AMERICAN UNIVERSITY OF BEIRUT

FEDERATED MACHINE LEARNING AND TINYML INFERENCE FOR CROP DISEASE AND PEST CLASSIFICATION ON SMARTPHONES

by
HADI SAMIH HASAN

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
April 2024

AMERICAN UNIVERSITY OF BEIRUT

FEDERATED MACHINE LEARNING AND TINYML INFERENCE FOR CROP DISEASE AND PEST CLASSIFICATION ON SMARTPHONES

by
HADI SAMIH HASAN

Approved by:

	Signature
Dr. Mazen Saghir, Associate Professor Department Electrical and Computer Engineering	Advisor
	Signature
Dr. Mariette Awad, Associate Professor Department of Electrical and Computer Engineering	Co-Advisor
	Signature
Dr. Jihad Fahs, Assistant Professor Department of Electrical and Computer Engineering	Member of Committee
	Signature
Dr. Daniel Asmar, Professor Department of Mechanical Engineering	Member of Committee

Date of thesis defense: April 16, 2024

ACKNOWLEDGEMENTS

I extend my sincere appreciation to my esteemed advisor, Professor Mazen Saghir, for his exceptional guidance, unwavering support, and scholarly insight throughout the completion of this thesis. His expertise and mentorship have been invaluable in shaping this research and my academic growth.

I am also grateful to my esteemed co-advisor, Professor Mariette Awad, for her invaluable contributions, rigorous critique, and scholarly wisdom, which have significantly enriched the depth and quality of this work.

I am indebted to my esteemed committee members, Professor Jihad Fahs and Professor Daniel Asmar, for their thorough evaluation, constructive feedback, and scholarly guidance, which have elevated the thoroughness and academic integrity of this thesis.

To my beloved family, whose devoted encouragement, unwavering support, and enduring patience have been my rock throughout this academic pursuit, I express my deepest gratitude. Your belief in me has been the cornerstone of my success, and I am profoundly grateful for your unwavering support.

Lastly, I extend my heartfelt thanks to my friends for their encouragement, friendship, and moral support throughout this academic journey. Your companionship and positivity have made this endeavor all the more enriching and fulfilling.

ABSTRACT
OF THE THESIS OF

Hadi Samih Hasan for Master of Engineering
Major: Electrical and Computer Engineering

Title: Federated Machine Learning and TinyML Inference for Crop Disease and Pest Classification on Smartphones

As the agricultural industry undergoes a technological revolution, the integration of machine learning (ML) and mobile technologies emerges as a promising solution to address crop disease management efficiently. In this thesis, we present a novel approach combining federated learning (FL) and TinyML inference for crop disease classification on smartphones. Our research encompasses the development of a web application for dataset collection, complemented by a mobile application tailored for farmers. Through rigorous training, we produced multiple ML models, each specialized in detecting diseases across different plant types. These models were subsequently hosted for offline use, empowering farmers with real-time disease identification capabilities directly on their smartphones. Leveraging FL techniques, our solution ensures adaptability and scalability, crucial factors in the agricultural domain. Furthermore, employing TinyML inference enables efficient model execution on resource-constrained devices without compromising accuracy. Evaluation results demonstrate an impressive average accuracy of 98% across all deployed models. This framework represents a significant step forward in democratizing access to advanced agricultural technologies, enhancing crop disease management, and contributing to global food security.

Keywords - Federated Learning, TinyML, Crop Disease Classification, Dataset Collection, Offline Model Hosting, Real-time Disease Identification, Resource-constrained Devices.

TABLE OF CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1

INTRODUCTION

The widespread adoption of mobile phones equipped with cameras and internet connectivity has brought about transformative changes across multiple sectors, including agriculture. In this context, the data engineering industry has made significant advances in employing mobile technology to effectively tackle critical challenges, notably in the realm of crop disease identification and protection [1]. An encouraging development in this regard is the emergence of mobile applications that leverage field-captured images for disease identification. These applications heavily rely on powerful computational resources housed within data centers, enabling them to process extensive volumes of image data and deploy sophisticated deep learning models with remarkable efficiency. Through this process, they can accurately identify diseases and furnish timely and precise information for crop protection, facilitating informed decision-making by farmers.

