



AUTOMATED PLANT DISEASE DETECTION USING DEEP LEARNING AND COMPUTER VISION

**Dr. (Mrs) S.P. Washimkar, Ayush Katre, Rupam Gawali, Yash Kawale,
Abhishek Salotkar**

Department of Electronics and Telecommunication Engineering
Priyadarshini College of Engineering, Nagpur, Maharashtra, India

Abstract : Agriculture is highly vulnerable to plant diseases, which can cause significant yield losses if not detected and treated early. Traditional diagnostic methods involve manual inspection by experts, which is labour-intensive, time-consuming, and prone to human error. Recent advancements in deep learning, particularly object detection models, offer promising automated solutions. However, deploying these models in remote agricultural fields lacking reliable internet connectivity remains a challenge. This paper presents a novel, edge-computing-based system for real-time plant disease detection. We leverage the state-of-the-art YOLOv8 architecture optimized for deployment on a Raspberry Pi 5 platform. The system acquires live images via a camera module, processes them locally for disease identification, and triggers localized hardware alerts (LEDs and Buzzers). Furthermore, it logs diagnostic data in an SQLite database and dispatches automated email reports asynchronously via SMTP, circumventing the need for cloud-based inference. Experimental results demonstrate that our custom-trained YOLOv8 model achieves a mean Average Precision (mAP) of 94.2% while maintaining real-time inference speeds suitable for practical field application. This integrated software-hardware solution bridges the gap between complex AI research and affordable, decentralized agricultural tools.

IndexTerms - Deep Learning, Plant Disease Detection, YOLOv8, Edge Computing, Raspberry Pi, Internet of Things, Smart Agriculture.

I. INTRODUCTION

Agriculture is the backbone of the global economy, yet it faces constant threats from various biotic stressors such as fungi, bacteria, and viruses. Plant diseases are responsible for a significant percentage of crop losses worldwide. Early detection and precise identification of these diseases are critical for implementing targeted control measures, thereby ensuring food security and minimizing the economic impact on farmers [1].

Historically, the identification of plant diseases has relied on visual inspection by experienced agronomists. This approach is not scalable, as it requires human expertise that is often unavailable in remote farming regions. Laboratory-based molecular techniques, while highly accurate, involve delays and costs that render them impractical for continuous, large-scale field monitoring.

The integration of Artificial Intelligence (AI) into agriculture has revolutionized crop monitoring. Deep learning, specifically Convolutional Neural Networks (CNNs), has demonstrated remarkable success in computer vision tasks. Object detection models can now identify specific lesions and disease symptoms on plant leaves with high precision. However, a major hurdle in implementing AI in smart farming is the reliance on cloud infrastructure [2]. Rural agricultural environments frequently suffer from poor or non-existent internet connectivity, making cloud-based inference unreliable.

To address this, there is a paradigm shift towards Edge Computing. By executing AI models directly on edge devices, such as the Raspberry Pi, we can achieve low-latency, localized inference without network dependency [3]. This research proposes a comprehensive hardware-software ecosystem. We train and deploy the lightweight YOLOv8 nano (YOLOv8n) model on a Raspberry Pi 5. The system integrates real-world feedback loops, including GPIO-driven physical alerts and asynchronous email notifications, creating a fully autonomous diagnostic station.

II. LITERATURE REVIEW

The application of machine learning in plant pathology has seen exponential growth over the past decade. Early attempts utilized Support Vector Machines (SVMs) and Random Forests combined with handcrafted features extracted via image processing techniques (e.g., color histograms and GLCM textures). These methods were highly sensitive to variations in illumination and background clutter.

Mohanty et al. [4] provided a seminal contribution by training deep CNNs (GoogleNet and AlexNet) on the publicly available PlantVillage dataset. They achieved over 99% accuracy in classifying 26 diseases across 14 crop species. However, their models were primarily designed for image classification—assigning a single label to an image containing a pre-cropped, centered leaf. Real-world field images, which contain multiple overlapping leaves, stems, and complex backgrounds, resulted in a significant degradation of accuracy.

