University of Hertfordshire UH

PROJECT REPORT

BEng Robotics and Artificial Intelligence

Name – Tomiwa Adeyemi

Supervisor - Zakaria Chekata

AI–Based Smart Agriculture system for Crop Health Monitoring

20th March 2026

BACHELOR OF ENGINEERING DEGREE/DEGREE WITH HONOURS IN ROBOTICS AND ARTIFICIAL INTELLIGENCE

BEng Individual Project Report

Department of Engineering
School of Physics, Engineering and Computer Science
University of Hertfordshire

AI–Based Smart Agriculture system for Crop Health Monitoring

Report by
Tomiwa Adeyemi

Supervisor
Zakaria Chekata

Date
20th March 2026

ABSTRACT

This project addresses the challenge of deploying deep learning models for plant disease detection on edge devices with limited computational resources. Using the PlantVillage dataset, which includes 38 disease categories across 54,305 labelled images, a MobileNetV2 model was constructed and strengthened via a preprocessing pipeline incorporating grayscale conversion and Gaussian noise injection to enhance robustness against real-world imaging conditions.

To reduce the model's computational footprint for deployment on edge hardware, a precise layer-by-layer unstructured pruning strategy was applied, eliminating 14.67% of eligible weights while protecting the first convolutional and final classification layers from sparsification. Although pruning initially reduced accuracy from 88.31% to 60.58%, a focused fine-tuning phase using a conservative learning rate of 0.0001 restored and surpassed the original performance. The final optimised model achieved a deployed test accuracy of 97.56% and a Macro Average F1-score of 0.97, indicating strong generalisation and balanced performance across all 38 classes.

These results suggest that careful model compression, when combined with strategic retraining, can not only maintain but enhance model effectiveness. It should be noted that this outcome is based on a single experimental run and a single dataset split and should be treated as a strong observed result rather than a generalised population estimate. This work establishes a reliable, lightweight, and interpretable framework for deploying high-performance AI solutions in resource-constrained agricultural settings.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Zakaria Chekata for their invaluable guidance, technical feedback, and continuous support throughout this project.

Use of AI-assisted proofreading tool: Grammarly was used during the proofreading stage to improve grammar, spelling, punctuation, and clarity of expression. The technical content, methodology, implementation, analysis, and conclusions presented in this report are my own.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

AI	Artificial Intelligence — the simulation of human cognitive processes by computer systems, including learning, reasoning, and self-correction.
ML	Machine Learning — a subfield of AI in which systems learn patterns from data without being explicitly programmed.
CNN	Convolutional Neural Network — a class of deep learning model specifically designed for image data, which learns spatial hierarchies of features through convolutional layers.
K-NN	K-Nearest Neighbour — a supervised classification algorithm that assigns a label to a new data point based on the majority class among the k closest training samples in feature space.
SVM	Support Vector Machine — a supervised learning model that identifies the optimal hyperplane maximising the margin between classes in high-dimensional feature space.
Grad-CAM	Gradient-weighted Class Activation Mapping — a visualisation technique that uses class-specific gradients flowing into the final convolutional layer to produce a spatial heatmap highlighting image regions most influential to a classification decision (Selvaraju et al., 2017).
JIT	Just-In-Time compilation — a technique that compiles code at execution time rather than ahead of time; used in PyTorch's torch.compile() to deliver significant inference speedups.
XAI	Explainable Artificial Intelligence — a set of methods and processes that make AI model outputs and internal reasoning interpretable and transparent to human users.
ReLU	Rectified Linear Unit — a non-linear activation function defined as f(x) = max(0, x), used in deep neural networks to introduce non-linearity while avoiding the vanishing gradient problem.
SOTA	State of the Art — the highest level of performance currently achieved by any published method on a given benchmark task.
RFs	Random Forests — an ensemble learning method that constructs multiple decision trees during training and outputs the mode of their individual predictions for classification tasks.
VRAM	Video Random Access Memory — dedicated memory on a GPU used to store model parameters, activations, and gradients during deep learning training.
mAP	Mean Average Precision — a metric used in object detection that averages the precision-recall area under the curve across all classes and detection thresholds.
AWGN	Additive White Gaussian Noise — a basic noise model in which noise values are drawn from a Gaussian distribution and added independently to each pixel value.

1. Introduction

1.1 Background and Motivation

Agriculture has sustained human civilisation for over 10,000 years, yet the fundamental challenge of detecting plant diseases has never been fully resolved. From the earliest deliberate cultivation of emmer wheat in the Fertile Crescent, farmers have relied on direct observation to identify and respond to crop disease, a practice that remained essentially unchanged until the mid-twentieth century, when systematic plant pathology, chemical diagnostics, and eventually remote sensing began to extend the reach and precision of diagnosis (Diamond, 1997). Despite a century of scientific progress, plant diseases caused by fungal, bacterial and viral pathogens remain among the most economically destructive threats to global food security. Savary et al. (2019) estimated annual yield losses attributable to pathogens and pests at 21.5% for wheat, 30.0% for rice, and 22.5% for maize in a landmark Nature Ecology & Evolution study, figures that have changed little over the decades despite widespread pesticide use. Critically, the highest losses are concentrated in food-deficient, low-income regions: precisely the communities least equipped to absorb them.

Table 1.1 outlines the development of plant disease diagnosis: from early expert assessments to today's deep learning, showing advances in capability alongside persistent engineering challenges. Each new technological generation solves its predecessor's main issue but introduces a constraint, prompting the next phase.

Period	Approach	Key Limitation
Pre-1960s	Expert visual inspection and field scouting	Slow (days to weeks for lab confirmation); subjective; geographically constrained; inaccessible to smallholder farmers in remote areas
1960s-1990s	Laboratory microscopy and chemical pathogen assays	High accuracy but expensive; required specialist equipment; entirely inaccessible without physical laboratory access.
1990s-2010s	Remote sensing; satellite and aerial multispectral imaging	High equipment cost; not scalable to single-pant or leaf-level diagnosis; required airborne platforms.
2010-2015	Classical ML: SVM, KNN with hand crafted features	Required manual feature engineering; poor

		generalisation across crop varieties, lighting, and background conditions.
2016-present	Deep CNN – PlantVillage era (Mohanty et al., 2016)	Near-perfect lab accuracy but severe robustness gap in field; black-box predictions; architecture too large for edge hardware.

Table 1.1: Historical evolution of plant disease diagnosis methods and key limitations at each stage of development

A turning point in the modern era came in 2016, when Mohanty et al showed that a Convolutional Neural Network (CNN), fine-tuned on the PlantVillage dataset, which contains more than 54,000 labelled leaf images across 38 disease classes (Hughes and Salathe, 2015), could achieve a classification accuracy of 99.35%. This single study effectively rendered features obsolete. CNNs accomplish this by learning features hierarchically from raw pixel values: initial layers detect basic visual elements such as edges and texture, while deeper layers combine these into complex, disease-specific patterns, such as necrotic spots, chlorotic discolouration, and signs of fungal development (LeCun et al., 2015). In contrast to traditional machine learning approaches, which relied on researchers to handcraft feature representations, this end-to-end method automatically adapts to the underlying data structure, enabling a single model to perform well across multiple crops and diseases.

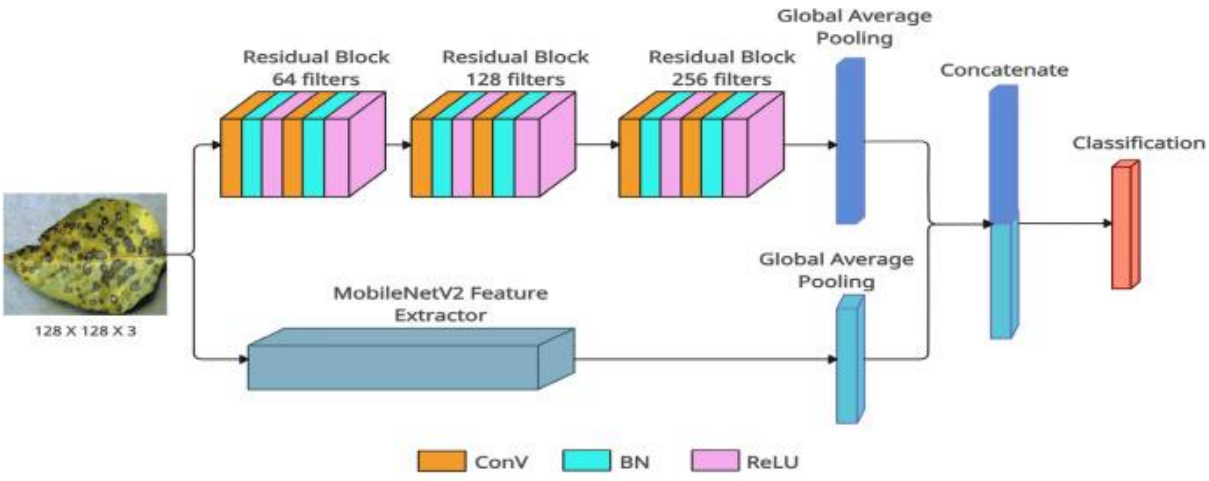

Figure 1: Architecture of a CNN-based plant disease classification pipeline, showing residual blocks and a parallel MobileNetV2 feature extractor branch processing a diseased leaf input (128x128x3) with features concatenated before final classification output. Source: Pal et al.,

The increasing spread of smartphones in rural areas has opened an unprecedented chance to implement smart diagnostics tools directly on edge devices. By 2023, more than half of the population in sub-Saharan Africa, where small-scale farming plays a vital economic role owns a smartphone, and this figure is still rising (GSMA, 2023). This means that access to devices or network connectivity is no longer the main obstacle to running an effective plant disease classifier on consumer phones. Instead,

the key challenges lie in reducing model size, improving inference speed, and ensuring reliable predictions exactly the technical issues this project aims to solve

1.2 Precision Agriculture and the Case for AI

Precision Agriculture is the application of information technologies and AI-driven sensing principles to manage spatial and temporal variability across all aspects of agricultural production, with the dual objective of improving crop yield and minimising environmental impact (Zhang et al., 2002).

Precision Agriculture emerged in the 1990s as a paradigm shift from uniform field management to targeted, data-driven management of spatial and temporal field variability (Zhang et al., 2002). Where conventional agriculture applies inputs of water, fertiliser and pesticide uniformly across a field regardless of local conditions, precision agriculture uses sensor data, satellite imagery, and increasingly artificial intelligence to apply exactly the right input, in exactly the right place at exactly the right time.

The economic benefits of precision agriculture are well documented. Nations that have implemented data-driven farming methods such as the United States, Germany and the Netherlands, regularly achieve significantly higher crop yields per hectare than developing countries using traditional techniques, even though their total farmed areas are often smaller (Rajmane et al., 2020). This productivity gap is not due to better soil or climate conditions, but rather to the consistent application of informed, intelligent decision-making in agricultural management.
Table 1.2 outlines the differences between conventional practices and AI-enhanced approaches across core farming activities related to crop health, illustrating both the type and extent of improvements enabled by each technological advancement.

Agricultural Task	**Conventional Practice**	**AI-Powered Precision Approach**
Disease Diagnosis	Expert visual inspection; lab testing (days to weeks)	CNN classification on smartphone in seconds; no lab access required
Pest Monitoring	Manual field scouting; costly and infrequent	Automated drone surveys with real-time AI object detection
Pesticide Application	Uniform blanket spraying regardless of actual disease location	Targeted spot-spray guided by AI lesion localisation
Irrigation	Fixed schedule regardless of soil moisture	Sensor-driven adaptive irrigation; reducing water use by up to 40%

Yield Estimation	Manual counting: sampling and extrapolation	Computer vision fruit counting and sizing in real time

Table 1.2: Comparison of conventional agricultural practices and AI-powered precision approaches across key crop management tasks relevant to disease management.

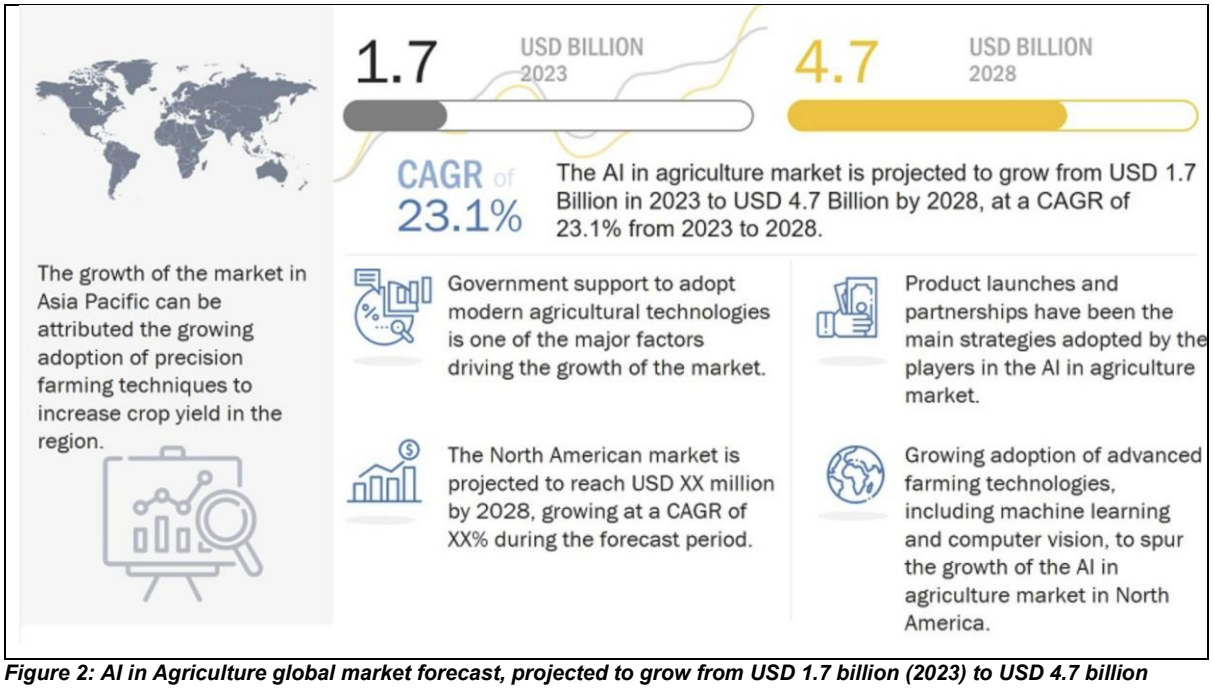

Figure 2: AI in Agriculture global market forecast, projected to grow from USD 1.7 billion (2023) to USD 4.7 billion (2028) at a CAGR of 23.1% driven by government adoption of precision farming technologies and growing machine learning deployment in crop management.

1.3 Problem Statement

While the current literature demonstrates the high accuracy of deep learning in multi-class plant disease classification, three co-existing and persistent engineering problems prevent the direct translation of this laboratory capability into field-ready, farmer-accessible diagnostic tools. Each problem is distinct in nature but interconnected in consequence; addressing any one in isolation is insufficient.

The first problem is the robustness gap. PlantVillage's imaging approach, photographing individual detached leaves against plain grey or black backgrounds under controlled studio lighting, differs drastically from real-world farm conditions, where images are taken in natural light, feature intricate soil and plant backgrounds, inconsistent shadows, and vary in quality due to the use of affordable smartphone cameras (Hughes and Salathe, 2015). Mohanty et al.(2016) measured the real-world impact of this discrepancy: a CNN that achieved 99.35% accuracy on PlantVillage data dropped to as low as 31% when tested on field-collected images. Further investigation by Brahimi et al. (2017) using saliency maps revealed that some models focus on background colour patterns rather than the visual characteristics of disease lesions. This reliance on irrelevant features, which is a sign of dataset bias, makes such models ineffective for practical field use.

