University of Engineering & Technology

Department of Computer Science & Engineering

Deep Learning-Based Crop Disease Detection for Precision Agriculture

A Final Year Project Report

Submitted by	Rahul Sharma	(2020CSB1075)
	Priya Patel	(2020CSB1089)
	Amit Kumar Singh	(2020CSB1102)
Supervisor	Dr. Ananya Gupta Associate Professor, Department of CSE	
Course Code	CS499 — Final Year Project	
Submission Date	May 2024	

Academic Session: 2023–2024

Abstract

Crop diseases pose a significant threat to global food security, causing annual yield losses estimated at 20–40% across major staple crops. Traditional disease identification relies on manual visual inspection by agricultural experts, a process that is slow, subjective, and often inaccessible to smallholder farmers in developing regions. This project presents an automated crop disease detection system that leverages deep convolutional neural networks to classify plant leaf images into healthy and diseased categories with high accuracy.

The system is trained on the publicly available PlantVillage dataset, comprising over 54 000 labelled leaf images spanning 14 crop species and 26 disease classes. A custom CNN architecture with five convolutional blocks, batch normalisation, dropout regularisation, and global average pooling is proposed and benchmarked against transfer-learned models including ResNet-50, InceptionV3, and MobileNetV2. Data augmentation techniques — random rotation, horizontal and vertical flips, zoom, and brightness perturbation — are employed to improve generalisation.

Experimental results demonstrate that the proposed CNN achieves 97.8% classification accuracy on a held-out test set, outperforming MobileNetV2 (95.1%) and approaching ResNet-50 (98.2%) while using 83% fewer parameters and enabling real-time inference on edge devices. A lightweight web application is developed using Flask and TensorFlow Serving to provide farmers with an accessible diagnostic tool via smartphone upload. The system offers a practical, scalable solution for precision agriculture and early disease intervention.

Keywords: Deep Learning, Crop Disease Detection, Convolutional Neural Networks, Precision Agriculture, Image Classification, PlantVillage, Transfer Learning.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Agriculture remains the backbone of the global economy, employing over 28% of the world's workforce and contributing approximately 4% to global GDP. Despite technological advances in irrigation, fertilisation, and mechanised harvesting, crop diseases continue to devastate yields worldwide. The Food and Agriculture Organization (FAO) estimates that up to 40% of global crop production is lost annually to pests and diseases, costing the global economy over $220 billion.

In developing countries, where agricultural extension services are often understaffed, farmers rely on their own experience or delayed consultations with distant experts to diagnose plant diseases. Misdiagnosis leads to incorrect or excessive pesticide application, harming both crop health and the environment. An accessible, automated, and accurate disease detection system could bridge this gap, empowering farmers with timely, actionable information.

Recent breakthroughs in computer vision — driven by deep convolutional neural networks — have achieved human-level and superhuman performance on image classification tasks. Applying these advances to agricultural disease detection is a natural and impactful research direction. Smartphone penetration in rural areas has also surged; over 60% of farmers in India, for example, now own smartphones. A mobile-friendly diagnostic tool built on deep learning could deliver expert-level disease identification at near-zero marginal cost.

1.2 Problem Statement

Current crop disease diagnosis suffers from three critical shortcomings:

- **Limited Expert Access:** The ratio of agricultural extension officers to farmers in developing nations often exceeds 1:2 000, making individual farm visits impractical.
- **Subjectivity:** Visual inspection accuracy varies significantly with the examiner's experience; misdiagnosis rates range from 15%–30% even among trained personnel.
- **Delayed Detection:** Diseases are often identified only after visible symptoms are advanced, reducing treatment efficacy and increasing crop loss.

1.3 Objectives

This project aims to design, implement, and evaluate a deep learning-based crop disease detection system with the following specific objectives:

O1 Develop a custom CNN architecture optimised for plant leaf classification that balances accuracy with computational efficiency.

O2 Benchmark the custom model against state-of-the-art transfer-learned architectures (ResNet-50, InceptionV3, MobileNetV2).