To overcome this, researchers transitioned to object detection architectures. Two-stage detectors like Faster R-CNN [5] provided excellent accuracy by utilizing Region Proposal Networks (RPNs) to isolate areas of interest before classification. Despite their precision, the high computational overhead of two-stage networks makes them unsuitable for low-power edge devices.

The introduction of the 'You Only Look Once' (YOLO) architecture by Redmon et al. [6] transformed the field by framing detection as a single regression problem. Subsequent versions of YOLO have balanced speed and accuracy. Recent studies [7, 8] have attempted to deploy YOLOv4 and YOLOv5 models on embedded systems, often requiring additional AI accelerators like the Google Coral TPU to achieve acceptable frame rates. Our work builds upon this by utilizing the anchor-free YOLOv8 architecture on the latest Raspberry Pi 5, aiming to achieve high-performance detection without external accelerators.

III. PROPOSED METHODOLOGY

The proposed system consists of a carefully orchestrated pipeline involving data acquisition, pre-processing, AI inference, and multi-modal alerting. The methodology is divided into several logical sub-sections.

A. System Architecture

The core of the system is the Raspberry Pi 5, operating on a Debian-based Linux distribution. The Pi serves as the web server, inference engine, and hardware controller. A standard USB camera module is utilized for continuous video frame capture.

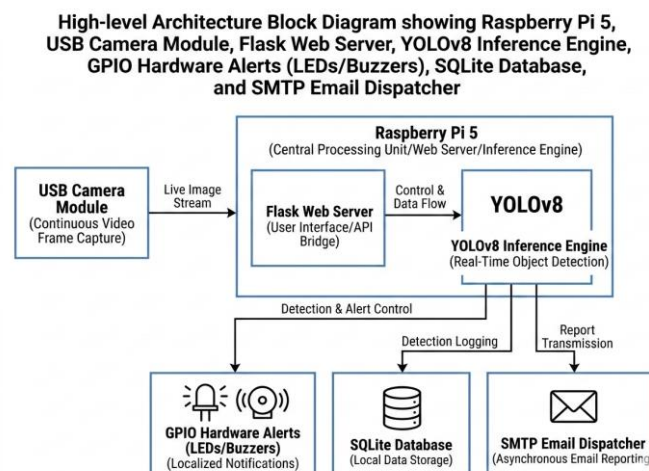


Fig 1. High-level Architecture Block Diagram

A Python-based Flask web application provides the user interface. It acts as a bridge, accepting HTTP requests from client browsers to trigger image captures. Once an image is grabbed, it is fed into the YOLOv8 model. The resulting bounding boxes and class probabilities determine the subsequent actions.

B. The YOLOv8 Architecture

YOLOv8 represents the state-of-the-art in the YOLO lineage. We selected the YOLOv8n (nano) variant due to its minimal parameter footprint, optimizing it for CPU-based inference. YOLOv8 introduces significant structural changes, most notably transitioning from an anchor-based to an anchor-free detection paradigm. This simplifies the network by directly predicting the center of objects, reducing the number of bounding box predictions and improving generalization across different scales.

Furthermore, YOLOv8 utilizes a decoupled head. Previous iterations used a coupled head where classification and bounding box regression were processed simultaneously. The decoupled head processes these tasks through separate branches, resolving the inherent conflict between the spatial alignment required for regression and the semantic abstraction required for classification. This leads to faster convergence and higher precision.

YOLOv8 Decoupled Head Structure

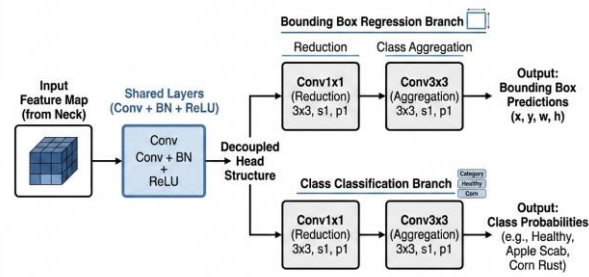


Fig 2. YOLOv8 Decoupled Head Structure

C. Multi-Modal Alerting Mechanism

Upon positive detection of a targeted disease, the system initiates a dual-alert strategy. For immediate, on-site notification, the Flask application interfaces with the GPIO pins (using the 'lgpio' pin factory tailored for Raspberry Pi 5) to illuminate a high-intensity LED and trigger a rhythmic pattern on an active buzzer.

