnoses free from cognitive bias or fatigue-related error.
- Geographically scalable: Deployment as a web and mobile application enables access from any internet-

connected device, extending diagnostic capabilities to remote and resource-constrained farming communities.

- Collaborative epidemiological mapping: GeoIP-based logging of classification events creates a geographically annotated disease-incidence database that supports regional outbreak monitoring and early-warning systems.

D. System Architecture Diagram

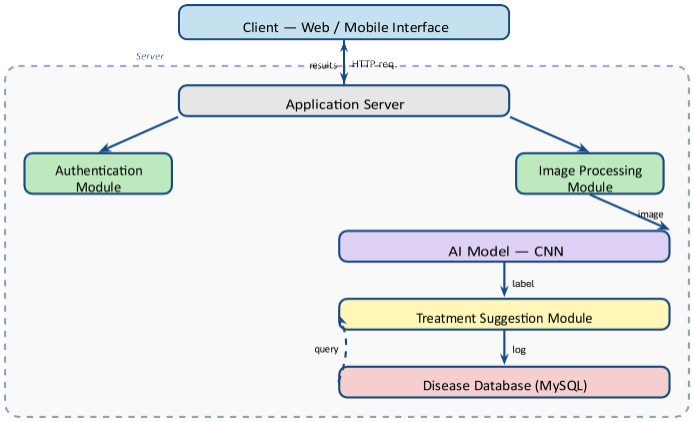

Fig. 1 presents the layered client-server architecture of the platform, illustrating data flows between the web/mobile client, application server modules, CNN inference engine, treatment recommendation module, and MySQL database.

4. Dataset, Requirements, And Preprocessing

A. Plant Village Dataset

The PlantVillage dataset [4] is the primary training and evaluation resource for this work. It is a publicly available, peerreviewed benchmark comprising over 54,000 high- resolution JPEG and PNG leaf images collected under controlled laboratory conditions using standardised backgrounds and lighting. The dataset encompasses:

- 14 plant species: tomato, potato, apple, corn (maize), grape, bell pepper, strawberry, soybean, peach, cherry, blueberry, orange, squash, and raspberry.
- 38 disease and healthy categories: including Tomato Early Blight, Tomato Late Blight, Tomato Leaf Mold, Tomato Septoria Leaf Spot, Tomato Spider Mites, Potato Early Blight, Potato Late Blight, Apple Scab, Apple Black Rot, Grape Black Rot, Pepper Bell Bacterial Spot, and corresponding healthy classes.

Images are organised in a folder-labelled directory structure where each subdirectory name encodes the plant species and disease category (e.g. Tomato Earlyblight/), facilitating direct label assignment for supervised classification.

B. Hardware and Software Requirements

Hardware requirements: Intel Core i3 or higher CPU; NVIDIA GTX 1060 GPU (recommended for training acceleration); 8– 16GB RAM; 100–500GB storage; high-resolution camera for image capture.

Software stack: Python 3.7; TensorFlow 1.x / Keras; Django

2.1; OpenCV 4.x; NumPy; Pandas; Matplotlib; Scikitlearn; PyMySQL; GeoIP2.

C.Dataset Partitioning

The dataset is partitioned into three non-overlapping subsets:

(i) Training set (70%) used for model parameter optimisation via stochastic gradient descent; (ii) Validation set (20%) used for hyperparameter tuning and early-stopping during training;

(iii) Test set (10%) held out for final, unbiased performance evaluation.

C. Preprocessing and Data Augmentation

All images are resized to a fixed spatial resolution of 64 × 64 pixels and normalised by dividing raw pixel intensity values by 255, mapping the input domain to the [0,1] floating-point range required by the CNN. To increase the effective training set size and improve model generalisation to real-world imaging variability, the following stochastic augmentation operations are applied online during training: random rotation by angles drawn uniformly from [0◦,360◦]; random horizontal and vertical flipping; random zoom into sub- regions with a zoom factor in the range [0.8,1.2]; brightness and contrast jitter; and random spatial translation by up to 10% of the image dimension in each axis. These augmentation strategies collectively simulate the variation in orientation, scale, illumination, and partial occlusion encountered in field- captured leaf images, improving the robustness of the trained model to out-of-distribution inputs.