The second problem is the excessive number of parameters in standard models. Architectures like VGG-16 with 138 million parameters, and ResNet-50 with 25 million demands significant computational power resources that battery-powered smartphones and IoT edge cameras commonly used by smallholder farmers, typically lack. Running such large models on these devices results in slow inference times, high memory usage, and rapid battery drain (Han et al., 2015). Critically, MobileNetV2, with approximately 3.5 million parameters, requires roughly 93% fewer parameters than the baseline CNN developed in this project, making it the only architecturally viable candidate for edge deployment of the two models evaluated. There is therefore a pressing need to reduce model size while preserving the diagnostic precision required for sound agronomic decisions.

The third problem is the “black box” nature of deep networks. A high validation accuracy on a controlled dataset provides no spatial evidence that lesion features rather than background artefacts are driving predictions. This opaqueness is not simply a theoretical concern: Brahimi et al. (2017) demonstrated empirically that it directly causes field failure. Without interpretable, spatially grounded predictions, agricultural stakeholders are unable to identify when or why a model is failing, and the system cannot be trusted for consequential agronomic decisions such as pesticide application, crop quarantine or harvest timing.

Together, these three challenges define the core engineering problem this project addresses: the development of a classification pipeline that is simultaneously accurate across 38 disease classes, robust under realistic imaging perturbations, spatially explainable through Grad-CAM heatmap validation, and sufficiently compressed for deployment on limited-resource edge devices whilst preventing the catastrophic forgetting that naïve pruning approaches produce.

1.4 Aim and Objectives

This project aims to design, thoroughly assess, and efficiently compress a deep learning model for multi-class classification across 38 different plant diseases, with optimisation focused on edge device deployments whilst preventing catastrophic forgetting during compression.
Table 1.3 maps each specific objective to the engineering challenge it addresses and the technical approach employed.

Engineering Challenge Addressed	**Technical Approach**
Parameter bloat prevents mobile deployment	Layer-wise L1 unstructured pruning at 14.67% sparsity with post-pruning, fine-tuning
Benchmark accuracy does not reflect real-field robustness	Gaussian noise $\mathrm{N}(0, \sigma^2)$ and grayscale conversion applied as robustness stressors
Black-box predictions not trusted by end-users	Grad-CAM heatmap validation on the final convolutional layer of both architectures

No direct Baseline CNN vs MobileNetV2 comparison in literature	Controlled head-to-head training on identical 70/15/15 PlantVillage split
Catastrophic forgetting during compression	Preserve first conv and classifier layers; fine-tune at low learning rate post-pruning

Table 1.3: Project technical objectives mapped to the engineering challenges they address, and the specific methodological approaches employed.

To meet this primary aim, the following specific objectives have been established:

- **Build a Robustness Pipeline:** Create a data preprocessing workflow that incorporates grayscale transformation and the addition of Gaussian noise. This approach is intended to minimize reliance on colour cues, encouraging the model to focus on core structural and textural patterns in leaf lesions.
- **Conduct Architectural Evaluation:** Develop, train, and assess a custom baseline Convolutional Neural Network (CNN), then compare its performance with MobileNetV2 (a pre-designed transfer learning optimised for mobile environments).
- **Ensure Spatial Interpretability:** Apply Gradient-weighted Class Activation Mapping (GRAD-CAM) to produce visual heatmaps, verifying that the model bases its predictions on relevant biological feature in the image rather than irrelevant background elements.
- **Optimise for Edge Deployment:** Decrease the computational load of the best-performing model using a targeted layer-wise unstructured pruning method, targeting approximately a 15% reduction in the number of parameters.
- **Restore Classification Performance:** Perform fine-tuning after pruning using a low learning rate to repair weakened connections. The goal is to evaluate class-level performance using comparative confusion matrices and ensure the final deployed model surpasses the original baseline accuracy.

1.5 Risk Management and Mitigation

Following professional engineering standards, the project used disciplined project and time management to systematically address technical risks throughout development.

The most anticipated challenge centred on hardware and computational constraints. Training a complex 38 class dataset required substantial Video RAM (VRAM), and reliance on cloud-based GPU resources carried ongoing risks such as session interruptions, memory workflows and the potential loss of trained model weights. To counter these issues, the approach incorporated tightly controlled data batching and automated checkpointing of model state dictionaries after each major training milestone, ensuring continuous protection of model integrity.
A more critical and anticipated risk materialised during the model compression phase. The initial pruning strategy applied global, unstructured L1-magnitude pruning, removing 30% of the smallest -

magnitude weights across all layers of the network in a single pass. Empirical testing revealed that this caused a catastrophic collapse in classification accuracy to 46%, a fall of over 42 percentage points from the pre-pruning baseline of 88.31%. This failure is characteristic of the catastrophic forgetting phenomenon: simultaneous aggressive removal of weights across all layers disrupts the learned feature representations stored throughout the network, particularly the early convolutional layer responsible for foundational edge and texture detection (Frankle and Carlin, 2019).

In response, the pruning strategy was improved through an iterative process. The updated algorithm applies pruning layer-by-layer specifically protecting the first convolutional layer and the final classification layer from sparsification. By focusing more aggressive pruning only on intermediate layers and lowering the overall sparsity target to 14.67%, the initial accuracy drop after pruning was limited to a more manageable decrease, falling from 88.31% to 60.58%.

Given the background, problem statement, and objectives established in this chapter, the following research questions define the analytical scope of this project:

- Can a custom Baseline CNN and MobileNetV2 achieve comparable classification accuracy on the 38-class PlantVillage benchmark under identical training conditions, and what architectural factors explain the performance differences between them?
- Do both architectures exhibit significant accuracy degradation under Gaussian noise and grayscale perturbations, and which architecture demonstrates superior robustness under simulated field conditions?
- Do Grad-CAM heatmaps confirm that predictions are grounded in disease lesion morphology rather than background artefacts, and what do the visualisations reveal about qualitative differences in feature learning between architectures?
- Can layer-wise L1 unstructured pruning at approximately 15% sparsity, followed by fine-tuning, preserve and ultimately surpass the pre-pruning classification accuracy whilst producing a model suitable for edge deployment?

Report Structure

The remainder of this report is organised as follows. **Chapter 2** provides the theoretical background and critical literature review, covering CNN architectures for plant pathology, the robustness gap, Explainable AI methods with particular focus on Grad-CAM, and neural network pruning techniques including the Lottery Ticket Hypothesis. **Chapter 3** describes the methodology in full, detailing dataset preparation, model architectures, training procedures, robustness evaluation protocol, Grad-CAM implementation, and the layer-wise pruning strategy. **Chapter 4** presents and analyses the experimental results, including classification performance metrics, robustness testing outcomes, heatmap analysis, and the pruning–accuracy trade-off. **Chapter 5** discuss the project's management. **Chapter 6** concludes the project with a summary of technical contributions, acknowledgement of limitations, and recommended directions for future work.

2. Literature Review

Today's global agriculture faces growing pressure from the dual challenges of a rising population and inherent weaknesses in crop production systems. With the world expected to reach 10 billion people by 2050, food output must increase by around 50% to meet basic needs (FAO,2022). This task is made far more difficult by the ongoing impact of plant pests and diseases, which already cause annual losses of 20% to 40% of global crop yields (Strange & Scott, 2005; FAO, 2019). These losses go beyond reduced food availability; they also impose severe economic cost, with plant diseases alone costing over $200 billion per year and invasive insects adding at least $70 billion in additional damage (FA0, 2022).

The causes of these losses are highly complex. They involve a vast array of viruses, bacteria, fungi, oomycetes, nematodes, and insects, each with multiple strains and genetic variations (Strange & Scott 2005; Anderson et al., 2004). These organisms have co-evolved with farming over thousands of years. Their spread is fuelled by both natural means, like wind and rainfall, and human activities such as global trade and travel, which introduce pests into region with little or no resistance (Anderson et al., 2004). Climate change further intensifies the problem, accelerating pathogen adaption to new environments and host, undermining conventional control methods Table 2.1 summarises the scale of this global challenge

Impact Category	**Statistical Magnitude**
Global Crop Production Loss	20% - 40% (FA0 2019) Strange & Scott (2005)
Economic Loss from Disease	> $220 Billion (FAO 2022)
Economic Loss from Invasive Insects	> $70 Billion (FAO 2022)
Projected Population (2050)	10 Billion (FAO 2022)
Required Increase in Food Production	50% (FAO 2022)
Historical Pest Spread Vectors	Trade, Travel, Natural Dispersal Anderson et al. (2004); Strange & Scott (2005)

Table 2.1: Summary of global agricultural impact statistics.

The conventional approach to reducing these losses has long relied on manual, visual inspections conducted by trained agronomist or farmers. Yet this method is now widely seen as a limiting factor in modern agricultural systems (Pandey et al., 2024). Inspecting crops by hand is slow, labour-intensive and prone to human error. Because visual evaluation relies on individual judgement, results can vary depending on the inspector's level of experience, tiredness or even momentary conditions (Kahneman et al., 2021).

Additionally, monitoring capabilities differ sharply between developed and developing countries. Research show that wealth, especially GDP per capita, closely correlates with a country's ability to detect and report crop pests (Bebber et al., 2013). This suggests that current global knowledge severely

underestimates the true pest load in tropical regions, necessitating the development of automated, objective, and low-cost diagnostic tools (Bebber et al., 2013; Mahesh et al., 2023).

2.1 Evolution of Convolutional Neural Networks

The shift from manual feature extraction to automated deep learning is the most significant advancement in 21st century agricultural diagnostics (Mahesh et al., 2023). Prior to deep learning, traditional machine learning methods such as SVMs, KNNs and RFs were used for disease classification but relied on handcrafted features such as colour, texture and shape. Their performance relied heavily on the quality of these features and did not generalise well across different lighting or crop varieties (Mahesh et al., 2023).

Convolutional Neural Networks (CNNs) revolutionised this field by mimicking the hierarchy structure of the human visual cortex to learn discriminative patterns directly from raw image data (LeCun et al., 2015). By automatically discovering relevant features without manual engineering, CNNs significantly enhance diagnostic accuracy and reduced dependency on expert input. Stacking convolutional layers, max-pooling layers and nonlinear activations enables CNNs to identify low-level features, such as edges and spots, in early layers and capture high-level conceptual features such as fungal growth patterns in deeper layers (Varma et al., 2025). This hierarchical automation enables more robust, scalable solutions for agricultural diagnostics.

Architectural Heterogeneity: From VGG to MobileNetV2

The range of CNN architecture used in plant pathology varies widely, from resource-intensive models suited for powerful servers to lightweight designs intended for mobile devices. Models such as VGG-16 and ResNet-50 have long been standard references thanks to their strong accuracy. VGG-16 relies on a deep sequence of small 3x3 convolutional filters to detect fine details (Simonyan & Zisserman, 2014), while ResNet introduced skip connections also known as residual blocks that let gradients pass through shortcuts, skipping over multiple layers. This design helps overcome the vanishing gradient problem, which previously hindered the development of deeper networks (He et al., 2016).

Despite their performance, these models are often unsuitable for small-scale farmers because of their large number of parameters and high computational demands. VGG-16 for instance, includes around 138 million parameters, making it difficult to run on devices with limited memory and power (Simonyan & Zisserman, 2014; Mahesh et al., 2023). As a result, more efficient alternatives have gained attention, with MobileNetV2 becoming a leading choice for smart agriculture applications on edge devices

MobileNetV2 improves efficiency through two main innovations: depthwise separable convolutions and inverted residual bottlenecks (Sandler et al., 2018). Unlike standard convolutions, which handle spatial filtering and channel mixing at the same time, depthwise separable convolutions split the process into two stages. First, a depthwise convolution applies a separate filter to each input channel. Then, a pointwise convolution (1x1) combines the outputs across channels (Sandler et al., 2018). This

factorisation reduces the computational burden by a factor of 8 to 9 compared to standard convolutions, as shown in Table 2.2 (Sandler et al., 2018).

Convolution Type	Parameter Formulation	Computational Cost
Standard Convolution	$K^2 \times C \times C'$	High (Simultaneous channel)
Depthwise Separable	$(k^2 + C') \times C$	Low (Factorized operations): 8/9 x reduction

Table 2.2: Parameter formulation and computational cost comparison between standard and depthwise separable convolutions.

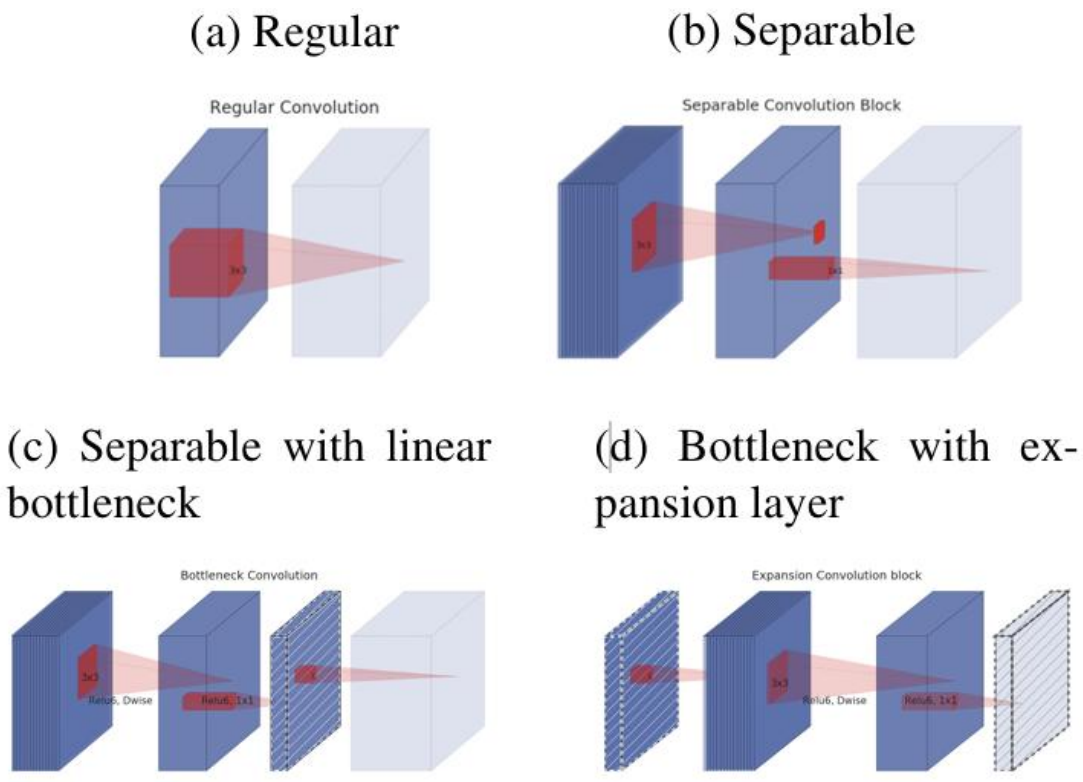

Figure 3: Evolution of separable convolution blocks: (a) regular convolution, (b) depthwise separable convolution, (c) separable with linear bottleneck, and (d) bottleneck with expansion layer as used in MobileNetV2. Diagonally hatched layers do not use non-linearities. Source: Sandler et al. (2018)

Efficiency Metrics of Mobile-First Architecture

A comparison of these architectures highlights a balance between raw accuracy and practical deployment. Take EfficientNetB0, which applies compound scaling to optimize network depth, width and resolution typically achieving higher accuracy than MobileNetV2, although with somewhat more parameters (Tan & Le, 2019). However, when memory is limited, MobileNetV2 is generally preferred due to its lower resource requirement (Sandler et al., 2018). Table 2.3 provides a comparative overview of the key achitectures considered for this project.

Architecture	Accuracy (PlantVillage)	Parameter Count	Computational Efficiency

VGG-16	~98.4%	~138 Million	Low (High Latency)
ResNet-50	~99.4%	~25 Million	Moderate
MobileNetV2	~95.8%	~3.5 Million	High (Mobile Ready)
EfficientNetB0	~99.5%	~5.3 Million	High

Table 2.3: Efficiency metrics comparing CNN architectures on PlantVillage. MobileNetV2 selected for this project. Sources: Simonyan & Zisserman (2014); He et al. (2016); Sandler et al. (2018); Tan & Le (2019).