However, a prominent drawback of the current framework lies in its reliance on a robust and uninterrupted internet connection, particularly in rural areas where farmers often encounter limited or unreliable access to the internet. This instability in connectivity hinders the seamless functioning of these applications, thereby constraining farmers from fully harnessing the potential of machine learning technologies to protect their crops.

The utilization of deep learning models in addressing the challenges posed in agricultural applications, particularly in the context of crop disease identification and protection, is motivated by several compelling reasons. One of the primary motivations is the capability of deep learning models to extract complex patterns and features from large volumes of data, such as images of crops and diseases. Deep learning models excel at tasks like image recognition and classification, making them highly effective in identifying diseases affecting crops accurately. However, while deep learning models offer exceptional performance, they often come with significant memory and computational requirements. This presents a challenge, especially when deploying these models on resource-constrained devices like mobile phones. Here's why the thesis focuses on developing compressed deep learning models:

1. Memory Size: Deep learning models, particularly convolutional neural networks (CNNs), can be memory-intensive. Mobile devices, especially in rural areas, might have limited memory available. Compressing these models reduces their memory footprint, making them more suitable for deployment on mobile phones without causing memory-related issues.
2. Energy Consumption: Running complex deep learning models can be computationally expensive, leading to increased energy consumption. This is a critical concern, as mobile devices are often powered by batteries with limited capacity. Compressed models are not only easier to load into memory but also require less energy to execute, prolonging the device's battery life.
3. Offline Execution: In rural agricultural areas, stable internet connectivity can be scarce. Deep learning models that require constant internet access for cloud-based processing can be impractical. By developing compressed models that can run directly on mobile devices offline, farmers gain the advantage of continuous access to disease identification tools, even in areas with limited or unreliable internet connections.
4. Accessibility: The ultimate goal is to make these applications more accessible to farmers in rural areas. Compressed deep learning models enable the

deployment of disease identification tools on affordable, low-end mobile devices, ensuring that even resource-constrained farmers can benefit from advanced technology without the need for high-end smartphones or constant internet access.

In response to this challenge, this thesis aims to investigate and develop offline compressed deep learning models capable of execution directly on mobile phones, by reducing the model size without compromising performance.

The key objectives of this thesis are as follows:

1. Offline Compressed Deep Learning Models: This thesis investigates the impact of federated learning on training TinyML models directly on mobile devices. It examines how this approach affects model accuracy and efficiency, considering the constraints of mobile hardware. Additionally, the study explores the implications of developing machine learning models specific to each type of plant disease on both model size and accuracy instead of training models on the entire dataset. By analyzing the trade-offs between model complexity and performance, insights into optimizing disease identification capabilities for diverse crop varieties are gained.
2. Pipeline for Collecting Image Datasets: This thesis outlines the development of a pipeline for efficiently collecting and processing images for training and validation purposes. This pipeline streamlines the data acquisition process, ensuring the availability of high-quality data for model training. Lastly, the study focuses on the creation of a mobile application designed to assist farmers in classifying diseases using an offline model with high accuracy. By leveraging the optimized models and efficient image processing pipeline, the

application empowers farmers to make timely and informed decisions regarding crop health, even in areas with limited internet access.

3. Empowering Farmers in Rural Areas: By leveraging offline capabilities, this project seeks to grant rural areas farmers access to cutting-edge machine learning technologies in the agricultural domain. Even in areas lacking reliable internet access, farmers can still utilize the advanced disease identification model integrated into the mobile application, ensuring their access to valuable technology.
4. Federated Model Training: Acknowledging the evolving nature of agricultural settings, this research thesis investigates the utilization of federated learning methods. Federated model training enables ongoing model refinement by distributing training tasks across multiple mobile devices that generate local data samples. By decentralizing the training process across multiple mobile devices, this approach ensures that the models learn from the diverse and dynamic datasets generated by farmers in different regions. This not only facilitates the detection of new and region-specific disease patterns but also enhances the models' ability to adapt swiftly to the ever-changing agricultural environment.