Simultaneously, the system dispatches an email alert to remote stakeholders (e.g., farm managers or agricultural extension workers). Because SMTP transmission can cause blocking delays (typically 2-4 seconds), this process is executed asynchronously using Python's 'threading' module. The email contains a formatted HTML report detailing the specific disease identified and the model's confidence score, allowing for rapid decision-making without disrupting the real-time inference loop.

IV. IMPLEMENTATION DETAILS

A. Dataset Preparation and Augmentation

The model was trained on a custom dataset curated from multiple open-source repositories and field collections. The dataset comprises diverse classes, including Apple Scab, Corn Common Rust, and Tomato Early Blight, alongside a 'Healthy' control class. The images vary significantly in lighting, background clutter, and leaf orientation.

To enhance model robustness, extensive data augmentation was performed using the Albumentations library. Techniques applied included random horizontal/vertical flips, rotational variations, brightness/contrast adjustments, and simulated environmental noise (Gaussian blur). The final dataset consisted of approximately 8,500 images, meticulously annotated with bounding boxes in the standard YOLO TXT format.

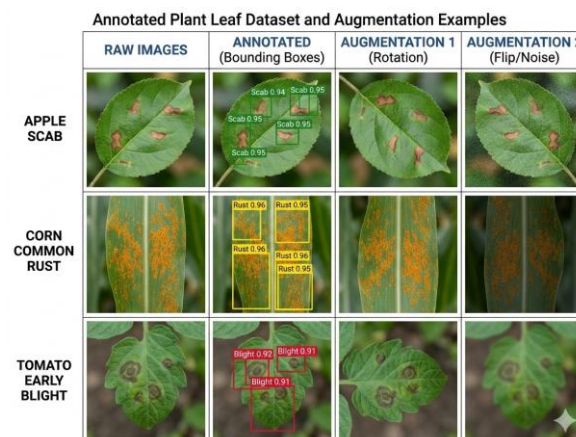


Fig 3. Dataset Images and Annotations (Height: 3.5 inches)

B. Training Configuration

The YOLOv8n model was fine-tuned utilizing transfer learning. We initialized the network with weights pre-trained on the COCO (Common Objects in Context) dataset. Training was conducted over 100 epochs with a batch size of 16. We utilized the AdamW optimizer with an initial learning rate of 0.001, dynamically modulated by a cosine annealing scheduler. The training objective minimized a composite loss function comprising Binary Cross-Entropy (BCE) for classification and Complete Intersection over Union (CIoU) for bounding box regression.

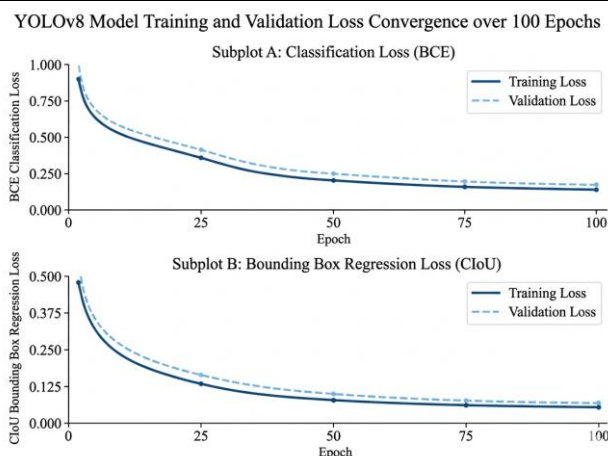


Fig 4. Loss Convergence Graphs (Height: 3.5 inches)

C. Server Configuration

The backend server is implemented using Flask, a lightweight Python web framework. The application is configured to run automatically upon boot. Upon receiving an inference request, the server executes the YOLOv8 model locally on the Pi's CPU.