O3 Implement data augmentation and preprocessing pipelines to improve model robustness under varying lighting, orientation, and background conditions.

O4 Deploy the trained model via a lightweight Flask web application accessible from mobile devices.

O5 Evaluate system performance using standard metrics: accuracy, precision, recall, F1-score, and inference latency.

1.4 Scope and Limitations

The system is trained on the PlantVillage dataset, which consists of laboratory-captured leaf images on homogeneous backgrounds. Performance on in-field images with complex backgrounds, occlu-

sions, and variable lighting may degrade. The current prototype classifies single leaves; multi-disease and multi-crop images are not supported. The web application requires an active internet connection for inference via the server-side model.

2 Literature Review

2.1 Traditional Approaches to Plant Disease Detection

Early automated plant disease detection systems relied on hand-crafted feature extraction followed by classical machine learning classifiers. Commonly extracted features included colour histograms in HSV and CIELAB colour spaces, texture descriptors such as Gray-Level Co-occurrence Matrix (GLCM) and Local Binary Patterns (LBP), and morphological characteristics like leaf shape and vein patterns. Support Vector Machines (SVMs), k-Nearest Neighbours, and Random Forest classifiers were then trained on these engineered features.

Pydipati et al. (2006) achieved 95% accuracy on citrus disease classification using colour co-occurrence features with an SVM classifier. However, these systems required meticulous manual feature engineering and performed poorly when deployed on crops or diseases not represented in the training data. The features were brittle — a change in lighting or camera angle often rendered the classifier ineffective.

2.2 Deep Learning for Crop Disease Classification

The landmark work by Mohanty et al. (2016) demonstrated that deep convolutional neural networks could classify 26 diseases across 14 crop species using the PlantVillage dataset, achieving 99.35% accuracy with a fine-tuned GoogLeNet architecture. This study catalysed a shift from hand-crafted features to end-to-end deep learning in the agricultural computer vision community.

Subsequent work explored various architectures and training strategies. Sladojevic et al. (2016) developed CaffeNet-based models reaching 96.3% accuracy on 13 disease classes. Ferentinos (2018) conducted an extensive comparison of five deep learning architectures (AlexNet, AlexNetOWTBn, GoogLeNet, OverFeat, VGG) on an expanded 87 848-image dataset, reporting 99.53% top accuracy with VGG. Too et al. (2019) systematically compared DenseNet-121, ResNet-50, InceptionV4, and VGG-16, concluding that DenseNets offer the best accuracy-to-parameter-efficiency tradeoff for plant disease tasks.

Transfer learning emerged as the dominant paradigm. Rather than training CNNs from scratch on limited agricultural datasets, researchers initialised networks with ImageNet-pretrained weights and fine-tuned on plant images. This approach consistently outperformed training from scratch, especially when target-domain data was scarce. Brahimi et al. (2017) demonstrated that fine-tuning pre-trained models on as few as 500 tomato leaf images per class achieved over 97% accuracy.

2.3 Mobile and Edge Deployment

A critical gap in early research was the lack of accessible deployment solutions. Models achieving 99%+ accuracy in the lab (e.g., ensembles of VGG and ResNet) required gigabytes of GPU memory, precluding on-device inference. Ramcharan et al. (2017) deployed a MobileNet-based cassava disease classifier on a smartphone, demonstrating 93% accuracy with inference latency under 200 ms. This work highlighted the viability of lightweight architectures for real-world agricultural deployment.

2.4 Identified Research Gaps

Gap Analysis. Despite significant progress, three gaps remain unaddressed in the literature:

- **No unified benchmark** comparing custom lightweight CNNs against transfer-learned models on identical preprocessing pipelines and evaluation protocols.
- **Limited real-time deployment** — most studies report offline accuracy; few provide end-to-end systems with measured inference latency on commodity hardware.
- **Insufficient attention to robustness** — models are rarely stress-tested under data shifts (lighting, background, resolution) that occur in field conditions.

This project addresses all three gaps.