In conclusion, this research project aims to bridge the digital gap in agriculture by harnessing the power of offline compressed deep learning models. By enabling farmers with limited internet access to access state-of-the-art disease identification technology, the project aims to empower agricultural communities, enhance crop protection, and foster sustainable farming practices. The investigation into federated model training further ensures that the models remain up-to-date and capable of handling emerging challenges in agriculture.

CHAPTER 2

RELATED WORK

2.1 TinyML for Image Classification

TinyML, also known as Tiny Machine Learning, has garnered significant attention in the field of image classification due to its potential to deploy lightweight machine learning models directly on edge devices, such as mobile phones and IoT devices. Researchers have explored various techniques to create efficient and accurate TinyML models for image recognition tasks.

2.1.1 Overview of TinyML in Mobile Devices

TinyML is a revolutionary field that aims to deploy machine learning models on resource-constrained devices, such as mobile phones, Internet of Things (IoT) devices, and microcontrollers [2]. The integration of TinyML in mobile devices has opened up new possibilities for on-device AI processing, enabling real-time inference and reducing the dependence on cloud-based services for machine learning tasks. Researchers and engineers have been exploring various techniques and architectures to optimize and deploy deep learning models on edge devices efficiently.

TinyML aims to bring the power of artificial intelligence to low-power, memory-limited devices, enabling them to perform intelligent tasks locally without relying on cloud computing [3, 4]. This literature review provides an overview of the advancements in TinyML, including model quantization, hardware acceleration, and novel model architectures. It also highlights the challenges faced, such as balancing model size, accuracy, and energy efficiency, and the limited data and computation capabilities for training models on resource-constrained devices. Furthermore, the review explores the potential applications of TinyML in industry, agriculture and environmental monitoring. Additionally, it mentions various TinyML frameworks and tools like TensorFlow Lite, ML Kit, Edge Impulse, uTensor, PyTorch Mobile, and Arm's CMSIS-NN library that aid in deploying and optimizing machine learning models for TinyML applications.

2.1.1.1 Edge AI and TinyML Integration

Edge AI refers to the paradigm of bringing artificial intelligence capabilities directly to edge devices, where data is generated, rather than relying on centralized cloud servers. TinyML plays a crucial role in enabling Edge AI by making it possible to run lightweight machine learning models on devices with limited computational resources. Existing studies [5] have explored different methodologies to integrate TinyML into mobile devices, including model quantization, network architecture pruning, and weight clustering, to ensure efficient model execution and minimal memory footprint. Additionally, custom hardware accelerators, such as Tensor Processing Units (TPUs) and Neural Processing Units (NPUs), have been investigated to further enhance inference speed on edge devices.

2.1.1.2 Challenges and Opportunities in TinyML Deployment

While TinyML brings promising opportunities for on-device AI, it also presents several challenges that researchers have been actively addressing [6]. One of the key challenges is striking a balance between model size, accuracy, and resource utilization. TinyML models must be compact enough to fit on resource-constrained devices while maintaining sufficient accuracy for real-world applications. Achieving this balance often requires trade-offs in model complexity and the selection of appropriate model architectures. Moreover, handling the heterogeneity of edge devices and optimizing TinyML models for various hardware configurations pose additional challenges.

On the other hand, TinyML presents numerous opportunities for diverse applications, ranging from real-time image and speech recognition to predictive maintenance and environmental monitoring. By enabling local inference and data processing, TinyML reduces the latency and privacy concerns associated with cloud-based AI services.

2.1.1.3 Real-time Inference on Resource-constrained Devices

Real-time inference on resource-constrained devices is a critical requirement for many edge applications. Achieving low-latency inference while adhering to the limitations of memory, power, and computational resources poses a significant research challenge [7]. Studies have investigated techniques like model quantization, which reduces the precision of model parameters to 8-bit or even lower, resulting in faster computations. Additionally, advancements in hardware architectures, like Mobile AI accelerators and custom neural processing units, have been explored to enable efficient real-time inference. Researchers have also worked on developing efficient algorithms and optimizations that leverage the sparsity and redundancy present in neural network models to accelerate inference without compromising accuracy.