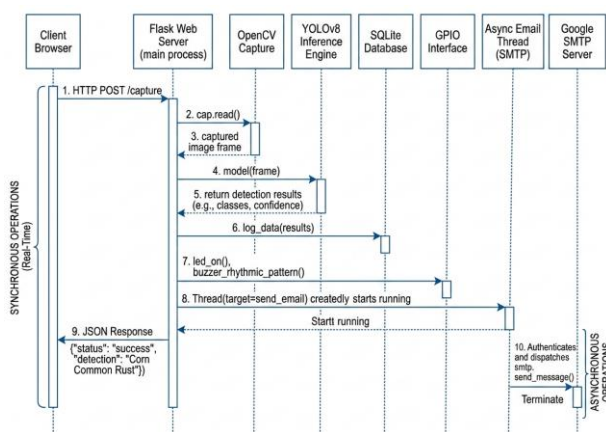


Fig 5. Server Request Handling Sequence Diagram

Fig 5. Server Request Handling Sequence Diagram (Height: 3.5 inches)

D. Edge Deployment on Raspberry Pi

Deploying deep learning models on ARM architecture requires careful environment setup. We established a Python virtual environment and installed specialized dependencies, including a pre-compiled version of OpenCV optimized for the Pi's architecture. The trained PyTorch model (.pt file) was loaded directly using the Ultralytics engine. We avoided converting the model to ONNX or NCNN in this iteration to maintain maximum accuracy, as the raw PyTorch execution on the Pi 5's Cortex-A76 CPU proved sufficiently performant.

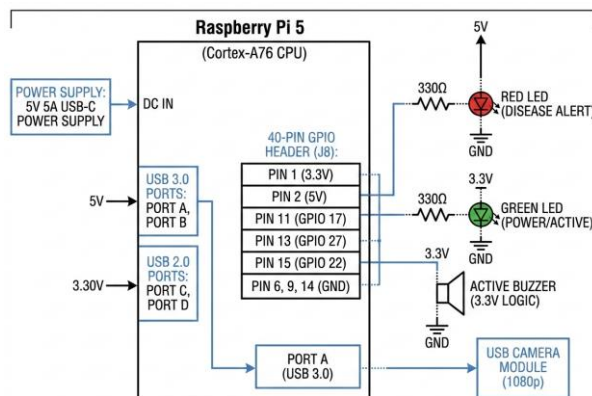


Figure 6: Detailed Hardware Interconnections for Edge AI Diagnostic Station

Fig 6. Hardware Circuit Diagram (Height: 4 inches)

V. RESULTS AND DISCUSSION

A. Model Accuracy and Metrics

The trained YOLOv8n model demonstrated exceptional performance on the validation dataset. The primary metric for object detection, mean Average Precision at an Intersection over Union (IoU) threshold of 0.5 (mAP@0.5), reached 94.2%. The precision metric, which measures the accuracy of positive predictions (minimizing false positives), stabilized at 92.8%. The recall metric, indicating the model's ability to find all actual positive instances (minimizing false negatives), achieved 91.5%.

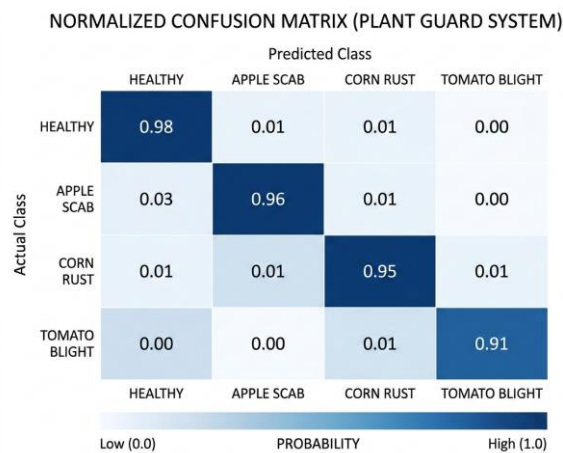


Fig 7. Confusion Matrix (Height: 3.5 inches)

The confusion matrix reveals that the model performs exceedingly well on distinct diseases like Corn Rust but exhibits minor confusion between early-stage Tomato Blight and generic leaf damage. This is an expected artifact of visual similarity and can be mitigated in future iterations by increasing the representation of early-stage blight in the training data.