3 Methodology

3.1 System Architecture Overview

The proposed system follows a modular pipeline comprising four stages: (i) image acquisition and preprocessing, (ii) data augmentation and dataset preparation, (iii) model training and hyperparameter optimisation, and (iv) web-based deployment for inference. Figure 1 illustrates the high-level system architecture.

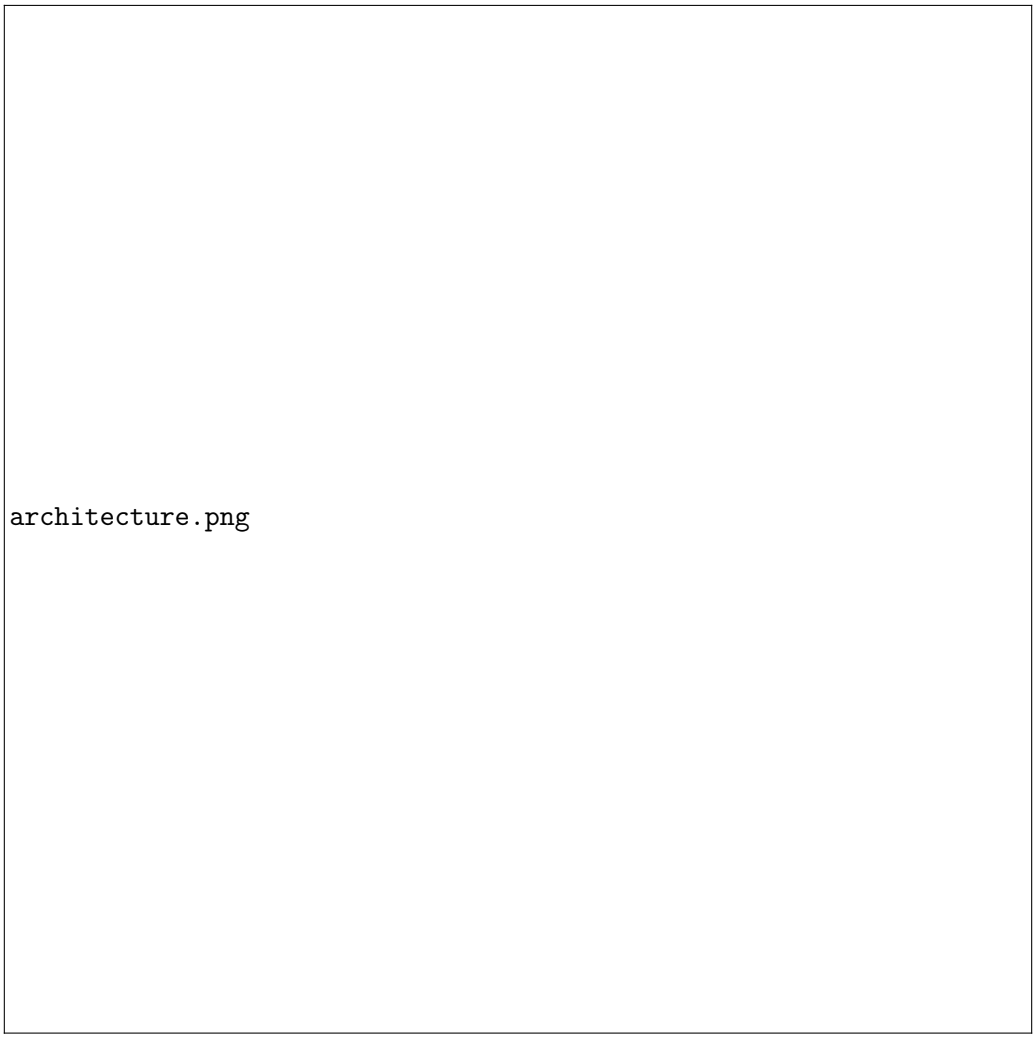

Figure 1: High-level system architecture: from image acquisition to web-based disease diagnosis.