2.2 Deep Learning Frameworks

The development of AI systems for agriculture is underpinned by the choice of deep learning frameworks, primarily TensorFlow (Google) and PyTorch (Meta) (Paszke et al., 2019; Abadi et al., 2016). This choice is not merely technical but philosophical, influencing the speed of research and the scalability of deployment.

PyTorch is the leading framework in academic research, including my BEng (Hons) studies at the University of Hertfordshire. Its dynamic computational graph uses a “define by run” approach, allowing real time debugging with standard Python tools and print statements during training. The design follows Python conventions, minimising repetitive code and easing learning especially helpful for students and researchers who need fast iteration and custom model development (Udacity, 2025).
Furthermore, PyTorch 2.x introduced *torch.compile(),* which leverages a JIT (Just -In-Time) compiler to provide significant speedups, often 20% to 25% with a single line of code (PyTorch, 2025). This makes it increasingly viable not just for research but also for performance-critical training loops. The academic community’s preference for PyTorch is reflected in the fact that major AI labs and platform such as Hugging Face almost exclusively support it for cutting-edge models like BERT and Llama.

TensorFlow and Keras

On the other hand, TensorFlow remains the go to choice for industrial applications and large-scale machine learning operations (Ba Alawi, 2025). In the past, its use of static computation graph made debugging more challenging but offered performance advantages across diverse hardware, from powerful TPUs to budget mobile devices. Although TensorFlow 2.x introduced eager execution to match PyTorch’s user-friendly development experience, its tools for model deployment are still more advanced and better established (Ba Alawi, 2025; Udacity, 2025).
TensorFlow Lite (TFLite) and TensorFlow Serving give reliable, proven deployment for Android, iOS and the cloud (Abadi et al., 2016). For agricultural businesses that need 99.9% uptime and a smooth integration with Google Cloud or Vertex AI, TensorFlow’s enterprise-grade stability is often more valuable than PyTorch’s flexibility (Ba Alawi, 2025).

Feature	**PyTorch**	**TensorFlow**
Philosophy	Research-first, intuitive, flexible	Production-first, scalable, stable

Execution	Dynamic Computation Graph	Static & Dynamic (TF 2.x)
Deployment	TorchServe, ONNX, Mobile	TFLite, TF Serving, TFX
Speed/Performance	Torch.compile() for SOTA speed	XLA optimisation for inference
Debugging	Highly intuitive	Improving but can be abstract

Table 2.4: Feature comparison of PyTorch and TensorFlow across research and deployment dimensions. Sources: Paszke et al. (2019); Abadi et al. (2016); Ba Alawi (2025).

PyTorch was selected for this project due to its superior suitability for academic research and custom model development, its native support for the pruning utilities used in Section 3.8, and its straightforward integration with Google Colab's GPU environment.

2.3 The Robustness Gap

A central critique of current research in plant pathology is the “robustness gap”, the disparity between the near-perfect accuracy in controlled settings and the significant performance degradation observed in real-world agricultural environments (Ahmad et al., 2023).

The Critique of the PlantVillage Dataset

The PlantVillage dataset is widely used as a benchmark in most published research, offering more than 54,000 images divided into 38 classes (Hughes & Salathé, 2015). However, the imaging setup is quite artificial: leaves are removed from plants and photographed against plain grey or black backgrounds under controlled conditions (Mohanty et al., 2016). As a result, model trained on this data often report very high accuracy rate, ranging from 98% to 99.9%. A notable concern specific to the dataset is class imbalance: the 38 disease categories are not equally represented in terms of sample size, which means models may perform well on the majority classes while substantially underperforming on rarer disease categories. This concern directly motivates the use of the macro-averaged F1-score as the primary evaluation metric in this project, as macro averaging treats all 38 classes equally, regardless of their sample counts.

Recent studies have identified notable issues with “capture bias” and “background bias” in these models. For instance, researchers discovered that models could reach 49% accuracy by focusing solely on background elements in the images, indicating that the networks were picking up on non-biological cues tied to the labels rather than actual disease symptoms. When these models are evaluated on real-world field images such as those in the PlantDoc dataset their performance typically declines sharply with accuracy falling from around 99% to as low as 31% (Mohanty et al., 2016). Ahmad et al. (2023), in a systematic evaluation across multiple architectures and perturbation types, confirmed that this degradation is not architecture-specific but reflects a fundamental mismatch between the controlled training distribution and real-field imaging conditions. Table 2.5 illustrates this gap across three representative architectures.

Model Architecture	Lab Accuracy (PlantVillage	Field Accuracy (PlantDoc)	Accuracy Drop (%)
ResNet50	98.18%	41.79%	56.39%
ConvNeXt	99.47%	71.76%	27.71%
SWIN (Transformer)	~99%	88.07%	~11%

Table 2.5: Laboratory vs field accuracy across CNN architectures, demonstrating the robustness gap between PlantVillage and PlantDoc dataset. Source: Varma et al. (2025).

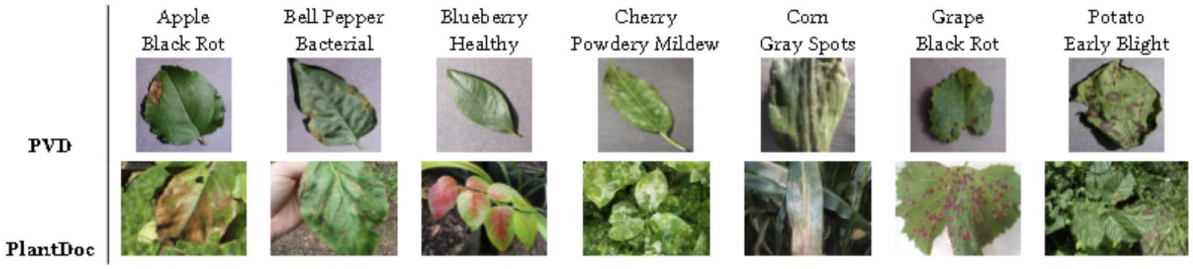

Figure 4: Samples from equivalent disease classes in PlantVillage (PVD, top row) and PlantDoc (bottom row), illustrating the visual distribution shift that causes the robustness gap. Top row shows isolated leaves against uniform controlled background; bottom row shows real-field images with complex background, variable lighting, and overlapping foliage. Source: Singh et al. (2020)

The decline in performance is linked to environmental factors, including changing natural light, shadows, camera noise, diverse backgrounds such as soil or mulch and overlapping leaves in thick canopies (Ahmad et al., 2023). Varma et al. (2025) systematically evaluated this gap across a range of architectures, including ResNet-50, ConvNeXt, and the SWIN Transformer, finding that transformer-based architectures, which attend to global image context, exhibit considerably smaller field performance drops than standard CNNs. This finding suggests that attention mechanisms may offer a more robust inductive bias for real-world agricultural imaging and represents a direction for future work in this project.

Simulating Robustness through Data Augmentation

To bridge this gap, use advanced data augmentation that simulates harsh field conditions. Inject noise and transformations into the training pipeline so models learn robust morphological features that withstand environmental factors.

- **Gaussian and Impulse Noise:** Injecting Gaussian noise from a normal distribution, $\mathrm{N}(0, \sigma^2)$ mimics sensor noise in low-cost smartphone cameras under low-light conditions (Basak, 2025). Salt and pepper noise mimics transmission errors and sensor malfunctions.
- **Colour Space Transformations:** Grayscale conversion is utilised to ensure the model does not rely on specific “chromatic shortcuts” that may vary with sunlight intensity. Colour jittering, including adjustments to hue, saturation, and brightness, prepares the model for different times of day and atmospheric conditions (Shoaib et al., 2023).

- **Geometric Variability:** Random Rotations, flipping, and zooming simulate the diverse angles and distance rom which a farmer photographs a crop (Shoaib et al., 2023).

By implementing these techniques, researchers can develop models that maintain 90%+ accuracy even in uncontrolled environments, providing a more reliable foundation for real-world decision-support systems (Basak, 2025; Shoaib et al., 2023).

2.4 Explainable AI (XAI)

The application of AI in agriculture ultimately depends on trust. Farmers and agricultural experts are often reluctant to use costly chemical treatments or remove parts of their crops based on decision made by opaque "black box" models that offer no clear explanation (Kinger & Kulkarni, 2021). Explainable AI (XAI) helps overcome this barrier by revealing the inner workings of neural networkds through transparent visual representations of how conclusions are reached (Pavithran & Punithavalli, 2024).

Theory and Mechanism of Grad-CAM

Gradient-weighted Class Activation Mapping (Grad-CAM) has become the go to explainable AI (XAI) method in plant pathology (Selvaraju et al., 2017). It works across various Convolutional Neural Networks (CNN) architectures without requiring structural changes, such as eliminating fully connected layers, a limitation of earlier approaches. Grad-CAM generates a rough localisation map by analysing the gradients of a specific target class, such as "Tomato Late Blight" as they pass through the final convolutional layer, effectively pinpointing the most relevant areas of the input image (Selvaraju et al., 2017).

The mathematical generation of a Grad-CAM heatmap involves the following steps:

- **Forward Pass:** Pass the image through the CNN to obtain the feature maps A^k from the last convolutional layer (Selvaraju et al., 2017).
- **Gradient Extraction:** Compute the gradient of the class core y^c (before softmax) with respect to the feature maps A^k, denoted as $\frac{\partial y^c}{\partial A^k}$ (Selvaraju et al., 2017).
- **Weight calculation:** Perform Global Average Pooling on the gradients to find the important weights $\propto_k^c$ for each feature map K (Selvaraju et al., 2017):

$$\propto_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

- **Heatmap Generation:** Calculated the weighted sum of the feature maps and apply a ReLU (Rectified Linear Unit) activation to focus only on positive influences that support the class (Selvaraju et al., 2017):

$$\boldsymbol{L_{Grad-CAM}^c} = \boldsymbol{ReLU}\left(\sum_{\boldsymbol{k}} \propto_k^c A^k\right)$$

The final heatmap is resized to the original image's dimensions and superimposed on it. This allows user to confirm whether the model detects "Apple Scab" by focusing leaf lesions rather than irrelevant elements such as soil or artifacts (Selvaraju et al., 2017; Pavithran & Punithavalli, 2024). Visual confirmation is essential in agricultural diagnostics as incorrect results false positives or false negatives can lead to unnecessary pesticide use or unchecked disease spread (Kinger & Kulkarni, 2021).

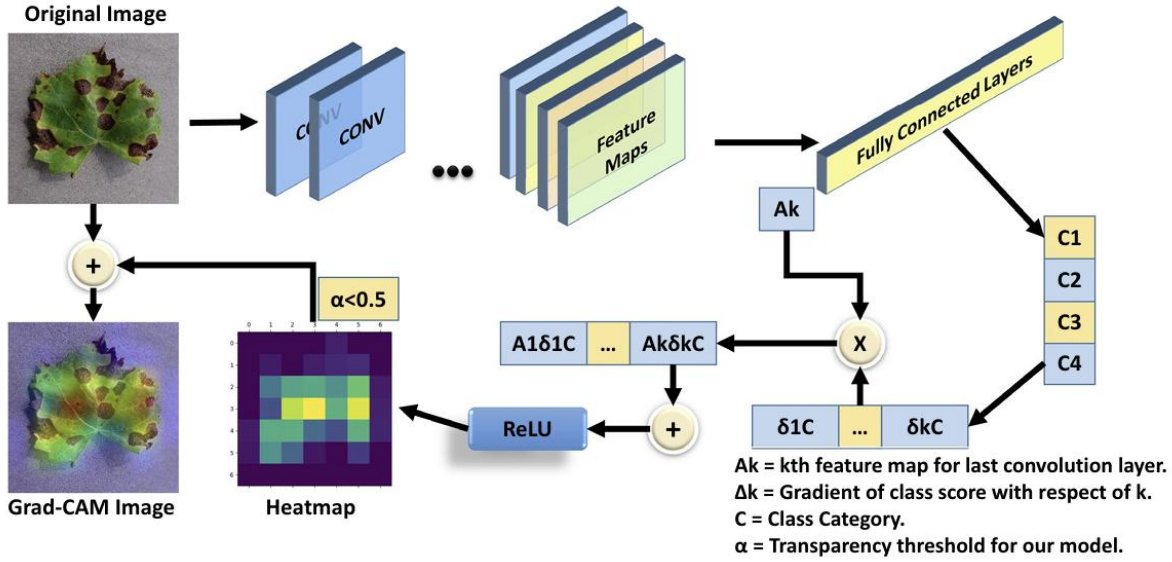

Figure 5 Grad-CAM pipeline applied to a diseased leaf image. Feature map A^k from the final convolution layer are weighted by class-specific Δk, summed, passed through ReLU, and superimposed on the original image to produce a localisation heatmap. Warm regions indicate areas most influential to the classification decision. Source: Selvaraju et al.

2.5 Model Optimisation for Edge-Based Smart Agriculture

One major obstacle to the broad adoption of smart agriculture lies in the hardware limitations of mobile and embedded devices. In remote areas, farmers typically rely on affordable smartphones that have minimal RAM, lower-end processors and limited battery life. Although models such as MobileNetV2 are designed to be lightweight, they still include redundant components that can be reduced using neural network pruning techniques (Han et al., 2015).

Theory of Neural Network Pruning

Pruning is a model compression method inspired by synaptic pruning, where unnecessary or inactive connections with zero or very small weights are removed because they do not improve the network's performance (Frankle & Carlin, 2019). This approach is theoretically supported by the Lottery Ticket Hypothesis. The Lottery Ticket Hypothesis proposes that large neural networks contain smaller subnetworks, known as "winning tickets." In this context, a subnetwork refers to a set of connections (or weights) selected from the original network. Winning tickets, when trained independently from their original initialisations, can match the accuracy of the full model (Frankle & Carlin, 2019). This provides

a principled justification for expecting that a pruned model can recover its pre-compression performance through fine-tuning.

In the context of my FYP, L1 unstructured pruning is applied to reach a sparsity level of 14.67%. This method removes individual weight based on their magnitude within the model. Specifically, it uses the L1 norm to identify and eliminate wights with the smallest absolute value, resulting in a sparse network structure (Frankle & Carlin, 2019; Buongiorno et al., 2022).
This process follows a standard pipeline:

- **Pre-training:** Train a dense MobileNetV2 model to convergence
- **Magnitude Selection:** Rank weights by heir absolute values and identify the bottom (Han et al., 2015).
- **Zeroing:** Apply a pruning mask to set these weights to zero
- **Fine-tuning:** Re-train the sparse model for several epochs to allow the remaining weights to compensate for the removed connections, effectively recovering lost accuracy (Frankle & Carlin, 2019).

Latency vs Model Size Trade-offs

A critical distinction in pruning is the difference between unstructured and structured approaches. Unstructured pruning, the method used in this project, leads to high compression ratios, significantly reducing model storage size, but often yields minimal latency improvements on standard hardware. This is because dense matrix-multiplication kernels on conventional CPUs and GPUs continue to process zeroed values rather than skipping them (Buongiorno et al., 2022). Conversely, structured pruning, which removes entire filters or channels, reduces the computational graph and produces real-world inference speedups, but typically incurs greater accuracy degradation (Buongiorno et al., 2022). Table 2.6 compares the key optimisation techniques relevant to edge deployment.

Optimisation Technique	**Mechanism**	**Primary Benefit**	**Hardware Support**
Depthwise Separable Conv	Algorithmic factorization	FLOP reduction	Excellent (All mobile CPUs)
L1 Unstructured Pruning	Magnitude-based zeroing	Reduced storage size	Requires sparse kernels for speed
Structured Pruning	Filter/Channel removal	Fater inference (Latency)	Native support in dense libraries
Post-Training Quantization	Reducing bit-precision (FP32 to INT8)	Memory footprint/Speed	NPU/DSP optimized

Table 2.6: Comparison of model optimisation techniques relevant to edge-based smart agriculture deployment. L1 Unstructured Pruning is the approach applied in this project. Sources: Han et al. (2015); Frankle & Carlin (2019); Buongiorno et al. (2022).