2.1.2 State-of-the-Art TinyML Models for Image Classification

TinyML research has witnessed significant advancements in developing state-of-the-art models for image classification on resource-constrained devices. Here, we explore two prominent TinyML models that have shown exceptional performance in image classification tasks:

2.1.2.1 EfficientNet-Lite for Mobile Devices

In May 2019, Google introduced a groundbreaking series of image classification models known as EfficientNet, which achieved unparalleled accuracy while using significantly fewer computations and parameters. EfficientNet-Lite is an efficient variant of the EfficientNet family, specifically designed for deployment on mobile devices [8]. This innovation had the potential to revolutionize applications on mobile and IoT devices where computational resources were limited. EfficientNet-Lite is specifically designed to run on TensorFlow Lite, catering to performance on mobile CPUs, GPUs, and EdgeTPUs. The EfficientNet-Lite series comprises five variants, offering options ranging from low latency and small model size (EfficientNet-Lite0) to high accuracy (EfficientNet-Lite4) presented in Figure 1.

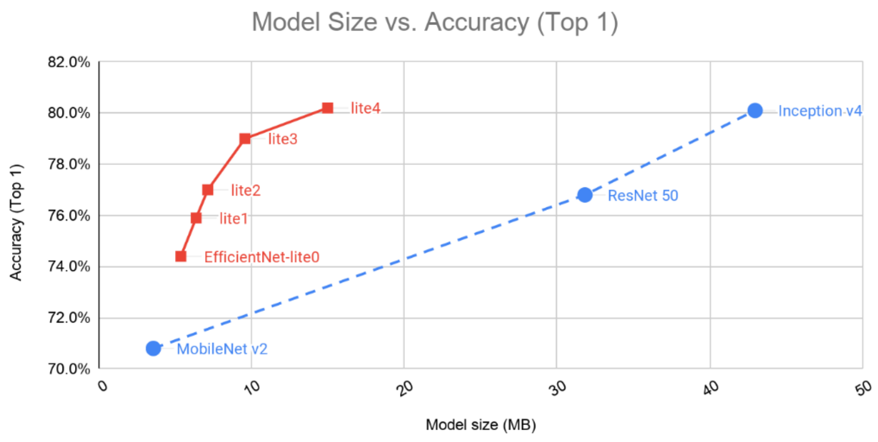


Figure 1: Model Size vs Accuracy of Different Pretrained Machine Learning Models

Even the largest variant, integer-only quantized EfficientNet-Lite4, achieves an impressive 80.4% ImageNet top-1 accuracy while maintaining real-time performance, running in just 30 milliseconds per image on a Pixel 4 CPU as shown in Figure 2.