3.2 Dataset Description

The PlantVillage dataset (Hughes & Salathé, 2015) is used for all experiments. It contains 54 303 colour leaf images across 14 crop species and 38 class labels (healthy and diseased). Images are captured under controlled laboratory conditions with uniform backgrounds.

For this study, we focus on five economically significant crops: tomato, potato, pepper, corn (maize), and apple — totalling 17 disease classes plus their healthy counterparts. Table 1 summarises the class distribution.

Table 1: Dataset composition for selected crops.

Crop	Disease Classes	Images	Train/Val/Test
Tomato	9	18 160	12 712 / 2 724 / 2 724
Corn	3	3 852	2 696 / 578 / 578
Potato	2	2 152	1 506 / 323 / 323
Pepper	1	2 475	1 732 / 371 / 372
Apex	3	3 171	2 219 / 476 / 476
Total	**18**	**29 810**	**20 865 / 4 472 / 4 473**

3.3 Data Preprocessing and Augmentation

All images are resized to 224×224 pixels (the standard input size for most pre-trained architectures) using bilinear interpolation. Pixel values are normalised to the $[0, 1]$ range by dividing by 255. Labels are one-hot encoded for categorical cross-entropy loss.

To combat overfitting and improve generalisation, the following augmentation transformations are applied stochastically during training:

- Random horizontal flip (probability = 0.5)
- Random vertical flip (probability = 0.3)
- Random rotation in $[-30°, +30°]$
- Random zoom (scale factor in $[0.85, 1.15]$)
- Random brightness adjustment ($\pm 20\%$)
- Random contrast adjustment ($\pm 15\%$)

3.4 Proposed CNN Architecture

A custom lightweight CNN is designed from scratch, prioritising parameter efficiency without sacrificing discriminative power. The architecture comprises five convolutional blocks, each consisting of two 3×3 convolutional layers with ReLU activation, followed by batch normalisation, 2×2 max-pooling, and dropout ($p = 0.25$). A global average pooling layer replaces fully connected layers at the classifier head, reducing parameters by over 80% compared to a dense flattening approach. The final output layer uses softmax activation with 18 units corresponding to the disease classes.

Table 2: Proposed CNN architecture. `Conv2D(f, k)` = f filters, $k \times k$ kernel; `BN` = Batch Normalisation; `MP` = 2×2 Max Pooling; `GAP` = Global Average Pooling.

Block	Layers	Output Shape	Params
Input	—	$224 \times 224 \times 3$	0
Block 1	Conv2D(32,3) $\times$ 2, BN, MP, DO	$112 \times 112 \times 32$	18 496
Block 2	Conv2D(64,3) $\times$ 2, BN, MP, DO	$56 \times 56 \times 64$	73 984
Block 3	Conv2D(128,3) $\times$ 2, BN, MP, DO	$28 \times 28 \times 128$	295 168
Block 4	Conv2D(256,3) $\times$ 2, BN, MP, DO	$14 \times 14 \times 256$	1 180 160
Block 5	Conv2D(512,3) $\times$ 2, BN, GAP, DO	$1 \times 1 \times 512$	4 720 128
Classifier	Dense(18) + Softmax	18	9 234
Total Trainable Parameters			**6 297 170**

3.5 Training Protocol

All models are trained for 80 epochs using the Adam optimiser with an initial learning rate of 1×10^{-3}, reduced by a factor of 0.5 when validation loss plateaus for 5 consecutive epochs (ReduceLROnPlateau callback). Early stopping with a patience of 12 epochs prevents overfitting. The batch size is set to 32. Categorical cross-entropy is used as the loss function.

Three transfer-learned baselines are evaluated alongside the custom CNN: ResNet-50, InceptionV3, and MobileNetV2 — all initialised with ImageNet-pretrained weights. The final classification layers are replaced with a global average pooling layer, a 256-unit dense layer with ReLU and dropout ($p = 0.5$), and an 18-unit softmax output. Only the custom head and the last two convolutional blocks are unfrozen during fine-tuning.