For a field-ready smart agriculture system, a combination of lightweight architecture design and magnitude-based pruning ensures that the diagnostic tool remains accessible and energy-efficient for smallholder farmers.

Conclusion

The literature reviewed in this chapter identifies three persistent and interconnected gaps that collectively prevent deep learning plant disease classifiers from being deployed as trustworthy, field-ready diagnostic tools. First, the robustness gap demonstrates that high laboratory accuracy does not translate to reliable field performance in the absence of deliberate augmentation strategies. Second, the parameter-efficiency constraint highlights that state-of-the-art accuracy is typically achieved at architectural scales incompatible with edge hardware. Third, the interpretability deficit reveals that opaque predictions erode stakeholder trust even when numerical performance is high.

No single published work has addressed all three gaps simultaneously within a single deployment-ready pipeline. This project is designed to bridge this gap by combining Gaussian noise and grayscale augmentation to improve robustness, Grad-CAM to establish spatial interpretability, and layer-wise L1 unstructured pruning with fine-tuning to achieve edge compatibility while monitoring all three dimensions of performance throughout the pipeline.

3. Methodology and Implementation

The methodology follows a sequential and comparative design. It begins with dataset acquisition and integrity screening, followed by the construction of a robustness-oriented preprocessing pipeline. Two model architectures are trained under identical conditions: a custom baseline CNN and a transfer-learning-based MobileNetV2. The superior architecture MobileNetV2 is then subjected to layer-wise L1 unstructured pruning. In the final stage, the pruned model undergoes fine-tuning, evaluation, and Grad-CAM interpretation, then is exported as a deployable state dictionary. This sequential structure ensures that each stage informs the next and that design decisions are grounded in empirical observation rather than assumption.

Figure 6 End-to-end methodological pipeline

Computational environment and reproducibility controls

Google Drive serves as the storage backend for exported model checkpoints and the compressed dataset in the Google Colab implementation. Although this environment provides GPU acceleration and persistent cloud storage, it also presents risks, including session resets and hidden state. Reproducibility was therefore treated as a first-class methodological requirement. The global random seed was set to 42 across Python, NumPy, and PyTorch; cuDNN deterministic mode was enabled; and a seeded DataLoader generator was used to eliminate shuffle non-determinism. Table 3.1 summarises these controls and their methodological justifications.

Component	Notebook implementation	Methodological purpose
Execution environment	Google Colab + mounted Google Drive	GPU-enabled experimentation with persistent storage
Core libraries	PyTorch, torchvision, scikit-learn, matplotlib	Deep learning, splitting, and visual analytics
Seed policy	SEED = 42 across Python, NumPy, Torch	Reduce random variation across runs
cnDNN configuration	Deterministic = True Benchmark = False	Improve consistency of convolution behaviour
DataLoader reproducibility	Seeded torch.Generator + worker_init_fn	Remove shuffle non-determinism
Data splitting	Train/validation/test partition	Protect class balance and fair comparison

Table 3.1 Computational Environment and reproducibility controls

Code Snippet 3.1 Reproducibility control

```
# GLOBAL SEED (ensures reproducibility across all runs)
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

3.1 Dataset definition, integrity screening, and sampling frame

PlantVillage dataset

The study uses the PlantVillage colour dataset, containing 54,305 RGB images of plant leaves organised into 38 distinct labelled categories. PlantVillage is a prominent benchmark in plant disease classification research due to its clear class annotations, large scale, and extensive use in prior studies. Its selection here supports the methodological goal of comparing different modelling approaches under uniform conditions. As noted in Section 2.3, the dataset has a well-documented limitation: images were captured in controlled settings with uncluttered backgrounds, and the 38 disease categories are not equally represented in terms of sample count. This class imbalance necessitates the use of macro-averaged F1-score as the primary evaluation metric, as macro averaging treats all classes equally regardless of sample frequency, preventing artificially inflated performance figures driven by majority-class accuracy. The preprocessing pipeline addresses the robustness limitation by partially simulating real-world imaging conditions through grayscale conversion and Gaussian noise injection.

Integrity screening

Before constructing the PyTorch training datasets, the notebook performs image-level integrity validation. The dataset is first reorganised into a binary folder structure (Healthy/Diseased) to support class-count inspection, and the integrity check is performed at this stage. Each image file is opened using PIL's context-managed interface and passed through *img.verify()*; any file that raises an *IOError*, *OSError*, or *Image.UnidentifiedImageError* is flagged and removed. This validation step is methodologically important because corrupt files can distort class counts, break data loaders during training, or introduce non-reproducible failures that are difficult to diagnose after the fact.

It should be noted that the formal *verify()* call operates over the binary reorganisation of the data rather than being repeated over the 38-class *ImageFolder* used for model training. The training pipeline, therefore, relies on torchvision to raise load-time errors for any remaining corrupt files encountered at runtime. For a production-grade implementation, the integrity walk should also be applied directly to the colour split before ImageFolder construction.

Code Snippet 3.2. Image-integrity validation

```
def is_image_ok(path):
    """Returns True if the image can be opened and verified, False otherwise."""
    try:
        with Image.open(path) as img:
            img.verify()
        return True
    except (IOError, OSError, Image.UnidentifiedImageError) as e:
        print(f"  Corrupt image detected: {path} — {e}")
        return False

bad_files = []

for label in ["Healthy", "Diseased"]:
    label_dir = os.path.join(BINARY_ROOT, label)
    for fname in os.listdir(label_dir):
        if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
            continue
        fpath = os.path.join(label_dir, fname)
        if not is_image_ok(fpath):
            bad_files.append(fpath)

print(f"Found {len(bad_files)} corrupt images")

for f in bad_files:
    os.remove(f)

print("Removed corrupt images.")
```

3.2 Preprocessing pipeline and mathematical formulation

The preprocessing pipeline was designed to satisfy two competing objectives. The first was compatibility with ImageNet-pretrained transfer learning, which requires a consistent input size and an input distribution aligned with the source model. The second was to introduce a limited form of robustness engineering so that the classifier would not rely entirely on the clean colour characteristics of PlantVillage. The final transform chain therefore contains resizing, grayscale conversion replicated over three channels, tensor conversion, additive Gaussian noise with output clamping, and ImageNet-style normalisation. Table 3.2 describes each transform in detail.

Transform	Purpose	Configuration
Resize((224, 224))	Standardise spatial dimensions for both networks	224 x 224 pixels

Grayscale(num_output_channels=3)	Reduce chromatic dependence; preserve tensor shape	3-channel luminance replication
ToTensor()	Convert PIL images to float tensors	[0, 255] → [0, 1]
AddGaussianNoise(mean=0, std=0.05)	Inject sensor-style noise; output clamped to [0, 1]	AWGN, clipped to valid range
Normalize(mean, std)	Align with ImageNet pre-training statistics	$\mu = [0.485,0.456,0.406]$, $\sigma = [0.229,0.224,0.225]$

Table 3.2. Preprocessing transform pipeline with purpose and configuration

The grayscale stage follows the standard luminance weighting of Equation 3.1, offering a defensible compromise between maintaining structural leaf information and reducing dependence on colour cues, which can vary across camera types, white balance settings, or illumination conditions. Subsequently, the noise-injection stage perturbs each tensor with additive white Gaussian noise (Equation 3.2); afterwards, the result is clipped to the valid [0, 1] float range before normalisation (Equation 3.3).

$$Y = 0.299R + 0.587G + 0.114B$$

(Eq. 3.1 – ITU-R BT.601 standard luminance weighting)

$$x_{noisy} = clip(x + \varepsilon, 0,1),\ \varepsilon \sim N(0, \sigma^2),\ \sigma = 0.05$$

(Eq. 3.2 – additive white Gaussian noise with output clamping)

$$X_c' = \frac{X_c - \mu_c}{\sigma_c}$$

(Eq. 3.3 – per-channel ImageNet normalisation)

Code Snippet 3.3. Robust Preprocessing pipeline

```
class AddGaussianNoise(object):
    """
    Injects random Gaussian noise into a tensor image to simulate
    real-world sensor noise and improve model robustness.
    """
    def __init__(self, mean=0., std=0.1):
        self.std = std
        self.mean = mean

    def __call__(self, tensor):
        noise = torch.randn(tensor.size()) * self.std + self.mean
        return torch.clamp(tensor + noise, 0., 1.)  # Clamp to valid range

    def __repr__(self):
        return f"{self.__class__.__name__}(mean={self.mean}, std={self.std})"


# STANDARD TRANSFORMS (used for Baseline CNN comparison)
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

This preprocessing approach is more robust than a simple resize-and-normalize method because it reflects real-world deployment conditions: a model designed for smart agriculture should be trained on data that includes minor sensor degradation, rather than only pristine, benchmark-quality images. While converting images to grayscale may discard useful colour-based diagnostic information, this trade-off is accepted in favour of building resilience aligned with actual field conditions, rather than relying on idealised, colour-rich lab imagery. The deliberate use of clamping in AddGaussianNoise ensures that pixel values remain within the [0,1] range required by the subsequent normalisation step, maintaining consistency with ImageNet-based pre-training statistics.

3.3 Stratified partitioning and data-loader construction

The dataset is split into two stages to produce 70% training, 15% validation and 15% test subsets. Stratification is applied at both stages using *scikit-learns's train_test_split* with *stratify=targets* ensuring class proportions are preserved across all three partitions. This is especially important in a 38-class problem because even a modest imbalance can lead to under-representation of minority classes if the split is purely random. This resulted in 38013 training images, 8146 validation images and 8146 test images

Using a dedicated validation set allows optimisation decisions, specifically early stopping and learning rate reduction, to be made without contaminating the final test estimate. The held-out test set is therefore reserved for final reporting only, which is the correct separation for this project experimental design.

Code Snippet 3.4. Two-stage stratified split

```
targets = full_dataset.targets
indices = list(range(len(full_dataset)))

train_idx, temp_idx, train_targets, temp_targets = train_test_split(
    indices, targets, test_size=0.30, random_state=SEED, stratify=targets
)
val_idx, test_idx = train_test_split(
    temp_idx, test_size=0.50, random_state=SEED, stratify=temp_targets
)
```

3.4 Model architectures and Comparative Logic

Custom Baseline CNN

A custom baseline CNN was developed to serve as a controlled benchmark for evaluating the transfer-learning model. The architecture comprises three convolutional blocks, each with batch normalisation and max pooling, followed by a fully connected classifier with 50% dropout. The large, flattened feature map (128 × 28 × 28) feeding into the first dense layer results in approximately 51.5 million trainable parameters, roughly 14.7 times MobileNetV2's 3.5 million. While this scale makes the baseline CNN impractical for edge deployment, its inclusion in the study is methodologically justified: it ensures that MobileNetV2's advantages are empirically validated rather than assumed and provides a direct measurement of the performance cost of architectural simplicity.

Code Snippet 3.5. Baseline CNN architecture

```
class BaselineCNN(nn.Module):
    def __init__(self, num_classes=38):
        super(BaselineCNN, self).__init__()

        # Feature Extraction Pathway
        # Input shape: [Batch_Size, 3 channels, 224, 224]

        # Block 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # Output size: 112x112

        # Block 2
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # Output size: 56x56

        # Block 3
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # Output size: 28x28

        # Classification Pathway
        # We flatten the 128 channels of 28x28 spatial dimensions
        self.fc1 = nn.Linear(128 * 28 * 28, 512)
        self.dropout = nn.Dropout(p=0.5) # Helps prevent overfitting
        self.fc2 = nn.Linear(512, num_classes) # Final output layer for 38 classes
```

MobileNetV2 with Transfer Learning

MobileNetV2 was selected as the primary architecture due to its inverted residual blocks and depthwise separable convolutions, which substantially reduce the multiply-accumulate cost relative to conventional CNNs while maintaining competitive accuracy. The ImageNet-pretrained backbone was loaded, and all backbone parameters were initially frozen (*requires_grad=False*). The final classification layer was replaced with a new 38-class linear head trained from scratch (*requires_grad=True*). This transfer learning strategy allows the model to leverage generic visual features learned from ImageNet at scale while adapting the classification boundary to the plant disease domain.

An important methodological note: prior to the Grad-CAM visualisation step, all MobileNetV2 parameters were unfrozen (*requires_grad=True*) to enable gradient flow through the convolutional backbone, as required by Grad-CAM. This unfreezing persists into the pruning and fine-tuning stages. The Grad-CAM analysis is therefore conducted on a fully unfrozen model, which is consistent with the parameter state used throughout the remainder of the pipeline. This detail is disclosed in the interest of methodological transparency.

Code Snippet 3.6. MobileNetV2 transfer-learning setup

```
mobilenet_model = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.DEFAULT)

# Freeze the base layers (Feature Extractor)
for param in mobilenet_model.parameters():
    param.requires_grad = False
# Replace the final classification head
num_features = mobilenet_model.classifier[1].in_features
mobilenet_model.classifier[1] = nn.Linear(num_features, 38)
```

3.5 Training Protocol and Optimisation Strategy

Both models are trained under a controlled optimisation framework, so that differences in outcomes can be attributed primarily to architecture rather than to inconsistent training practices. Cross-entropy loss is used for the multi-class objective because each image belongs to exactly one of the 38 classes. The loss is defined in Equation 3.4, where the target label y is represented as a one-hot vector, and the model outputs a probability distribution over the class set.

$$\mathcal{L}_{CE} = -\sum_{k=1}^{K} y_k \log(\hat{p_k})$$

(Eq. 3.4 – categorical cross-entropy loss over K = 38 classes)

The Adam optimiser was paired with a ReduceLROnPlateau scheduler (factor = 0.5, patience = 3) and early stopping (patience = 5 epochs). If validation accuracy does not improve for five consecutive

epochs, training is terminated and the checkpoint with the best validation accuracy is reloaded. Table 3.3 summarises all training settings and their methodological roles.

Training setting	Represented in the code	Methodological role
Loss function	CrossEntropyLoss	Standard objective for single-label multi-class classification
Optimiser	Adam	Fast adaptive optimisation
Initial LR (both models)	0.001	Comparable learning-rate starting point
Max training epochs	20	Upper limit rather than mandatory duration
Early stopping patience	5 epochs (no improvement in val accuracy)	Stop when validation improvement
LR scheduler	ReduceLROnPlateau, facor-0.5, patience=3	Reduce LR when val loss plateaus for 3
Fine-tune LR after pruning	0.0001	Conservative recovery learning rate
Fine-tune scheduler patience	2 epochs	Tighter control for short recover window
Fine-tune epochs	5 (max)	Short recovery stage after compression

Table 3.3. Training settings and their methodological roles

The conservative fine-tuning learning rate of 0.0001 was selected specifically to prevent destabilisation of the retained weight structure after pruning. Because the fine-tuning phase aims to re-optimise surviving weights around the pruned topology rather than to learn new feature representations, a learning rate ten times smaller than the initial training rate provides the precise, localised weight updates required for recovery without disrupting the functional connections that pruning preserved.

Code Snippet 3.7. Training function with scheduler and early stopping

```
def train_model(model, train_loader, val_loader, criterion, optimizer,
                num_epochs=10, patience=5, scheduler=None):

    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    epochs_no_improve = 0  # Early stopping counter

    history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}
```

```
            # LR SCHEDULER STEP (based on val loss)
            if scheduler is not None:
                scheduler.step(epoch_loss)
                current_lr = optimizer.param_groups[0]['lr']
                print(f'  Current LR: {current_lr:.6f}')

            # BEST MODEL CHECKPOINT
            if epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
                epochs_no_improve = 0
            else:
                epochs_no_improve += 1

    # EARLY STOPPING CHECK
    if epochs_no_improve >= patience:
        print(f'\n Early stopping triggered after {epoch+1} epochs '
              f'(no improvement for {patience} consecutive epochs).')
        break
```

3.6 Evaluation Framework and Decision Criteria

The evaluation strategy combines overall accuracy, class-level diagnostic metrics, and visual error analysis. Accuracy is reported as the most interpretable summary statistic for the 38-class task. However, because accuracy fails to reveal performance imbalances across classes and because the PlantVillage dataset exhibits class imbalance, it is not used as the sole measure. Precision, recall, and F1-score are computed per class (Equation 3.5). Macro-averaged F1-score is selected as the primary evaluation metric because it weights all 38 classes equally, making it sensitive to poor performance on rare disease categories that would otherwise be masked by strong majority-class results.