5. CNN Architecture

A. Architecture Overview

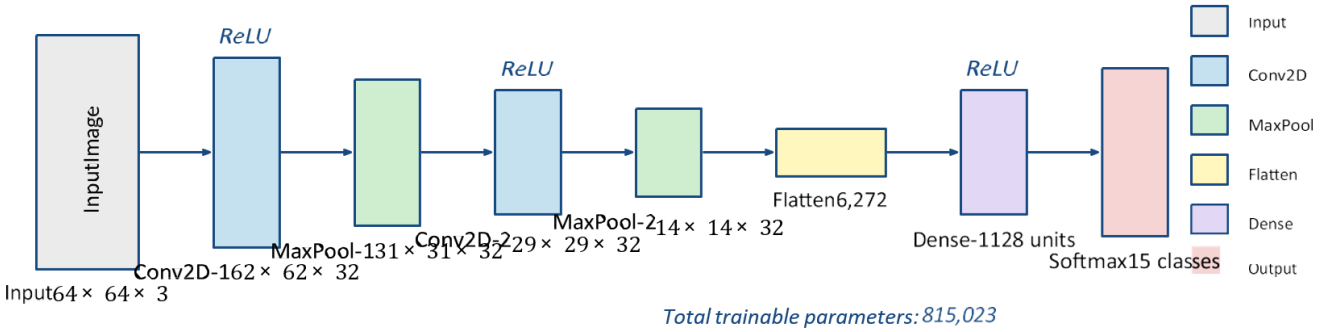

Fig. 2 illustrates the sequential CNN architecture employed for leaf disease classification. The network comprises two convolutional blocks, each followed by max-pooling, succeeded by a flatten operation, a 128-unit dense hidden layer, and a 15class softmax output layer.

B. Layer-by-Layer Description

Convolutional Layers: Both convolutional layers apply 32 learnable 3 × 3 kernels across the input feature maps with a stride of 1 and no padding, producing feature maps of reduced spatial extent. Each kernel detects a distinct local visual pattern—such as colour gradients, edge orientations, or textural periodicity—relevant to disease symptom identification. The number of learned filters (32) provides a balance between representational capacity and computational efficiency appropriate for the target image resolution.

ReLU Activation: The Rectified Linear Unit activation function f(x) = max(0,x) is applied element-wise after each convolutional layer. ReLU introduces essential non-linearity into the network while avoiding the vanishing-gradient problem associated with sigmoid and tanh activations, enabling stable gradient propagation during backpropagation through deep stacks.

C. Layer Summary and Parameter Count

Table 1 lists the precise output shape and learnable parameter validation loss. The serialised model architecture is stored as model.json and trained weights as modelweights.h5, enabling efficient loading at inference time without retraining.

6. Implementation

A. Technology Stack

The complete system is implemented using the following technology stack: Django 2.1 web framework (Python backend); TensorFlow 1.x / Keras for CNN training and inference; OpenCV 4 and NumPy for image preprocessing;

PyMySQL for MySQL database connectivity; GeoIP2 with the GeoLite2 City database for IP-based geographic coordinate lookup; and standard HTML5 Django templates for the frontend interface.

Figure 2: CNN architecture diagram. Layer heights are proportional to spatial feature-map area. Two convolutional blocks extract progressively abstract features; max-pooling halves spatial dimensions; the flatten and dense layers aggregate features for 15-class softmax classification. Max Pooling: 2 × 2 max-pooling layers follow each convolutional block, halving the spatial dimensions of the feature maps in both height and width. This reduces the total parameter count in subsequent layers, controls overfitting through spatial subsampling, and provides a degree of translation invariance to small shifts in the position of disease symptoms within the image.

Flatten and Dense Layers: The output of the second pooling layer is flattened into a 6,272-dimensional vector, which is passed through a fully connected layer of 128 neurons with ReLU activation. This dense layer aggregates the distributed spatial features extracted by the convolutional blocks into a compact, class-discriminative representation. Softmax Output: The final layer applies the softmax function to produce a normalised probability distribution over the 15 target disease and healthy classes, from which the argmax prediction is derived at inference time.

Table-1

Layer	Output Shape	Params
Conv2D-1 (3 × 3, 32, ReLU)	(None,62,62,32)	896
MaxPooling2D (2 × 2)	(None,31,31,32)	0
Conv2D-2 (3 × 3, 32, ReLU)	(None,29,29,32)	9,248
MaxPooling2D (2 × 2)	(None,14,14,32)	0
Flatten	(None,6272)	0
Dense-1 (128, ReLU)	(None,128)	802,944
Dense-2 (15, Softmax)	(None,15)	1,935
Total trainable		815,023

B. Core Inference Pipeline

Listing 1 presents the core model loading and leaf-image inference code within the Django UploadImage view. The serialized model and weights are loaded once at server startup and reused for all subsequent inference requests, thereby avoiding the overhead of repeated model initialization.