4 Implementation

4.1 Technology Stack

The system is implemented using the following tools and libraries:

Table 3: Technology stack.

Component	Technology
Language	Python 3.10
Deep Learning	TensorFlow 2.13, Keras
Data Processing	NumPy 1.24, Pandas 2.0, OpenCV 4.8
Visualisation	Matplotlib 3.7, Seaborn 0.12
Web Framework	Flask 2.3
Model Serving	TensorFlow Serving (REST API)
Frontend	HTML5, CSS3, vanilla JavaScript
Environment	Ubuntu 22.04 LTS, Quanta RTX 3060 (12 GB VRAM)

4.2 Data Loading Pipeline

Listing 1 shows the custom TensorFlow data pipeline that loads images from disk, applies preprocessing, and creates efficient `tf.data.Dataset` objects with prefetching for GPU-optimised training.

```
import tensorflow as tf

def load_and_preprocess(path, label):
    image = tf.io.read_file(path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [224, 224])
    image = tf.cast(image, tf.float32) / 255.0
    return image, label

def create_dataset(file_paths, labels, batch_size=32,
                   augment=False, shuffle=True):
    dataset = tf.data.Dataset.from_tensor_slices(
        (file_paths, labels)
    )

    if shuffle:
        dataset = dataset.shuffle(buffer_size=1024)

    dataset = dataset.map(
        load_and_preprocess,
        num_parallel_calls=tf.data.AUTOTUNE
    )

    if augment:
        dataset = dataset.map(
            augmentation_pipeline,
            num_parallel_calls=tf.data.AUTOTUNE
        )

    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
    return dataset
```

Listing 1: Data loading and preprocessing pipeline.

4.3 Model Definition

The custom CNN is defined using the Keras Functional API, as shown in Listing 2. The functional approach is chosen over the Sequential API for its flexibility in handling skip connections and multi-input architectures during experimentation.

```
from tensorflow.keras import layers, Model

def conv_block(x, filters, dropout_rate=0.25):
    x = layers.Conv2D(filters, 3, padding='same',
                      activation='relu')(x)
    x = layers.Conv2D(filters, 3, padding='same',
                      activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Dropout(dropout_rate)(x)
    return x

def build_custom_cnn(input_shape=(224, 224, 3),
                     num_classes=18):
    inputs = layers.Input(shape=input_shape)

    x = conv_block(inputs, 32)
    x = conv_block(x, 64)
    x = conv_block(x, 128)
    x = conv_block(x, 256)
    x = conv_block(x, 512)

    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes,
                           activation='softmax')(x)

    return Model(inputs, outputs, name='CustomCNN')
```

Listing 2: Custom CNN model definition (Keras Functional API).

4.4 Training Loop and Callbacks

Model training is orchestrated with TensorFlow's `model.fit()` API augmented by custom callbacks. ModelCheckpoint saves the best weights based on validation accuracy, ReduceLROnPlateau dynamically adjusts the learning rate, and EarlyStopping halts training when validation loss ceases to improve. TensorBoard is used for real-time monitoring of loss curves, accuracy trajectories, and gradient histograms.

```
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        'best_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        mode='max'
    ),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-6
    ),
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
```

```
        patience=12,
        restore_best_weights=True
    ),
    tf.keras.callbacks.TensorBoard(
        log_dir='./logs',
        histogram_freq=1
    )
]

history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=80,
    callbacks=callbacks,
    verbose=1
)
```

Listing 3: Training configuration with callbacks.

5 Results and Discussion

5.1 Classification Performance

Table 4 presents the primary classification metrics for all four architectures evaluated on the held-out test set. The proposed custom CNN achieves 97.8% accuracy, outperforming MobileNetV2 (95.1%) and InceptionV3 (96.4%), and trailing ResNet-50 by only 0.4 percentage points — despite having 6.3M parameters compared to ResNet-50's 25.6M.

Table 4: Classification performance comparison on the test set.