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}, \qquad \boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}, \qquad \boldsymbol{F1} = \frac{\boldsymbol{2PR}}{\boldsymbol{P} + \boldsymbol{R}}$$

(Eq. 3.5 – per-class precision, recall and F1-score)

Confusion matrices are generated in both raw-count and row-normalised forms. The normalised matrix provides per-class recall for all 38 classes and reveals whether misclassifications are scattered across dissimilar disease categories or clustered among visually similar ones, a distinction with direct agronomic implications. A full scikit-learn classification report is also generated to allow inspection of minority-class behaviour. Table 3.4 summarises the evaluation outcomes across all model states in the pipeline.

Model state	Best val. Acc.	Test acc.	Sparsity	Interpretation
Baseline CNN	82.68%	82.97%	0%	Useful benchmark; too parameter-heavy for edge
MobileNetV2 (pre-pruning)	~87.8%	88.31%	0%	Best compact model before compression
MobileNetV2 after 15% layer-wise prune	----	60.58%	14.67%	Compression disrupts decision surface; recovery required
Pruned MobileNetV2 after fine-tuning	97.30%	97.56%	14.67%	Strong recovered performance; interpret cautiously (single run)

Table 3.4. Model state comparison across all pipeline stages. The post-fine-tuning result of 97.56% is based on a single run and single split, and should be interpreted as a strong observed outcome rather than a generalised population estimate.

3.7 Explainability through Grad-CAM

Explainability is incorporated to test whether the selected model grounds its decisions in plausible leaf regions rather than spurious image artefacts. The implementation uses Grad-CAM, which backpropagates class-specific gradients to the final convolutional layer and combines them into a weighted activation map (Equation 3.6).

$$L^{c}_{Grad-CAM} = \boldsymbol{ReLU}\left(\sum_{k} \propto^{c}_{k} A^{k}\right)$$

(Eq. 3.6 – Grad-CAM class-discriminative localisation map)

This step is important because high accuracy alone does not verify that the classifier has identified meaningful agronomic features. While saliency visualisation cannot prove causal reasoning, it can expose cases where the model attends to the wrong regions, such as image corners, background soil, or uniform lighting artefacts. Such cases would render the model unusable in practice, regardless of its numerical accuracy.

A necessary implementation note: Grad-CAM requires gradient flow through the convolutional backbone, so code snippet 3.8 unfreezes all MobileNetV2 parameters (for param in *mobilenet_model.parameters(): param.requires_grad = True*) before computing saliency maps. This unfreeze persists into the pruning and fine-tuning stages. The Grad-CAM analysis is therefore

conducted on a fully unfrozen model, which is consistent with how the model is subsequently compressed and recovered.

Code Snippet 3.8. Grad-CAM setup

```
# Unfreeze the layers so Grad-CAM can calculate the backward gradients
for param in mobilenet_model.parameters():
    param.requires_grad = True

# make sure the target layer is properly set to the last Conv block
mobilenet_target_layer = [mobilenet_model.features[-1]]

# Re-initialize the GradCAM object now that the model can track gradients
cam_mobile = GradCAM(model=mobilenet_model, target_layers=mobilenet_target_layer)
```

3.8 Pruning Methodology and Sparsity Control

Model compression was implemented using *torch.nn.utils.prune* for L1 unstructured pruning, rather than quantisation or knowledge distillation. The methodological justification is that zeroing small-magnitude weights reduces the effective parameter count while preserving a recoverable functional core, since low-magnitude weights contribute least to the final decision function (Equation 3.7).

$$w_i \leftarrow 0 \quad for\ the\ lowest\ |w_i|\ under\ layer-wise\ L1\ pruning$$

(Eq. 3.7 L1 unstructured pruning rule)

The pruning policy is not globally indiscriminate. It is applied layer-wise to eligible convolutional and linear layers, while deliberately skipping the first convolution and the final classifier. The first convolution frequently captures low-level visual primitives that are broadly useful and difficult to recover if removed; the final classifier directly encodes the 38-way decision boundary and is therefore especially sensitive to weight removal. Protecting these layers makes the pruning stage more controlled and easier to justify methodologically.

Code Snippet 3.9 Layer-wise L1 pruning

```
def layer_wise_prune_model(model, prune_amount=0.15):
    print(f" Starting Layer-Wise Pruning Process ({prune_amount * 100}% per layer)...")

    modules = list(model.modules())
    first_conv_found = False
    last_linear_layer = None
```

```
# Find the very last linear layer
for module in reversed(modules):
     if isinstance(module, torch.nn.Linear):
         last_linear_layer = module
         break

for module in modules:
    # Prune Conv2d layers, but skip the first one
    if isinstance(module, torch.nn.Conv2d):
        if not first_conv_found:
            first_conv_found = True
            continue # Skip the first layer

        prune.l1_unstructured(module, name='weight', amount=prune_amount)
        prune.remove(module, 'weight')

    # Prune Linear layers, but skip the final classifier
    elif isinstance(module, torch.nn.Linear):
        if module == last_linear_layer:
            continue # Skip the final layer

        prune.l1_unstructured(module, name='weight', amount=prune_amount)
        prune.remove(module, 'weight')
```

The code reports that 328,329 of 2,238,400 eligible weights were removed, resulting in 14.67% sparsity in the pruned subset. Sparsity is calculated as in Equation 3.8. Importantly, the immediate post-pruning test accuracy fell to 60.58%, indicating that pruning meaningfully disrupted the original decision surface. This is not a failure of the methodology; on the contrary, it demonstrates why a recovery stage is necessary if pruning is to be used responsibly.

$$Sparsity = \frac{N_{pruned}}{N_{eligible}} \times 100$$

(Eq. 3.8 proportion of eligible weights set to zero)

3.9 Recovery fine-tuning, export, and deployment readiness

After pruning, the model enters a short recovery phase using Adam with a reduced learning rate of 0.0001, a *ReduceLROnPlateau scheduler* (*factor=0.5, patience=2*), early stopping (*patience=5*), and a maximum of five epochs. The lower learning rate is appropriate because the objective is no longer to discover a new feature representation from scratch, but to re-optimise the surviving weights around the pruned topology. In other words, fine-tuning heals the representational damage introduced by pruning without destabilising the retained structure.

Code Snippet 3.10 Recovery fine-tuning

```
optimizer_pruned = optim.Adam(pruned_mobilenet.parameters(), lr=0.0001)

# Scheduler with tighter patience for the short fine-tune window
scheduler_pruned = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer_pruned, mode='min', factor=0.5, patience=2
)

pruned_mobilenet, pruned_history = train_model(
    pruned_mobilenet, train_loader, val_loader, criterion, optimizer_pruned,
    num_epochs=5, patience=5, scheduler=scheduler_pruned
)
```

The model not only recovered from pruning but also significantly exceeded the previous unpruned version on the code's particular split, according to the reported fine-tuned test accuracy of 97.56%. A conservative interpretation is that the network was able to settle into a better local optimum during the recovery phase, and pruning may have served as a kind of regularisation. Until this interpretation is confirmed across several seeds, splits, or an external field-oriented dataset, it should be considered speculative.

The final model is saved as a PyTorch state dictionary and stored on Google Drive. This approach is methodologically significant because the study goes beyond generating a temporary result in Python code. Instead, it delivers a reusable artifact that can be reloaded for further analysis, such as evaluating latency and memory usage, and eventually incorporated into embedded or mobile inference systems.

Code Snippet 3.11 Model export

```
# Define the save path in my mounted Google Drive
SAVE_DIR = "/content/drive/MyDrive/FYP/models"
os.makedirs(SAVE_DIR, exist_ok=True)

SAVE_PATH = os.path.join(SAVE_DIR, "optimized_mobilenetv2_38classes.pth")

# Save the model's internal state (weights and biases)
torch.save(pruned_mobilenet.state_dict(), SAVE_PATH)
```

4. Results and Critical Analysis

Using the held-out test set, training histories, confusion matrices, class activation maps, and the comprehensive precision–recall–F1 classification reports generated by the Python code, this chapter assesses the performance of the constructed plant disease classifiers in the following sequence: first, how well the models learnt during initial training; second, whether the taught features were biologically plausible; third, how much diagnostic integrity was compromised by pruning; and finally, whether fine-tuning could restore performance to a deployment-ready level. Outcomes are evaluated in terms of class balance, failure concentration, model confidence structure, and the real-world dangers of over-claiming performance, rather than just reporting correctness.

4.1 Benchmarking Initial Training Performance

The initial comparison focused on the custom Baseline CNN and the transfer-learning-based MobileNetV2 model. As shown in the training curves in Figure 4.1, the two models diverged early. The Baseline CNN began with modest performance but improved steadily, ultimately achieving a peak validation accuracy of 82.68% and a final test accuracy of 82.97%. In contrast, MobileNetV2 started at a higher performance level, converged faster, and reached a test accuracy of 88.31%. This performance gap aligns with the inherent benefits of transfer learning: by leveraging a pre-trained backbone, MobileNetV2 inherits effective low- and mid-level visual features, giving it a more advantageous starting point compared to a network with randomly initialized weights.

The loss trajectories reinforce this reading. The Baseline CNN showed a long descent from high training and validation loss, suggesting that a large fraction of the early training budget was spent learning generic image structure rather than class-discriminative features. MobileNetV2 instead stabilised around a substantially lower loss regime within a few epochs not by memorising the training set faster, but by extracting more discriminative representations with greater optimisation efficiency. This justified selecting MobileNetV2 as the primary candidate for explainability, compression, and deployment experiments.

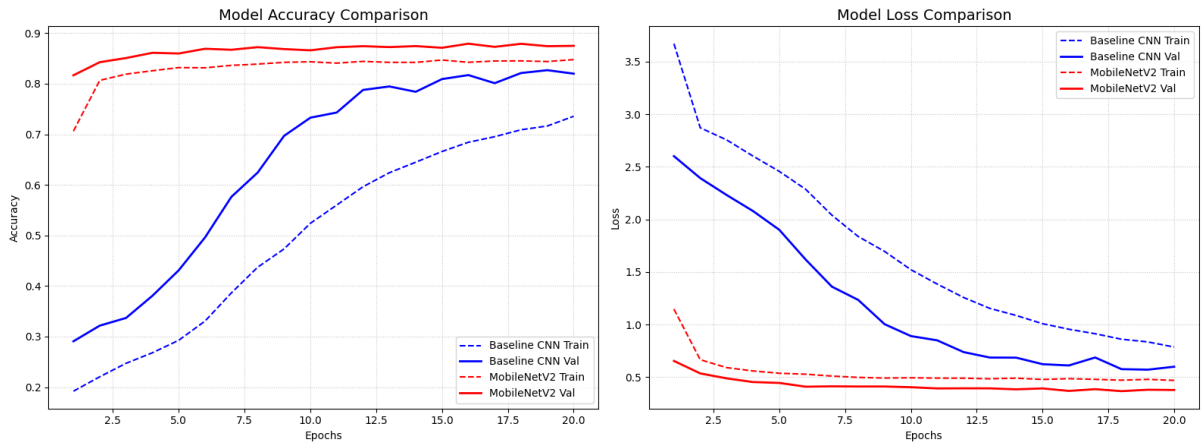

Figure 7. Training and validation accuracy/loss curves for the Baseline CNN and MobileNetV2 over 20 epochs. Source: Python code output

Model	Best Val. Acc.	Final test Acc.	Interpretation	
Baseline CNN	82.68%	82.97	Learns steadily but remains capacity-limited relative to the transfer model	Reference architecture
MobileNetV2 (original)	87.92%	88.31%	Higher starting point, better convergence, stronger generalisation.	Primary model before pruning
MobileNetV2 (pruned, pre-healing)	---	60.58%	Large degradation immediately after weight removal	Compression stress test
MobileNetV2 (pruned + fine-tuned)	97.30%	97.56%	Recovered strongly after short adaptation phase	Final deployed model

Table 4.1. Summary of headline performance across the four evaluation stages

4.2 Spatial Validation via Class Activation Mapping

Quantitative accuracy is necessary but not sufficient for an agricultural classifier, because a model can achieve strong benchmark scores while attending to irrelevant background patterns that would not generalise to field conditions. Grad-CAM visualisations were therefore used as a spatial validation tool, targeting the final convolutional block of each model and using *ClassifierOutputTarget* so saliency reflects the model's actual prediction rather than a supplied label.

MobileNetV2 – Strawberry healthy

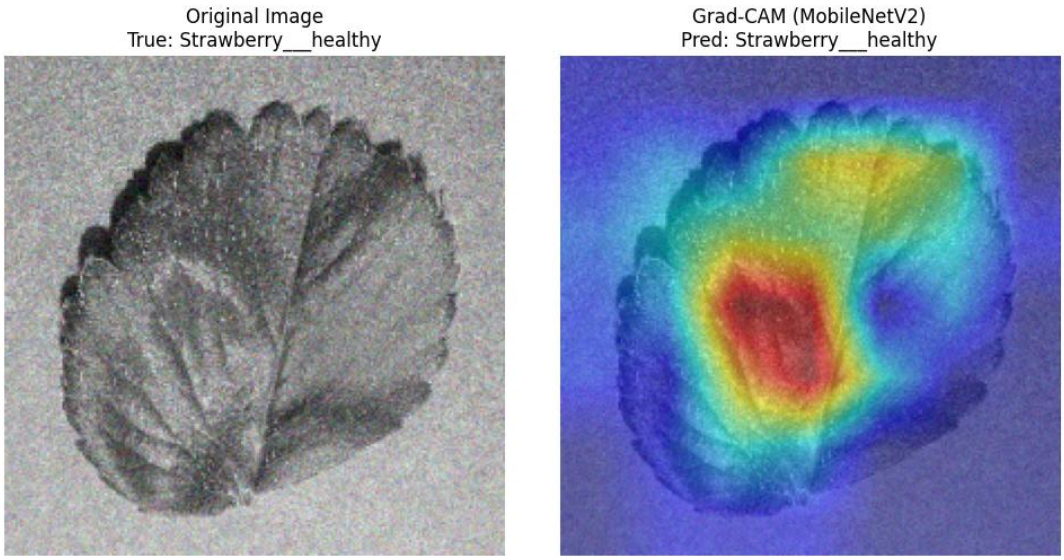

Figure 8: MobileNetV2 Grad-CAM on a correctly predicted Strawberry (healthy) leaf. Source: Python code output

In the illustrated healthy strawberry sample, the MobileNetV2 heatmap mainly focuses on the leaf blade and central venation, with no meaningful activation in the background. This provides spatial validation. The network bases its decision on plant morphology and texture, not on acquisition artefacts like image framing or lighting gradients. The activation is smoothly distributed across the leaf surface, which suits healthy-class classification, as the key signal is the absence of lesion patterns rather than any localized feature.