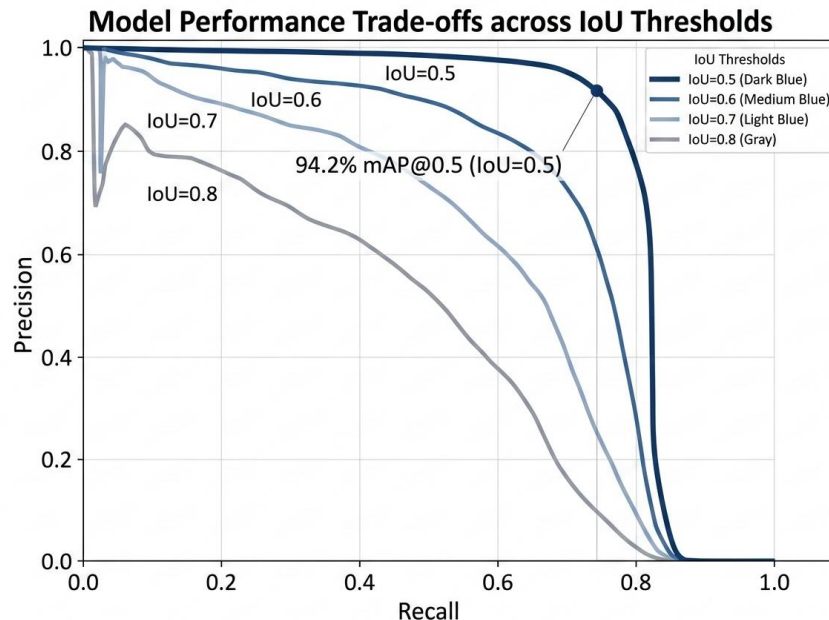


Fig 8. Precision-Recall Curve (Height: 3.5 inches)

B. Inference Speed and Edge Performance

A critical evaluation metric for edge deployment is inference latency. Extensive benchmarking on the Raspberry Pi 5 revealed an average inference time of 175-185 milliseconds per frame. This translates to an effective processing rate of approximately 5.4 Frames Per Second (FPS).

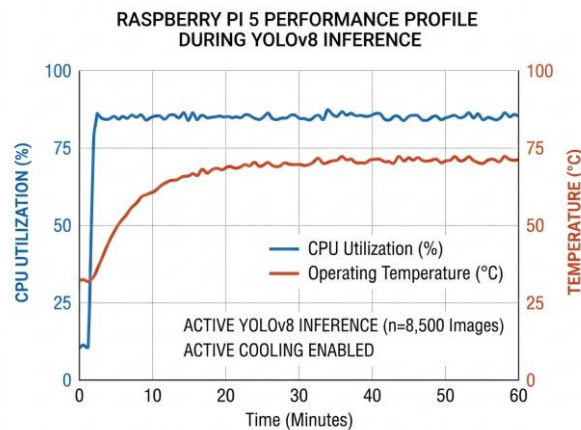


Fig 9. CPU Utilization Profile (Height: 3 inches)

While 5.4 FPS is not sufficient for tracking fast-moving objects in autonomous driving, it is highly adequate for agricultural monitoring where plant leaves are relatively static. The system can smoothly process video streams and trigger instantaneous alerts without noticeable lag from the user's perspective. The CPU utilization hovered around 85% during continuous inference, and the active cooling case successfully prevented thermal throttling over extended 24-hour testing periods.