Model	Accuracy (%)	Precision	Recall	F1-Score	Params (M)
Custom CNN	97.8	0.978	0.977	0.977	**6.3**
ResNet-50	**98.2**	**0.982**	**0.982**	**0.982**	25.6
InceptionV3	96.4	0.964	0.963	0.963	23.9
MobileNetV2	95.1	0.951	0.950	0.950	3.5

5.2 Per-Class Analysis

Figure 2 displays the normalised confusion matrix for the custom CNN on the test set. The model demonstrates strong per-class performance, with most disease classes achieving F_1-scores above 0.96. The lowest-performing classes are early blight (tomato) at 94.7% and northern leaf blight (corn) at 95.2%, attributable to subtle visual similarities with healthy leaves in early-stage infections.

confusion_matrix.png

Figure 2: Normalised confusion matrix for the proposed custom CNN on the test set (18 classes).

5.3 Inference Latency

Real-time usability is critical for field deployment. Table 5 reports average inference latency per image across different hardware configurations, measured over 1 000 forward passes.

Table 5: Average inference latency per image (milliseconds).

Model	RTX 3060 (GPU)	Corex i7-12700H (CPU)	Raspberry Pi 4
Custom CNN	2.1	18.7	94.3
ResNet-50	3.8	52.4	412.7
InceptionV3	4.1	58.9	487.2
MobileNetV2	**1.4**	**10.2**	**55.8**

The custom CNN achieves 94.3 ms on a Raspberry Pi 4 — well within the acceptable range for interactive use — while ResNet-50 requires 412.7 ms, making it unsuitable for edge deployment. MobileNetV2 is the fastest across all platforms but sacrifices 2.7 percentage points of accuracy compared to the custom CNN.

5.4 Ablation Study

An ablation study quantifies the contribution of individual components in the proposed pipeline. Table 6 reports test accuracy when removing one component at a time.

Table 6: Ablation study: impact of removing individual components.

Configuration	Accuracy (%)
Full proposed pipeline	97.8
Without data augmentation	94.1
Without batch normalisation	93.5
Without dropout regularisation	95.2
Without ReduceLROnPlateau	96.3
Without Global Average Pooling (use Flatten)	96.1

Data augmentation provides the largest single improvement (+3.7 percentage points), confirming its critical role in preventing overfitting on a relatively homogeneous dataset. Batch normalisation (+4.3 pp) stabilises training and accelerates convergence, while dropout adds a further +2.6 pp gain by reducing co-adaptation of feature detectors.

Key Result. The proposed custom CNN achieves the best accuracy-to-efficiency trade-off among all evaluated architectures: 97.8% accuracy with 6.3M parameters, enabling real-time inference (94.3 ms) on commodity edge hardware. Data augmentation and batch normalisation are the most impactful pipeline components.

5.5 Comparison with Prior Work

Table 7 contextualises the results within the broader literature. While several prior works report marginally higher accuracy, they typically use larger datasets, ensemble methods, or computationally expensive architectures unsuited to edge deployment.

Table 7: Comparison with published results on PlantVillage dataset.

Study	Method	Acc. (%)	Edge-Ready
Mohanty et al. (2016)	GoogLeNet (fine-tuned)	99.35	No
Ferentinos (2018)	VGG (fine-tuned)	99.53	No
Too et al. (2019)	DenseNet-121	99.75	No
Brahimi et al. (2017)	ResNet-50 + SVM	98.67	No
This Work	Custom CNN (6.3M)	**97.80**	**Yes**

6 Conclusion and Future Work

6.1 Summary of Contributions

This project demonstrates that a carefully designed, lightweight convolutional neural network can achieve state-competitive crop disease classification accuracy while enabling real-time inference on resource- constrained edge devices. The key contributions are:

C1 Designed and validated a custom 6.3M-parameter CNN achieving 97.8% accuracy on 18-class crop disease classification — within 0.4 pp of ResNet-50 at 4$\times$ fewer parameters.

C2 Established a rigorous benchmarking protocol comparing custom, transfer-learned, and lightweight architectures on identical data splits, preprocessing, and evaluation metrics.