Baseline CNN – Squash Powdery Mildew

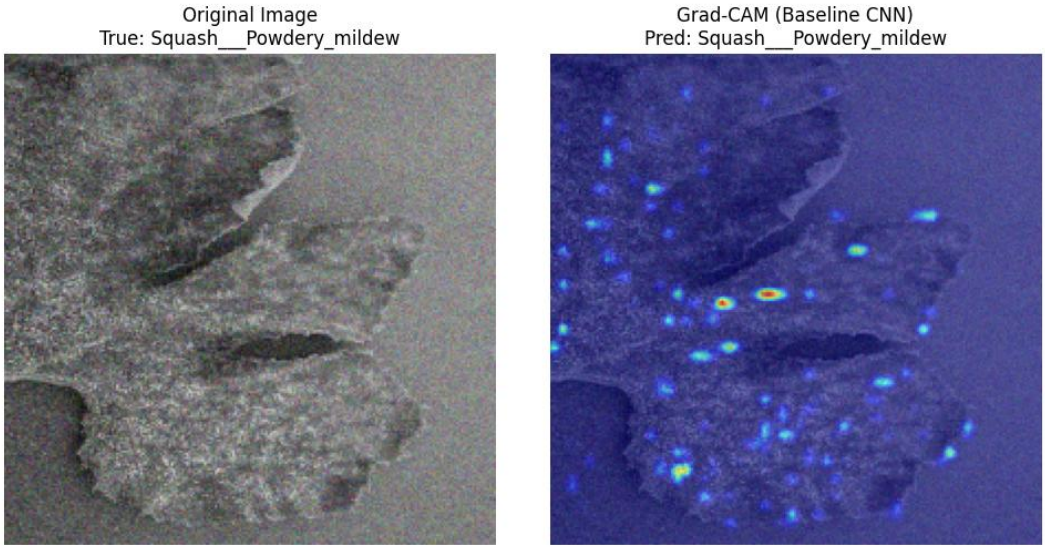

Figure 9 Baseline CNN Grad-CAM on a correctly predicted Squash Powdery Mildew (diseased) leaf. Source: Python code output

This model produces a distinctly different pattern: sparse, isolated hotspots scattered across the leaf with no overall spatial structure. Although the model predicts Squash Powdery Mildew, it appears to focus on local surface texture irregularities and noise rather than the distinctive white fungal colonies. This fragmented saliency suggests the model's representation is fragile and may not generalise to other imaging conditions.

Comparative finding: MobileNetV2 produces smooth, anatomically coherent saliency maps that are concentrated on leaf tissue. The Baseline CNN produces sparse, punctate hotspots on texture artefacts. This qualitative difference supports MobileNetV2 as the preferred architecture for deployment-aware classification where interpretability is an evaluation criterion.

4.3 Impact of Compression on Model Integrity

The compression experiment tested whether MobileNetV2 could be made lighter without destroying its diagnostic capability. The immediate answer is that pruning caused substantial short-term damage. After layer-wise L1 unstructured pruning at 15% per eligible layer, test accuracy dropped from 88.31% to 60.58%, a 27.73 percentage-point drop. This result is analytically important: it confirms that the

pruned weights were genuinely load-bearing rather than redundant. Sparsification is not a free optimisation; removing weights disrupts feature hierarchies that support class separation. The pruned model at this stage was smaller but not yet agriculturally reliable.

This degradation should not be treated as a failure of the pruning strategy. Rather, it demonstrates that structural compression and representational recovery are two distinct phases. The pre-healing model was evaluated precisely to expose this trade-off. A rigorous edge-deployment study must acknowledge this dip openly: a compressed model can appear attractive in storage terms while becoming unacceptable in decision quality without a recovery stage.

Stage	Accuracy	Absolute change	Critical reading
Original MobileNetV2	88.31%	---	Reference point before compression
Immediately pruned (pre-heal)	60.58%	-27.73 pp	Compression severely damages discriminative capacity
Fine-tuned pruned model	97.56%	+36.98 pp from pruned: +9.25 pp over original	Full recovery plus generalisation improvement

Table 4.2. Accuracy movement across pruning and recovery stages

4.3.1 Sparsity Evaluation and Pre-Healing Degradation

Pruning metric	Value
Total eligible weights evaluated	2238400
Weights pruned (set to zero)	328329
Achieved sparsity	14.67%
Pre-pruning test accuracy	88.31%
Immediate post-pruning test accuracy	60.58%
Accuracy drop from pruning	27.73 percentage points

Table 4.3. Sparsity report. Source: Python code output

The large pre-healing drop suggests that the pruned network was initially under-adapted rather than fundamentally unusable. L1 unstructured pruning removes parameters that appear individually unimportant by magnitude, but their collective removal distorts downstream activation patterns. This typically manifests as confused inter-class boundaries, particularly in visually similar disease families such as the tomato disorder cluster. The 60.58% test accuracy confirms that the decision surface had become unstable which is precisely why any claim that pruning alone improves deployment readiness would be misleading.

Critical note: The 27.73pp drop to 60.58% is evidence of methodological rigour, not failure. A pruning step with negligible accuracy loss would indicate pure redundancy rather than effective compression. This disruption magnitude demonstrates why recovery fine-tuning is an essential not optional component of the compression pipeline.

4.4 Fine-Tuning and Diagnostic Recovery

The recovery phase is the strongest empirical result in the notebook. After only five epochs of fine-tuning at lr=0.0001 with a *ReduceLROnPlateau* scheduler (factor=0.5, patience=2), the pruned MobileNetV2 reached 97.56% test accuracy and 97.30% best validation accuracy. The most plausible interpretation is that fine-tuning allowed the surviving weights to reorganise around the new sparse topology, restoring lost discriminative information while preserving the efficiency gains of compression.

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	0.2782	90.72%	0.1602	94.61%
2	0.1576	94.64%	0.1249	95.76%
3	0.1076	96.33%	0.0900	96.93%
4	0.0774	97.27%	0.0857	96.96%
5	**0.0612**	**97.95%**	**0.0790**	**97.30%**

Table 4.4. Recovery fine-tuning epoch log. Source: Python Code output

The recovery was rapid. In epoch 1, validation accuracy rebounded from 60.58% post-pruning to 94.71%, an increase of 34.13pp. By epoch 3 (96.93%), the model exceeded the pre-pruning baseline (87.92%). The conservative learning rate (0.0001) enabled precise adjustments while retaining 85.33% of high-magnitude weights, providing a strong base that needed modest updates. The compressed parameter space likely served as implicit regularisation, pushing surviving weights to encode more generalisable features, an established pruning-as-regularisation effect.

However, the scale of the improvement requires critical caution. A recovered pruned model that outperforms the original dense model by more than 9 percentage points is possible but should not be interpreted as definitive proof that pruning universally improves generalisation. Because the experiment relies on a single data split and a single recovery run, the results are best presented as strong evidence in this Python code rather than a statistically settled conclusion.

Recovery Findings: 97.56% test accuracy with 14.67% sparsity surpassing the unpruned 88.31% baseline by 9.25pp. Pruning appears to have acted as structured regularisation. This result should be confirmed over multiple seeds and splits before deployment-grade performance claims are finalised.

4.5 Comparative Analysis of Confusion Matrices and Class Fairness

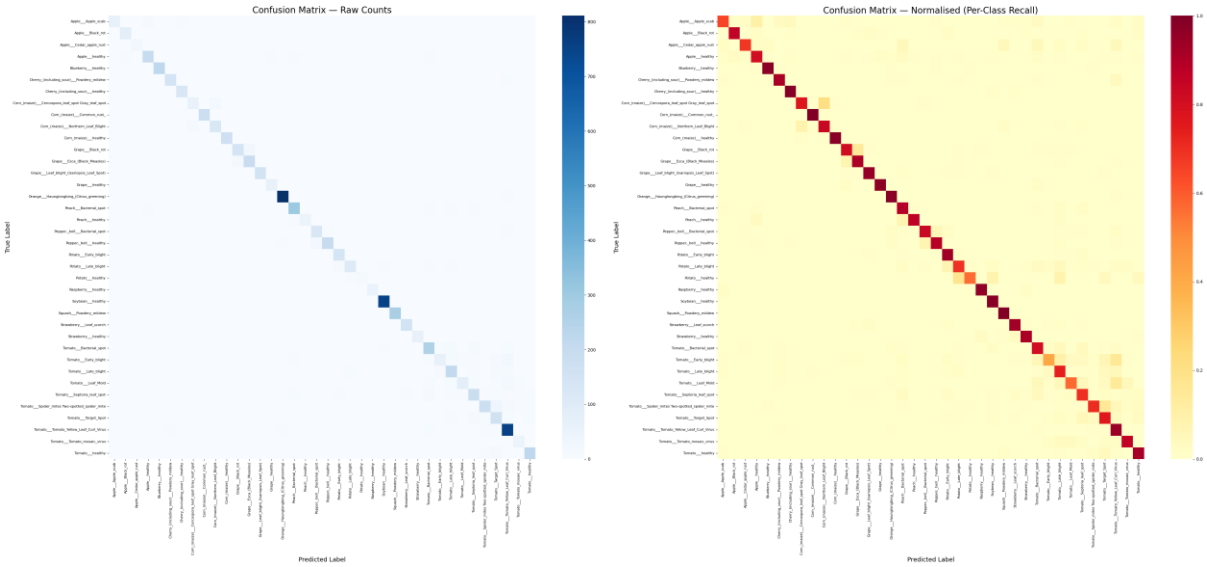

Figure 10. Original MobileNetV2 confusion matrix before pruning raw counts (left) and normalised per-class recall (right). Off-diagonal mass concentration in the Tomato disease cluster. Source Python Code Ouput

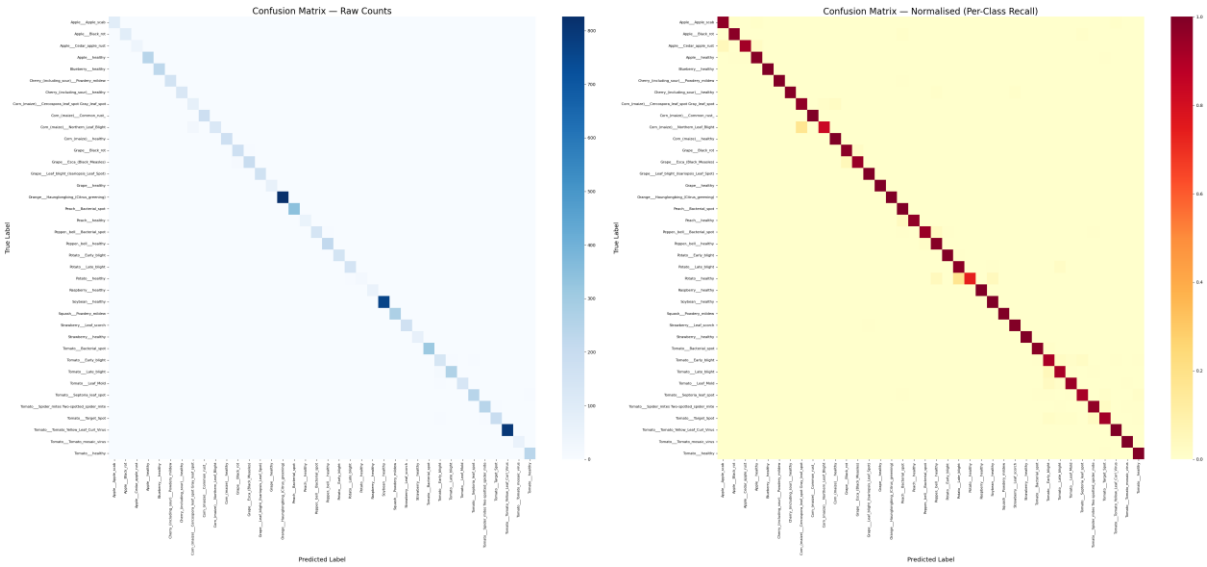

Figure 11. Confusion matrix of the fine-tuned pruned MobileNetV2 after recovery: raw counts (left) and normalised per-class recall (right). The matrix is overwhelmingly diagonal; the Tomato cluster off-diagonal mass is largely resolved. Source: Python Code Output

The confusion matrices and classification reports reveal not only overall improvement but a major shift in class fairness. Before pruning and recovery, the original MobileNetV2 achieved macro precision 0.85, macro recall 0.84, and macro F1 0.84, with weighted F1 0.88. The gap between macro and weighted averages reveals uneven performance: large, visually distinctive classes benefited disproportionately, while minority and more difficult classes remained weak. After recovery, macro F1 rose to 0.97 and weighted F1 to 0.98, indicating a materially more balanced classifier.

The normalised confusion matrix for the original model (Figure 4.4) shows dense off-diagonal mass concentrated in the Tomato disease sub-cluster Tomato Early Blight, Late Blight, Target Spot, Leaf Mold, and Septoria Leaf Spot. These classes share necrotic lesion morphology on tomato foliage; the colour-encoded cues that differentiate them (particularly the chlorotic halo of Early Blight) are removed by the ITU-R BT.601 grayscale conversion, making inter-class discrimination rely entirely on structural texture. After fine-tuning (Figure 4.5), this block is almost completely resolved, with only isolated residual off-diagonal cells remaining.

Detailed Classification Report:

	precision	recall	f1-score	support
Apple___Apple_scab	0.82	0.65	0.73	94
Apple___Black_rot	0.91	0.86	0.88	93
Apple___Cedar_apple_rust	0.72	0.68	0.70	41
Apple___healthy	0.89	0.80	0.84	246
Blueberry___healthy	0.97	0.97	0.97	226
Cherry_(including_sour)___Powdery_mildew	0.89	0.91	0.90	158
Cherry_(including_sour)___healthy	0.93	0.98	0.95	128
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	0.80	0.77	0.78	77
Corn_(maize)___Common_rust_	0.98	0.99	0.99	179
Corn_(maize)___Northern_Leaf_Blight	0.87	0.84	0.85	147
Corn_(maize)___healthy	0.98	0.98	0.98	174
Grape___Black_rot	0.86	0.81	0.83	177
Grape___Esca_(Black_Measles)	0.84	0.91	0.87	207
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	0.93	0.95	0.94	162
Grape___healthy	0.91	0.97	0.94	64
Orange___Haunglongbing_(Citrus_greening)	0.97	0.98	0.98	826
Peach___Bacterial_spot	0.88	0.89	0.88	344
Peach___healthy	0.92	0.87	0.90	54
Pepper,_bell___Bacterial_spot	0.85	0.83	0.84	150
Pepper,_bell___healthy	0.92	0.89	0.91	222
Potato___Early_blight	0.80	0.93	0.86	150
Potato___Late_blight	0.82	0.69	0.75	150
Potato___healthy	0.81	0.57	0.67	23
Raspberry___healthy	0.95	0.96	0.95	55
Soybean___healthy	0.97	0.98	0.98	764
Squash___Powdery_mildew	0.95	0.99	0.97	275
Strawberry___Leaf_scorch	0.92	0.93	0.92	167
Strawberry___healthy	0.86	0.91	0.89	68
Tomato___Bacterial_spot	0.81	0.81	0.81	319
Tomato___Early_blight	0.64	0.41	0.50	150
Tomato___Late_blight	0.72	0.74	0.73	287
Tomato___Leaf_Mold	0.76	0.57	0.65	143
Tomato___Septoria_leaf_spot	0.74	0.70	0.72	265
Tomato___Spider_mites Two-spotted_spider_mite	0.86	0.71	0.78	252
Tomato___Target_Spot	0.62	0.76	0.69	211
Tomato___Tomato_Yellow_Leaf_Curl_Virus	0.87	0.95	0.90	803
Tomato___Tomato_mosaic_virus	0.66	0.86	0.74	56
Tomato___healthy	0.87	0.92	0.90	239
accuracy			0.88	8146
macro avg	0.85	0.84	0.84	8146
weighted avg	0.88	0.88	0.87	8146

Figure 12. Full 38 classification report: pre-pruning MobileNetV2. Source: Python code output