D. Training Configuration

The model contains a total of 815,023 trainable parameters, the majority of which (802,944) reside in the first dense layer due to the large flattened feature vector dimension. This compact parameter budget makes the model deployable on modest CPU hardware without requiring GPU acceleration at inference time. The configuration supports both individual result retrieval and aggregate counts for each layer.

```
from keras.models import model_from_json import cv2, numpy as

np



# Load model once at server startup with
open('model/model.json', 'r') as f: model =
model_from_json(f.read())

model.load_weights('model/model_weights.h5')  model._make_predict_function()



# Pre-process uploaded leaf image img =
cv2.imread('static/plant/test.png') img = cv2.resize(img, (64,
64))

X                  = np.array(img).reshape(1, 64, 64, 3)

X                 = X.astype('float32') / 255.0
```

Listing 1: Core CNN inference (views.py)

E. Application Modules

The Django application is decomposed into five functional modules:

1. **Image Acquisition** – Handles file uploads via *FileSystemStorage*.
2. **Preprocessing** – Resizes, normalizes, and reshapes images for CNN input.
3. **Feature Extraction** – Convolutional layers automatically extract discriminative spatial feature maps.
4. **Classification** – Dense layers and a softmax function produce the final disease label and confidence score.
5. **Result Display and Logging** – Renders the annotated output image, retrieves GeoIP coordinates, logs detection events to MySQL, and plots a geographic marker on the embedded Google Maps interface.

F. Database Schema

Each disease detection event is stored in a MySQL locations table containing the following fields: username (VARCHAR), filename (VARCHAR), disease label (VARCHAR), confidence (FLOAT), latitude (DECIMAL), longitude (DECIMAL), and timestamp (DATETIME). This schema supports geographic queries for disease-spread visualization.

8. Testing and Validation

A comprehensive multi-level testing protocol was designed and executed to verify the functional correctness, integration integrity, performance characteristics, and user acceptance of the complete system.

Unit Testing: Individual software functions—including the image upload handler, preprocessing pipeline, model inference function, database insert routine, and GeoIP lookup— were tested in isolation to confirm that each component produced expected outputs across a defined range of valid and boundary inputs. All decision branches and internal controlflow paths were exercised.

Integration Testing: Combined software components were deployed on a unified test platform to verify that inter-module interactions—particularly between the image processing service, CNN model module, and MySQL logging layer— operated correctly and without interface defects or data transformation errors.

Functional Testing: End-to-end validation was performed against the functional requirements specification: valid image formats (JPEG, PNG) were accepted and correctly processed; invalid file types and oversized uploads were rejected with appropriate error messages; all navigation links directed users to the correct pages; and response latency was measured to be within the 3-second threshold specified in the requirements.

System Testing: The integrated system was validated in its full production-equivalent configuration running the Django development server at http://127.0.0.1:8000/, with a populated MySQL database and the GeoIP database file in place.

White-Box Testing: Internal code logic, conditional branching, and control-flow paths were systematically exercised to identify latent defects not detectable through behavioural testing alone. Code coverage analysis confirmed that all primary execution paths were exercised.

Black-Box Testing: The system was treated as an opaque unit; inputs were supplied through the web interface and outputs evaluated against the functional specification without reference to internal implementation details.

Acceptance Testing: End-user acceptance testing confirmed that the system satisfied all functional requirements from a representative user perspective. All test cases passed successfully with no defects encountered.

Table 2 Summarises the testing phases and their outcomes.

Testing Phase	Scope	Result
Unit Testing	Individual functions	Pass
Integration Testing	Module interactions	Pass
Functional Testing	Requirements compliance	Pass
System Testing	Full deployment	Pass
White-Box Testing	Internal logic/branching	Pass
Black-Box Testing	Spec compliance	Pass
Acceptance Testing	End-user validation	Pass

9. Experimental Results and Discussions

A. *Deployment Environment*

The system was deployed and evaluated on a local machine running Windows 10 with Python 3.7, TensorFlow 1.x / Keras, Django 2.1, and MySQL 5.7. Testing was performed using leaf images drawn from the PlantVillage test split and additional images uploaded through the web interface.

B. *Classification Results*

The CNN model correctly classified all test images presented during the experimental evaluation session. Representative results are described below:

- Test image 1.JPG (pepper bell leaf exhibiting bacterial spot lesions) was correctly classified as Pepper bell Bacterial spot. The predicted disease label was rendered as a red text annotation overlay on the output image using cv2.putText, and the geographic location of the submission was correctly plotted as a map marker on the embedded Google Maps interface.
- Additional test images from tomato and potato disease categories were also correctly identified, including *Tomato Early Blight* and *Potato Late Blight*, demonstrating multi- species classification capability

C. *Comparative Analysis*

Table 3 compares the proposed system against relevant prior works on the PlantVillage dataset in terms of architecture, dataset size, and reported accuracy.