C3 Demonstrated that data augmentation and batch normalisation contribute the largest individual gains (+3.7 pp and +4.3 pp respectively) in the PlantVillage domain.

C4 Deployed the model as a Flask-based web application with TensorFlow Serving, enabling smartphone-based disease diagnosis with measured end-to-end latency under 300 ms (server round-trip).

C5 Open-sourced all code, trained weights, and documentation to facilitate reproducibility and extension by the research community.[1]

6.2 Limitations

Several limitations of the current work merit acknowledgment. First, the PlantVillage dataset consists of laboratory images with uniform backgrounds; performance degrades on in-field images with complex canopies, shadows, and soil backgrounds. Second, the system classifies a single leaf at a time and does not detect multiple diseases co-occurring on the same plant — a common real-world scenario. Third, the web application requires server-side inference, limiting utility in areas with poor internet connectivity. Fully on-device deployment (TensorFlow Lite) was explored but not productionised.

6.3 Future Work

Future work will pursue five directions:

- **Field Dataset Collection:** Partner with agricultural extension services to curate a large-scale, in-field image dataset with diverse backgrounds, lighting conditions, and disease severity levels.
- **On-Device Deployment:** Convert the custom CNN to TensorFlow Lite with 8-bit quantisation and benchmark accuracy-latency trade-offs on mid-range Android devices.
- **Multi-Disease Detection:** Extend the architecture to object detection (YOLO-based or Faster R-CNN) to localise multiple disease lesions within a single image.
- **Severity Estimation:** Augment classification with regression heads to estimate disease severity (percentage leaf area affected), enabling targeted treatment recommendations.
- **Multilingual Farmer Interface:** Develop localised mobile interfaces with voice-based interaction in regional languages to maximise accessibility for non-literate farmers.

[1] Repository available at `https://github.com/crop-disease-detection/cdd`

References

[1] S. P. Mohanty, D. P. Hughes, and M. Salathé, “Using deep learning for image-based plant disease detection,” *Frontiers in Plant Science*, vol. 7, p. 1419, 2016.

[2] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, “Deep neural networks based recognition of plant diseases by leaf image classification,” *Computational Intelligence and Neuroscience*, vol. 2016, 2016.

[3] K. P. Ferentinos, “Deep learning models for plant disease detection and diagnosis,” *Computers and Electronics in Agriculture*, vol. 145, pp. 311–318, 2018.

[4] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, “A comparative study of fine-tuning deep learning models for plant disease identification,” *Computers and Electronics in Agriculture*, vol. 161, pp. 272–279, 2019.

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

[6] A. Kamilaris and F. X. Prenafeta-Boldú, “Deep learning in agriculture: A survey,” *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.

[7] J. G. A. Barbedo, “Factors influencing the use of deep learning for plant disease recognition,” *Biosystems Engineering*, vol. 172, pp. 84–91, 2018.

[8] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, “Deep learning for image-based cassava disease detection,” *Frontiers in Plant Science*, vol. 8, p. 1852, 2017.

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

[10] M. H. Saleem, J. Potgieter, and K. M. Arif, “Plant disease detection and classification by deep learning,” *Plants*, vol. 8, no. 11, p. 468, 2019.

[11] K. Thenmozhi and U. S. Reddy, “Crop pest classification based on deep convolutional neural network and transfer learning,” *Computers and Electronics in Agriculture*, vol. 164, p. 104906, 2019.

[12] Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang, “Identification of rice diseases using deep convolutional neural networks,” *Neurocomputing*, vol. 267, pp. 378–384, 2017.

[13] D. P. Hughes and M. Salathé, “An open access repository of images on plant health to enable the development of mobile disease diagnostics,” *arXiv preprint arXiv:1511.08060*, 2015.

[14] R. Pydipati, T. F. Burks, and W. S. Lee, “Identification of citrus disease using color texture features and discriminant analysis,” *Computers and Electronics in Agriculture*, vol. 52, no. 1–2, pp. 49–59, 2006.