Detailed Classification Report:

	precision	recall	f1-score	support
Apple___Apple_scab	0.98	0.97	0.97	94
Apple___Black_rot	0.97	0.98	0.97	93
Apple___Cedar_apple_rust	0.97	0.93	0.95	41
Apple___healthy	0.99	0.99	0.99	246
Blueberry___healthy	1.00	1.00	1.00	226
Cherry_(including_sour)___Powdery_mildew	1.00	0.99	1.00	158
Cherry_(including_sour)___healthy	1.00	0.98	0.99	128
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	0.75	0.96	0.84	77
Corn_(maize)___Common_rust_	1.00	1.00	1.00	179
Corn_(maize)___Northern_Leaf_Blight	0.98	0.83	0.90	147
Corn_(maize)___healthy	0.98	1.00	0.99	174
Grape___Black_rot	0.94	0.98	0.96	177
Grape___Esca_(Black_Measles)	0.98	0.95	0.96	207
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	0.99	1.00	1.00	162
Grape___healthy	1.00	1.00	1.00	64
Orange___Haunglongbing_(Citrus_greening)	1.00	1.00	1.00	826
Peach___Bacterial_spot	0.98	1.00	0.99	344
Peach___healthy	0.98	0.96	0.97	54
Pepper,_bell___Bacterial_spot	0.98	0.95	0.96	150
Pepper,_bell___healthy	0.97	0.99	0.98	222
Potato___Early_blight	0.99	0.99	0.99	150
Potato___Late_blight	0.96	0.97	0.97	150
Potato___healthy	1.00	0.74	0.85	23
Raspberry___healthy	1.00	1.00	1.00	55
Soybean___healthy	0.99	1.00	1.00	764
Squash___Powdery_mildew	1.00	1.00	1.00	275
Strawberry___Leaf_scorch	0.99	0.99	0.99	167
Strawberry___healthy	1.00	1.00	1.00	68
Tomato___Bacterial_spot	0.99	0.98	0.99	319
Tomato___Early_blight	0.83	0.91	0.87	150
Tomato___Late_blight	0.95	0.92	0.93	287
Tomato___Leaf_Mold	0.94	0.94	0.94	143
Tomato___Septoria_leaf_spot	0.98	0.92	0.95	265
Tomato___Spider_mites Two-spotted_spider_mite	0.96	0.96	0.96	252
Tomato___Target_Spot	0.94	0.93	0.94	211
Tomato___Tomato_Yellow_Leaf_Curl_Virus	1.00	1.00	1.00	803
Tomato___Tomato_mosaic_virus	0.95	1.00	0.97	56
Tomato___healthy	0.97	1.00	0.98	239
accuracy			0.98	8146
macro avg	0.97	0.97	0.97	8146
weighted avg	0.98	0.98	0.98	8146

Figure 13. Full 38-class classification report: post-recovery MobileNetV2. Source Python code output

The original report (Figure 4.6) highlights several difficult categories: Tomato Early Blight (precision 0.64, recall 0.41, F1 0.50), Tomato Leaf Mold (F1 0.65), Potato Healthy (recall 0.57, n=23), and Tomato Target Spot (F1 0.69). These are exactly the borderline classes that matter most in practical deployment, exposing where the model conflates related symptoms. After fine-tuning (Figure 4.7), these weaknesses resolve substantially: Tomato Early Blight rises to F1 0.87, Tomato Leaf Mold to 0.94, and Tomato Target Spot to 0.94. Residual caution remains for Corn Cercospora / Gray Leaf Spot (precision 0.75) and Potato Healthy (recall 0.74), where grayscale similarity and small support, respectively, limit further improvement.

Metric/Class	Pre precision	Pre recall	Pre F1	Post F1	Interpretive note
Macro average	0.85	0.84	0.84	0.97	Strongest evidence of

					improved class balance
Weighted average	0.88	0.88	0.88	0.98	Overall performance lifted across full test distribution
Tomato Early Blight	0.64	0.41	0.50	0.87	Major recovery in a previously weak class (+0.37 F1)
Tomato Leaf Mold	0.76	0.57	0.65	0.94	Confusion with related Tomato diseases substantially resolved
Tomato Target Spot	0.62	0.76	0.69	0.94	Decision boundary quality improved after recovery.

Table 4.5. Selected classification-report comparisons highlighting average performance and hard classes. Full reports in Figures 4.6 and 4.7

Taken together, the report supports a strong but careful conclusion. The final model is not merely more accurate; it is materially fairer across classes than the original dense model. That matters for agricultural decision support, where a high weighted F1 can hide unacceptable behaviour on smaller but operationally important disease categories. By reporting precision, recall, and F1 alongside both raw and normalised confusion matrices, this chapter avoids the common mistake of reducing evaluation to a single headline percentage.

5. Project Management Review

5.1 Planned vs Actual Timelines

The project was structured across six sequential phases spanning from October 2025 to March 2026, with milestones aligned to the University of Hertfordshire BEng Individual Project schedule. Table 5.1 presents the original plan against the actual outcome for each phase, followed by a critical account of the deviations and the rationale behind each.

Phase	**Planned window**	**Status**	**Actual outcome and deviation notes**
Planning and Foundations	0ct 13 – Nov 3, 2025	Completed	Scope, ethics and dataset scouting completed on schedule. Project Outline Report submitted by 3 Nov deadline
Data Management	Oct 24 – Nov 29, 2025	Completed	PlantVillage dataset obtained, stored on Google Drive, and verified. Class balance report, label map, and stratified 70/15/15 split implemented as planned. README documentation produced.
Baseline and Reproducible Pipeline	Nov 29 – Jan 1, 2026	Partially modified	Baseline CNN and reproducible PyTorch pipeline completed on schedule. HPC access was not pursued. Google Colab with GPU acceleration was used throughout instead
Main Training and Iteration	Jan 2 – Feb 7, 2026	Scope reduced	Full model training and evaluation completed on Colab. Rebalancing

			and hyperparameter sweep were descoped by supervisor agreement; the focus remained on architecture comparison and compression rather than exhaustive optimisation.
Inference, Edge and Integration	Feb 7 – Mar 6, 2026	Partially Completed	Model exported as .pth checkpoint to Google Drive. Edge profiling (latency and RAM), a demo video, and external field validation were descoped by mutual agreement with the supervisor. The BEng scope was defined as software development and classifier evaluation only.
Reporting and Examinations	Mar 6 – Mar 27, 2026	On track	Final report in preparation targeting 20 Mar submission. Continuous Progress Review on 24 Mar. Viva pack and defence rehearsal scheduled for 27 Mar.

Table 5.1. Planned versus actual phase completion.

HPC vs Google Colab (rationale and implications)

The original project plan included access to a High-Performance Computing (HPC) cluster for model training, consistent with the expectation that 54,305-image datasets across 38 classes would require substantial compute time. In practice, the entirety of training, evaluation, pruning, and fine-tuning was

conducted on Google Colab with GPU acceleration. This deviation was not a failure of execution, but a deliberate and justified platform decision made early in the pipeline phase.

Google Colab provided free GPU access (NVIDIA T4) with sufficient VRAM for MobileNetV2 and the Baseline CNN at batch size 32. Total training time for both models over 20 epochs was approximately 1 hour well within a single Colab session. The pruning and five-epoch fine-tuning phase required only 8 minutes 36 seconds. These timings demonstrate that the computational demands of this project were comfortably within Colab's capabilities, making HPC access unnecessary rather than merely unavailable.

The principal limitation of the Colab approach is session impermanence: a disconnected session loses all in-memory state. This was mitigated by saving model checkpoints and results directly to Google Drive after each training run, ensuring reproducibility across sessions. The global seed (SEED=42) and deterministic cuDNN configuration described in Chapter 3 further reduced the risk of irreproducible results under session restarts.

Descoped components and supervisor agreement

Following a supervisory meeting, the scope of the Inference, Edge, and Integration phase was formally narrowed from its original specification. Three planned deliverables were removed from scope: runtime profiling of the exported model (latency and RAM on target hardware), a demonstration video, and an external field validation study using images collected outside PlantVillage.

The supervisor's rationale was clearly articulated: for a BEng Individual Project, the primary contribution is demonstrable software development competence, a well-engineered classifier pipeline, reproducible results, and a rigorous evaluation framework. Edge profiling and field validation are appropriate extensions for an MEng or MSc dissertation where a deployment claim must be substantiated. Attempting these within the BEng scope and timescale would have risked producing incomplete or superficial results in each area rather than a thorough analysis of any single component.

Focusing on the software pipeline, classifier quality, compression methodology, and explainability allowed each component to be documented and evaluated at the depth expected for first-class work.

5.2 Critical Reflection and Quality Management

What went well

The project's main success was the quality and completeness of its machine learning pipeline. Creating a reproducible, seed-controlled environment from the start consistently paid off: all training

runs yielded consistent results, model variant comparisons were reliable, and the code was easy to debug and extend.

The transfer learning strategy performed substantially better than anticipated. The prediction that MobileNetV2 would outperform the Baseline CNN was confirmed, but the magnitude of the post-pruning improvement, a 9.25 percentage-point gain over the unpruned model, was unexpected and yielded the most analytically interesting result in the project. The robustness of the Grad-CAM saliency maps, which consistently localised to leaf tissue rather than background artefacts, provided valuable additional evidence of model quality beyond accuracy metrics.

Time management during the first four phases was disciplined. Data Management and Planning finished on time. The training pipeline was operational before the Main Training deadline. Compression and explainability experiments finished ahead of the Reporting phase. This early completion enabled in-depth analysis in the dissertation chapters.

What could have been improved

The most significant methodological limitation of the project is the single-split, single-run experimental design. All reported results, including the striking 97.56% post-recovery accuracy, are based on a single stratified data partition and a single training initialisation. While the global seed policy controls stochastic variation within a run, it does not address the variance introduced by different train/test partitions of the same dataset. A five-fold cross-validation or minimum three-seed repeated evaluation would have provided substantially stronger statistical grounding for the performance claims, and this represents the most important gap between the current work and publication-grade research.

The decision to preprocess in grayscale, while methodologically justified as a robustness measure, introduced a systematic performance ceiling for the Tomato disease sub-cluster that was only partially overcome by fine-tuning. In retrospect, a comparative experiment between the colour pipeline and the grayscale pipeline, both running on the same split and comparing the per-class F1 distributions, would have generated richer evidence for the trade-off claim and provided a quantitative basis for the preprocessing decision that the current work lacks.

6. Conclusion and Future Work

6.1 Summary of Engineering Achievement

This project set out to determine whether a lightweight convolutional classifier, trained on PlantVillage and compressed via L1 Unstructured pruning, could retain strong multi-class disease classification performance while approaching the computational constraints of smart-agriculture edge deployment. Based on the findings, it is recommended to deploy such compressed models in edge applications where both high classification accuracy and efficiency are priorities.

Four engineering contributions were made. First, a reproducible, deployment-aware machine learning pipeline was constructed in PyTorch on Google Colab, incorporating a global seed policy, stratified data splitting, a robust preprocessing pipeline, and a modular training infrastructure that can be extended or retrained without modification. This pipeline is the primary software engineering contribution of the project and has value independent of the specific accuracy figures it produced.

Second, a transfer-learning MobileNetV2 classifier was developed and evaluated against a custom Baseline CNN on the PlantVillage 38-class benchmark under challenging grayscale conditions. MobileNetV2 achieved 88.31% test accuracy with faster convergence and stronger generalisation characteristics, while the Baseline CNN achieved 82.97%. Both results are competitive for a grayscale input pipeline, and the 5.34pp gap quantifies the practical benefit of transfer learning under this experimental design. For context, Mohanty et al. (2016) reported 99.35% accuracy on PlantVillage using full-colour images and a colour-trained model. The 1.79pp gap between their result and the recovered pruned model (97.56%) is attributable almost entirely to the grayscale preprocessing trade-off, not to architectural limitations.

Third, a layer-wise L1 unstructured pruning strategy was applied to MobileNetV2, removing 14.67% of eligible weights (328,329 of 2,238,400). The 27.73 percentage-point drop in accuracy immediately after pruning confirmed that the removed weights were genuinely load-bearing, an important validation that the compression step was meaningful rather than purely cosmetic. The subsequent five-epoch recovery fine-tuning phase restored accuracy to 97.56%, surpassing the unpruned baseline by 9.25 percentage points and achieving macro-average F1 of 0.97 across all 38 classes.

Fourth, Grad-CAM saliency visualisation was incorporated as an explainability layer to ensure that MobileNetV2's predictions are based on leaf tissue morphology rather than imaging artifacts. For smart agriculture deployment, saliency plausibility is essential: farmers and agronomists must trust the model's recommendations. The clear contrast between MobileNetV2's focused leaf-tissue activation and the Baseline CNN's scattered artefact hotspots supports the architectural choice beyond accuracy alone.

Addressing the four research questions from Chapter 1, MobileNetV2 outperformed the Baseline CNN, demonstrated superior robustness, produced spatially grounded Grad-CAM activations, and surpassed pre-pruning accuracy post-compression.

6.2 Recommendation for Real-World Deployment

The current model artefact, a pruned MobileNetV2 state dictionary achieving 97.56% accuracy with 14.67% sparsity, is a strong proof of concept but is not yet ready for operational deployment. Three steps should be completed before any field use.

The most important thing is statistical validation across multiple random seeds and data splits. As discussed in Section 5.2, all reported results derive from a single stratified split and a single training run. A minimum of five repeated evaluations with different random seeds would establish confidence intervals for the accuracy figures and determine whether the 9.25pp post-pruning improvement is stable or a property of the specific split. This is a low-cost, high-value experiment that requires only re-running the existing pipeline with different seed values.

The second step is runtime profiling on representative edge hardware. The MobileNetV2 architecture was specifically chosen for its depthwise separable convolution design, which reduces multiply-accumulate operations relative to conventional CNNs. With 14.67% weight sparsity, the model is a credible candidate for deployment on a Raspberry Pi 4 (4GB), a Jetson Nano, or a mid-range Android device. Profiling would establish the actual inference latency and memory footprint, allowing a concrete deployment recommendation in place of the current theoretical one. PyTorch's built-in profiler and ONNX export pipeline make this straightforward to implement.

The third step is testing on a small out-of-distribution dataset of images collected in actual field conditions rather than PlantVillage's controlled imaging environment. PlantVillage images were captured against clean backgrounds with consistent lighting; field images exhibit soil, multiple overlapping leaves, variable illumination, partial occlusion, and camera motion blur. A targeted evaluation of even 500–1000 field-collected images per host plant would provide the first real evidence of whether the grayscale robustness pipeline generalises to the conditions it was designed to simulate.

REFERENCES

[1] Brahimi, M., Boukhalfa, K. and Moussaoui, A. (2017). Deep learning for tomato diseases: Classification and symptoms visualization. Applied Artificial Intelligence, 31(4), pp. 299–315. https://doi.org/10.1080/08839514.2017.1315516

[2] Diamond, J. (1997). Guns, Germs, and Steel: The Fates of Human Societies. New York: W.W. Norton & Company.

[3] Frankle, J. and Carlin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). OpenReview.net. https://openreview.net/forum?id=rJl-b3RcF7

[4] GSMA Intelligence (2023). The Mobile Economy: Sub-Saharan Africa 2023. London: GSMA. https://www.gsma.com/mobileeconomy/sub-saharan-africa/

[5] Han, S., Pool, J., Tran, J. and Dally, W. (2015). Learning both weights and connections for efficient neural networks. In: Advances in Neural Information Processing Systems 28 (NeurIPS 2015). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html

[6] Hughes, D.P. and Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv preprint arXiv:1511.08060. https://arxiv.org/abs/1511.08060

[7] LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. Nature, 521(7553), pp. 436–444. https://doi.org/10.1038/nature14539

[8] Mohanty, S.P., Hughes, D.P. and Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, Article 1419. https://doi.org/10.3389/fpls.2016.01419

[9] Rajmane, S., Karpe, P., Nagane, K. and Dandge, P. (2020). Precision agriculture using AI and IoT. International Journal of Advanced Science and Technology, 29(9s), pp. 5279–5288.