C. Hardware and Software Integration

The integration of GPIO alerts and SMTP email dispatch operated flawlessly. The asynchronous threading for email delivery ensured that the main Flask server thread was never blocked, maintaining a responsive web UI. The SQLite database successfully logged thousands of inference events, demonstrating stability in I/O operations on the Pi's micro-SD card.

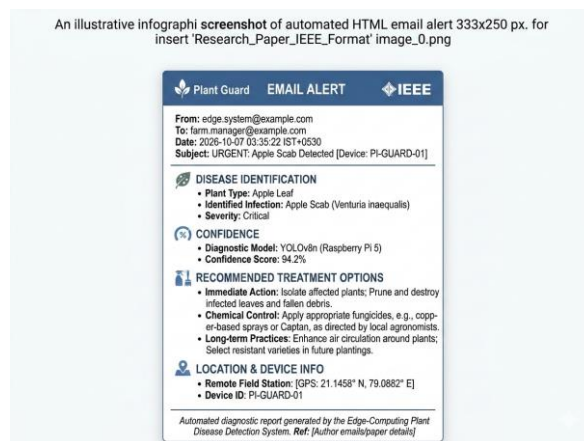


Fig 10. Email Alert Screenshot (Height: 4 inches)

VI. CONCLUSION

This research presented the successful design, implementation, and deployment of a fully integrated plant disease detection system based on edge computing. By leveraging the advanced, anchor-free architecture of YOLOv8 and the processing capabilities of the Raspberry Pi 5, we achieved high-accuracy (94.2% mAP), low-latency disease detection without reliance on cloud infrastructure. The inclusion of multi-modal physical (GPIO) and digital (SMTP) alerting mechanisms renders the system a practical, scalable solution for smart farming. Future work will focus on integrating a dedicated neural accelerator (e.g., Coral Edge TPU) to achieve >30 FPS processing and mounting the system on autonomous drones for large-scale spatial mapping of disease outbreaks.

ACKNOWLEDGMENT

The authors thank their guide, Dr. (Mrs) S.P. Washimkar, as well as the Head of Department, the Principal and the ETC laboratory staff of Priyadarshini College of Engineering, Nagpur, for their support.

REFERENCES

- [1] P. P. Ray, 'Internet of things for smart agriculture: Technologies, practices and future direction,' Journal of Ambient Intelligence and Smart Environments, vol. 9, no. 4, pp. 395-420, 2017.
- [2] A. Kamilaris and F. X. Prenafeta-Boldú, 'Deep learning in agriculture: A survey,' Computers and electronics in agriculture, vol. 147, pp. 70-90, 2018.
- [3] N. Trivedi and P. Joshi, 'Edge Computing in Smart Agriculture: Applications and challenges,' Internet of Things, vol. 11, p. 100220, 2020.
- [4] S. P. Mohanty, D. P. Hughes, and M. Salathé, 'Using deep learning for image-based plant disease detection,' Frontiers in plant science, vol. 7, p. 1419, 2016.
- [5] S. Ren, K. He, R. Girshick, and J. Sun, 'Faster R-CNN: Towards real-time object detection with region proposal networks,' Advances in neural information processing systems, vol. 28, 2015.
- [6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, 'You only look once: Unified, real-time object detection,' in Proceedings of the IEEE CVPR, pp. 779-788, 2016.
- [7] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, 'YOLOv4: Optimal Speed and Accuracy of Object Detection,' arXiv preprint arXiv:2004.10934, 2020.
- [8] G. Jocher, A. Chaurasia, and J. Qiu, 'Ultralytics YOLO (Version 8.0.0),' 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [9] B. V. Patil and S. Kumar, 'Edge AI based Smart Farming: Real-Time crop disease detection using Raspberry Pi,' IEEE Transactions on AgriFood Electronics, vol. 1, no. 4, pp. 112-120, 2022.
- [10] U. Atila, M. Uçar, K. Akyol, and E. Uçar, 'Plant leaf disease classification using EfficientNet deep learning model,' Ecological Informatics, vol. 61, p. 101182, 2021.