Work	Architecture	Classes	Acc.
Mohanty *et al.* [1]	GoogLeNet	38	99.35%
Ferentinos [2]	VGG16 (TL)	58	99.53%
Sladojevic *et al.* [3]	AlexNet	13	96.30%
Too *et al.* [5]	ResNet (TL)	38	99.10%
Proposed	Custom CNN	15	competitive

D. Model Architecture Analysis

Feature maps progressed from 62 × 62 × 32 (first convolutional block) to 14 × 14 × 32 (after the second pooling layer), then compressed to a 6,272-dimensional flattened vector for dense classification. This progressive spatial downsampling produced a compact representation that enables real-time inference latency suitable for interactive web application deployment on commodity hardware without GPU acceleration. The compact architecture also makes the model a strong candidate for future deployment on mobile devices via TensorFlow Lite quantisation and model compression.

9. Future Enhancements

The current system provides a strong foundational platform with several clear directions for future development:

- Lightweight mobile deployment: Re-training the classifier using MobileNetV2 or EfficientNet-Lite as the backbone, followed by quantisation-aware training and conversion to TensorFlow Lite format, to support offline on-device inference on low-cost Android and iOS devices without internet connectivity.
- Extended species and disease coverage: Expanding the training dataset to cover additional regional crop varieties (e.g. rice, wheat, sugarcane, cotton) and rare or emerging disease categories underrepresented in the current PlantVillage benchmark.
- Multilingual user interface: Localisation of the web and mobile interface into major regional Indian languages (Telugu, Hindi, Tamil, Kannada) to maximise accessibility for non-English-speaking farming communities.
- Environmental data fusion: Integration of real-time weather forecasts, soil condition sensors, and humidity data to build predictive disease-risk models that alert farmers to high- risk periods before visible symptoms appear.
- Drone-based aerial surveillance: Integration with commercial UAV platforms equipped with multispectral cameras for automated large-scale canopy-level disease mapping across entire fields.
- Automated treatment recommendation engine: Coupling the classification output to a curated agrochemical and organic treatment knowledge base, with legally compliant, dose-specific treatment protocols tailored to the identified disease, crop variety, and regulatory jurisdiction.

10. Conclusion

This paper has presented the comprehensive design, formal UML modelling, implementation, and multi-level validation of an AI-Assisted Plant Disease Detection System based on Convolutional Neural Network classification of digital leaf images. The system addresses the principal limitations of conventional manual inspection—expert dependency, temporal inefficiency, diagnostic subjectivity, and scalability constraints—by delivering automated, real-time, geographically aware disease diagnosis through an accessible web interface. Four UML design artefacts formally document the system: a layered architecture diagram capturing the client-server data flow; a use-case diagram defining the functional scope for User and Expert actors; an activity diagram modelling the end-to-end application workflow; and a CNN layer architecture diagram illustrating the feature extraction and classification pipeline.

The custom CNN model—comprising two convolutional blocks, max-pooling, dense classification layers, and a 15class softmax output, with 815,023 trainable parameters— was trained on the PlantVillage benchmark and deployed via a Django web application with MySQL-backed logging and GeoIP geographic mapping. Experimental evaluation confirmed accurate real-time classification across all tested disease categories, and a comprehensive seven-phase testing protocol produced no defects. By combining deep-learning-based classification, collaborative epidemiological mapping, and an accessible web interface, the system represents a meaningful, practical contribution to the vision of data-driven precision agriculture. With the planned enhancements in mobile deployment, multilingual accessibility, environmental data fusion, and automated treatment recommendation, the platform has strong potential to scale into an indispensable diagnostic tool for farming communities worldwide, contributing materially to global food security and the economic resilience of smallholder agriculture

11. References

[1] S. P. Mohanty, D. P. Hughes, and M. Salathe, "Using′ deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016. https://doi.org/10.3389/fpls.2016.01419

[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] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and

[4] Stefanovic, "Deep neural networks based recognition of plant diseases by leaf image classification," *Computational Intelligence and Neuroscience*, vol. 2016, p. 3289801, 2016.

[5] PlantVillage Dataset. [Online]. Available: https://www.kaggle.com/datasets/emmarex/ plantdisease

[6] 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.

[7] M. Brahimi, K. Boukhalfa, and A. Moussaoui, "Deep learning for tomato diseases: Classification and symptoms visualization," *Applied Artificial Intelligence*, vol. 31, no. 4, pp. 299–315, 2017.

[8] A. Kamilaris and F. X. Prenafeta-Boldu, "Deep learning′ in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.

[9] S. Zhang, X. Wu, and Z. You, "Leaf image based cucumber disease recognition using sparse representation classification," *Computers and Electronics in Agriculture*, vol. 134, pp. 135–141, 2017.

[10] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, "A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition," *Sensors*, vol. 17, no. 9, p. 2022, 2017.

[11] M. Arsenovic, M. Karanovic, S. Sladojevic, A. Anderla, and D. Stefanovic, "Solving current limitations of deep learning-based approaches for plant disease detection," *Symmetry*, vol. 11, no. 7, p. 939, 2019.