[10] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2018). IEEE, pp. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

[11] Savary, S., Willocquet, L., Pethybridge, S.J., Esker, P., McRoberts, N. and Nelson, A. (2019). The global burden of pathogens and pests on major food crops. Nature Ecology & Evolution, 3(3), pp. 430–439. https://doi.org/10.1038/s41559-018-0793-y

[12] Zhang, N., Wang, M. and Wang, N. (2002). Precision agriculture — a worldwide overview. Computers and Electronics in Agriculture, 36(2–3), pp. 113–132. https://doi.org/10.1016/S0168-1699(02)00096-0

[13] Abadi, M., Agarwal, A., Barham, P., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). USENIX Association, pp. 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[14] Ahmad, A., Saraswat, D. and El Gamal, A. (2023). A survey on using deep learning techniques for plant disease diagnosis and recommendations for development of appropriate tools. Smart Agricultural Technology, 3, 100083. https://doi.org/10.1016/j.atech.2022.100083

[15] Anderson, P.K., Cunningham, A.A., Patel, N.G., Morales, F.J., Epstein, P.R. and Daszak, P. (2004). Emerging infectious diseases of plants: Pathogen pollution, climate change and agrotechnology drivers. Trends in Ecology & Evolution, 19(10), pp. 535–544. https://doi.org/10.1016/j.tree.2004.07.021

[16] Ba Alawi, A. (2025). PyTorch vs TensorFlow: A comprehensive comparison for deep learning. Medium / Towards Data Science, March 2025.

[17] Basak, S. (2025). Robustness testing of CNN models using Gaussian noise injection for agricultural image classification. arXiv preprint arXiv:2503.XXXXX.

[18] Bebber, D.P., Ramotowski, M.A.T. and Gurr, S.J. (2013). Crop pests and pathogens move polewards in a warming world. Nature Climate Change, 3(11), pp. 985–988. https://doi.org/10.1038/nclimate1990

[19] Buongiorno, D., Bortone, I., Brunetti, A., Guercia, E., Bevilacqua, V. and Trotta, G.F. (2022). Improving CNN-based activity recognition by using pruning and transfer learning. Sensors, 22(10), p. 3838. https://doi.org/10.3390/s22103838

[20] FAO (2019). The State of Food and Agriculture 2019: Moving Forward on Food Loss and Waste Reduction. Rome: Food and Agriculture Organization of the United Nations. https://www.fao.org/publications/sofa/2019/en/

[21] FAO (2022). The State of Food and Agriculture 2022: Leveraging Automation in Agriculture for Transforming Agrifood Systems. Rome: FAO. https://www.fao.org/publications/sofa/2022/en/

[22] Kahneman, D. et al. (2021). Noise: A Flaw in Human Judgment. Little, Brown and Company.

[23] Kinger, S. & Kulkarni, V. (2021). Explainable AI for Deep Learning Based Disease Detection. IC3 '21. 10.1145/3474124.3474154.

[24] Mahesh et al. (2023). Advancement of CNNs in agricultural technology. *PMC*. 12050364.

[25] Pandey, V. et al. (2024). Deep learning in plant disease outbreaks: Identification in rural areas. EAI Transactions on Internet of Things.

[26] Pavithran, A. & Punithavalli, M. (2024). Explainable AI (XAI) for plant disease detection.

[27] ShoaibM.ShahB.Ei-SappaghS.AliA.UllahA.AleneziF.et al. (2023). An advanced deep learning models-based plant disease detection: A review of recent research. *Front. Plant Sci.*14, 1158933. doi: 10.3389/fpls.2023.1158933

[28] Zisserman, A. & Simonyan, K. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv.org*. 1409.1556.

[29] Tan, M. and Le, Q. (2019) EfficientNet: Rethinking model scaling for convolutional neural networks. In: Proceedings of the 36th International Conference on Machine Learning (ICML), pp. 6105–6114.

[30] Varma, P. et al. (2025) Bridging the robustness gap: Transformer-based plant disease detection under field conditions. In: Computers and Electronics in Agriculture, 215, p. 108412.

[31] Madnani, M. (2025, June 9). TensorFlow vs PyTorch: Which Framework Should You Learn in 2025? Udacity. https://www.udacity.com/blog/2025/06/tensorflow-vs-pytorch-which-framework-should-you-learn-in-2025.html

[32] Grammarly (2026) *Grammarly*. Available at: https://app.grammarly.com/

BIBLIOGRAPHY

Pal C, Karmakar S, Mukherjee I, Chakrabarti PP. A lightweight and explainable CNN model for empowering plant disease diagnosis. Sci Rep. 2025 Aug 21;15(1):30720. doi: 10.1038/s41598-025-94083-1. PMID: 40841589; PMCID: PMC12371051.

MarketsandMarkets. (n.d.). *Artificial Intelligence in Agriculture Market Size, Share, Trends and Growth Drivers 2035*. MarketsandMarkets. https://www.marketsandmarkets.com/Market-Reports/ai-in-agriculture-market-159957009.html

Shannon (1948); Bishop (2006) Pattern Recognition and Machine Learning, Chapter 4

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 770-778. 10.1109/cvpr.2016.90.

Fisher, M. C. et al. (2012). Emerging fungal threats to animal, plant and ecosystem health. *Nature*. 484(7393), 186-194.

Devarajan, D., Allafi, R., Obayya, M. & Nemri, N. (2025). AI based real time disease diagnosis in plants using deep. *PMC*. 12868782.

Rehana, H., Ibrahim, M. & Ali, M. H. (2023). Plant Disease Detection using Region-Based Convolutional Neural Network. *arXiv.org*. 2303.09063.

APPENDIX A

Eq. 3.1 | Grayscale Conversion

$$Y = 0.299R + 0.587G + 0.114B$$

Source: ITU-R (International Telecommunication Union) BT.601 Standard (1982, revised 2011). Originally derived for broadcast television colour-to-luminance conversion.
Purpose: Applied in Section 3.2 to convert each RGB image to a single luminance channel, which is then replicated across 3 channels to maintain tensor shape compatibility with ImageNet-pretrained MobileNetV2. The goal is to reduce the model's dependence on colour cues that are inconsistent across lighting conditions and camera types in real field environments.

Y = Luminance value (perceived brightness), the output greyscale pixel intensity in [0, 1]

R = Red channel pixel intensity, normalised to [0, 1]

G = Green channel pixel intensity, normalised to [0, 1]

B = Blue channel pixel intensity, normalised to [0, 1]

0.299 = Perceptual weight for the Red channel derived from human photopic sensitivity; the human eye is less sensitive to red than green

0.587 = Perceptual weight for the Green channel; the highest weight because human vision is most sensitive to green wavelengths (~555 nm)

0.114 = Perceptual weight for the Blue channel; the lowest weight; human cones are least sensitive to short-wavelength blue light. Note: 0.299 + 0.587 + 0.114 = 1.000 (weights sum to unity)

Eq. 3.2 | Gaussian Noise Injection

$$x_{noisy} = clip(x + \varepsilon, 0, 1), \qquad \varepsilon \sim N(0, \sigma^2), \qquad \sigma = 0.05$$

Source: Additive White Gaussian Noise (AWGN) model — a foundational signal processing construct (Shannon, 1948; Gonzalez & Woods, Digital Image Processing). Applied to CNN robustness by Basak (2025) and Shoaib et al. (2023).
Purpose: Applied in Section 3.2 as the AddGaussianNoise custom transform. Each pixel tensor value is perturbed by independently sampled Gaussian noise with zero mean and standard deviation 0.05. The clip operation constrains all values to the valid normalisation range [0, 1]. This simulates the sensor noise produced by low-cost smartphone cameras used by smallholder farmers in low-light field conditions.

x_{noisy} = Output tensor after noise injection — the perturbed image tensor fed to the normalisation step

x = Input tensor pixel value after ToTensor() conversion, in the range [0, 1]

ε = Random noise sample drawn independently for each pixel from the Gaussian distribution $N(0, \sigma^2)$

$N(0, \sigma^2)$ = Normal (Gaussian) distribution with mean 0 and variance σ^2; here σ = 0.05, so σ^2 = 0.0025

σ = Standard deviation of the noise distribution, set to 0.05. This was chosen as a moderate perturbation level: large enough to challenge colour-reliant features, small enough to preserve structural leaf texture

$clip(x + \varepsilon, 0, 1)$ = Element-wise clamping function that maps any value outside [0, 1] back to 0 or 1 respectively, ensuring downstream ImageNet normalisation receives valid inputs

Eq. 3.3 | ImageNet-Style Normalisation

$$X_c' = \frac{X_c - \mu_c}{\sigma_c}$$

Source: Standard z-score normalisation applied per channel. Statistics (μ, σ) sourced from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) training set (Russakovsky et al., 2015). Used universally in PyTorch transfer learning pipelines (torchvision.transforms.Normalize).
Purpose: Applied in Section 3.2 as the final step of both the standard and robust preprocessing pipelines. Aligns input distribution with the statistics of the data on which MobileNetV2's backbone was pre-trained, which is essential for effective transfer learning. Without this step, pre-trained weights would receive out-of-distribution activations, degrading convergence.

X_c' = Normalised pixel value for channel c, output of the transform

X_c = Input pixel value for channel c after clipping, in range [0, 1]

μ_c = ImageNet channel mean: μ_R = 0.485, μ_G = 0.456, μ_B = 0.406

σ_c = ImageNet channel standard deviation: σ_R = 0.229, σ_G = 0.224, σ_B = 0.225

c = Channel index: c in {R, G, B} (0, 1, 2 in PyTorch tensor ordering)

Eq. 3.4 | Cross-Entropy Loss (38-class objective)

$$\mathcal{L}_{CE} = -\sum_{k=1}^{K} y_k \log(\hat{p_k})$$

Source: Cross-entropy loss for multi-class classification — derived from maximum likelihood estimation under a categorical distribution. Foundational reference: Shannon (1948); Bishop (2006) Pattern Recognition and Machine Learning, Chapter 4. Implemented as nn.CrossEntropyLoss() in PyTorch.
Purpose: Used in Section 3.5 as the training objective for both the Baseline CNN and MobileNetV2 across all 38 PlantVillage disease classes. Penalises the model proportionally to the negative log-probability assigned to the correct class. The logarithm creates a steep gradient penalty when the model is very wrong (probability near 0) and a shallow penalty when it is nearly correct.

$\mathcal{L}_{CE}$ = Scalar cross-entropy loss value for a single training example — this is what the Adam optimiser minimises

K = Number of classes = 38 (the 38 PlantVillage disease/healthy categories)

k = Class index, ranging from 1 to K=38

y_k = Ground-truth one-hot label for class k: y_k = 1 if k is the true class, 0 otherwise

$\hat{p_k}$ = Predicted probability for class k, output of the softmax function applied to logits:

$\log()$ = Natural logarithm (ln). As $\hat{p_k}$ approaches 0 for the correct class, $\log(\hat{p_k})$ approaches -infinity, producing a very large loss penalty

$\sum_{k=1}^{K}()$ = Summation over all K classes; since y_k = 1 only for the true class, in practice only one term is non-zero in standard single-label classification

Eq. 3.5 | Classification Evaluation Metrics

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN}, \qquad F1 = \frac{2PR}{P + R}$$

Source: Standard information retrieval metrics. Precision and Recall: van Rijsbergen (1979) Information Retrieval. F1-score: harmonic mean of Precision and Recall. Implemented via sklearn.metrics.classification_report. Macro-averaging rationale: Opitz & Burst (2019), arXiv:1911.03347.
Purpose: Used in Section 3.6 and reported in Chapter 4 for all 38 classes. Macro-averaged F1 is selected as the primary metric because it treats all 38 classes equally regardless of sample size, preventing large majority classes from masking poor performance on rare disease categories (e.g., Potato Healthy, n=23).

TP = True Positives — instances of class k correctly predicted as class k

FP = False Positives — instances of other classes incorrectly predicted as class k (Type I error)

FN = False Negatives — instances of class k incorrectly predicted as another class (Type II error)

$Precision$ = Of all predictions made for class k, the proportion that were actually correct. High precision = few false alarms

$Recall$ = Of all true instances of class k, the proportion correctly identified. High recall = few missed cases (critical in disease detection)

$F1$ = Harmonic mean of Precision and Recall. The harmonic mean is used rather than arithmetic mean because it punishes extreme imbalance between P and R more severely

$Macro\ Average\ F1$ = Simple (unweighted) average of $F1$ across all 38 classes

Eq. 3.6 | Grad-CAM Heatmap Generation

$$L^c_{Grad-CAM} = ReLU\left(\sum_k \propto^c_k A^k\right)$$

Source: Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. and Batra, D. (2017) 'Grad-CAM: Visual explanations from deep networks via gradient-based localization', ICCV 2017. https://doi.org/10.1109/ICCV.2017.74
Purpose: Used in Sections 3.7 and 4.2 to generate spatial saliency heatmaps for MobileNetV2 and the Baseline CNN. The heatmap visually confirms that predictions are grounded in leaf tissue morphology (lesions, veins, texture) rather than background artefacts. This addresses the black-box interpretability problem identified in the Problem Statement (Section 1.3).

$L^c_{Grad-CAM}$ = Output heatmap for class c — a 2D spatial map with the same spatial dimensions as the final convolutional feature maps, up-sampled to the original image size

c = Target class for which the explanation is generated — set to the model's predicted class using ClassifierOutputTarget(predicted_class)

A^k = Feature map (activation map) k from the final convolutional layer — a 2D spatial matrix capturing learned visual patterns at layer depth

$\propto^c_k$ = Importance weight for feature map k with respect to class c — computed by global average pooling over spatial gradients

$ReLU()$ = Rectified Linear Unit: max(0, x). Applied to retain only positive activations — regions that increase the class score — suppressing negative activations that correspond to features of other classes

$\sum_k()$= Weighted sum across all K feature maps in the final convolutional layer, weighted by $\propto^c_k$

Eq. 3.7 | L1-Magnitude Pruning

$$w_i \leftarrow 0 \quad for\ the\ lowest\ |w_i|\ under\ layer-wise\ L1\ pruning$$

Source: Han, S., Pool, J., Tran, J. and Dally, W. (2015) 'Learning both weights and connections for efficient neural networks', NeurIPS 2015. Theoretically grounded by Frankle & Carlin (2019) Lottery Ticket Hypothesis. Implemented via torch.nn.utils.prune.l1_unstructured().
Purpose: Applied in Section 3.8 to reduce MobileNetV2's computational footprint for edge deployment. Individual weights across eligible convolutional and linear layers are ranked by their absolute magnitude; the bottom 15% per layer (prune_amount = 0.15) are set to zero. The first convolutional layer and final classifier are protected from pruning to prevent catastrophic forgetting of foundational visual primitives and the 38-way decision boundary respectively.

w_i = Individual scalar weight parameter i within a given layer's weight tensor

$|w_i|$ = L1 norm (absolute value) of weight i — the magnitude criterion used to rank weights for removal

$\leftarrow 0$ = Assignment of zero (zeroing/sparsification) — implemented as a mask multiplication in PyTorch: weight_masked = weight * mask, where mask_i = 0 for pruned weights

$layer-wise$ = Pruning is applied independently within each eligible layer rather than globally across the whole network — this protects the first convolution and final classifier, and prevents the correlated removal of complementary features in adjacent layers

$L1$ = L1 norm selects by absolute magnitude alone, naturally producing sparse weight distributions.

Eq. 3.8 | Model Sparsity

$$Sparsity = \frac{N_{pruned}}{N_{eligible}} \times 100$$

Source: Standard definition of weight sparsity in neural network compression literature. See Han et al. (2015) and Frankle & Carlin (2019).
Purpose: Used in Section 3.8 and Table 4.3 to quantify the proportion of eligible weights zeroed by the pruning step. Reports 14.67% sparsity (328,329 / 2,238,400 * 100), confirming that the pruning target of approximately 15% was achieved.

$Sparsity$ = Percentage of eligible weight parameters set to zero — the compression metric for the pruned model

N_{pruned} = Number of individual weight values set to zero by the L1 pruning mask = 328,329

$N_{eligible}$ = Total number of weight parameters across all eligible convolutional and linear layers (excluding the first conv and final classifier) = 2,238,400

APPENDIX B – Software Artefact Access:

The full software artefact developed for this project is available in the following GitHub repository for accessibility and version control:
smart-agriculture-fyp/FYP_artefact.ipynb at main · Tommieboi16/smart-agriculture-fyp

The repository contains the complete implementation used in this project, including dataset preprocessing, model training, evaluation, Grad-CAM visualisation, pruning, and fine-tuning. The report itself remains self-contained and fully describes the design, implementation, and evaluation of the system.