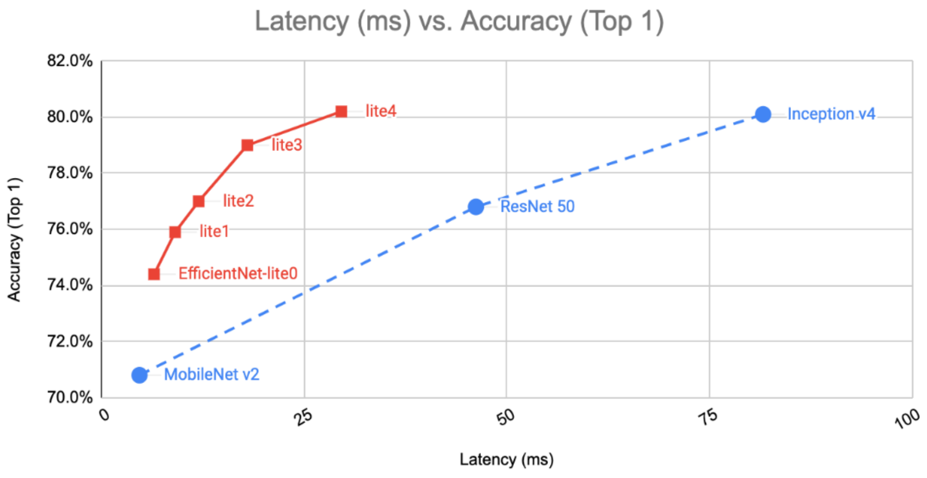


Figure 2: Latency vs Accuracy of Different Pretrained Machine Learning Models

EfficientNet-Lite addresses key challenges inherent in edge devices, notably quantization and heterogeneous hardware. Given the limited floating-point support on many edge devices, quantization is a commonly employed technique, but it often requires complex quantization-aware training or results in diminished accuracy post-training. Google's solution to this challenge involves utilizing the TensorFlow Lite post-training quantization workflow, ensuring minimal accuracy loss while quantizing the model. Another hurdle, heterogeneous hardware, poses difficulties in running the same model on a variety of accelerators, such as mobile GPUs and EdgeTPUs, due to hardware specialization. To tackle this, Google adapted the original EfficientNets by removing certain elements, switching activations, and optimizing the model's architecture to better align with the capabilities of different accelerators.

The TensorFlow Model Optimization Toolkit played a pivotal role in enabling efficient post-training quantization, leading to a 4x reduction in model size and 2x enhancement in inference speed. One notable challenge encountered was the initial accuracy drop during post-training quantization. Google's team identified that this was linked to a wide quantized output range, prompting the replacement of swish activations with "restricted-ranged" activations (RELU6). This adjustment significantly improved accuracy, mitigating the accuracy loss incurred during quantization. Overall, the EfficientNet-Lite models extend the prowess of EfficientNet to edge devices, ushering in a new era of efficient and accurate image classification in resource-constrained environments.

2.1.2.2 MobileNetV3: Squeeze-and-Excitation Networks for Mobile Vision

MobileNetV3 is an evolution of the MobileNet family, incorporating novel architectural elements to improve accuracy and efficiency for mobile vision tasks [8, 9]. The model introduces Squeeze-and-Excitation (SE) blocks, which capture channel-wise dependencies and recalibrate feature maps adaptively. This allows MobileNetV3 to achieve higher accuracy with fewer parameters, making it well-suited for deployment on edge devices with limited resources.

MobileNetV3 encompasses a range of innovative approaches, encompassing hardware-aware network architecture search (NAS), coupled with the inventive NetAdapt algorithm, to optimize network architecture based on specific mobile devices. It also devises enhanced versions of nonlinearities, such as ReLU and Swish, to maximize efficiency in mobile contexts. Its novel network design surpasses previous iterations of MobileNet models in terms of efficiency and demonstrates capacity for 8-bit quantization with minimal accuracy loss.. Furthermore, MobileNetV3 introduced an advanced segmentation decoder, more efficient than its predecessors. And it the authors validate their techniques across diverse mobile tasks including image classification, object detection, and semantic segmentation, consistently demonstrating superior performance compared to existing methods. For instance, their MobileNetV3-Large model achieves a notable 77.1% accuracy on the ImageNet classification task, outperforming the prior state-of-the-art (MnasNet) by 0.8%, while also boasting significantly faster execution, clocking at 300 FPS on a Pixel 4 phone (20% faster than MnasNet). The study's innovative contributions significantly enhance both accuracy and efficiency in neural networks tailored for mobile applications, representing a significant advancement in mobile computer vision. In addition, MobileNetV3-Small attained 75.2% ImageNet accuracy (2.4% surpassing MobileNetV2), and achieving a 700 FPS on a Pixel 4 phone (50% faster than MobileNetV2).

2.2 Federated Learning on Mobile Phones

Federated learning is a decentralized machine learning approach that enables training models on mobile devices without sharing raw data with a central server. This section explores various aspects of federated learning on mobile phones.

2.2.1 Federated Learning Architecture and Workflow

Federated learning involves a unique architecture and workflow that distinguishes it from traditional centralized machine learning. The federated averaging algorithm and gradient descent are fundamental components of federated learning [10]. Federated averaging allows mobile devices (clients) to compute model updates locally and share them securely with a central server. The server aggregates these updates to create a global model, which is then sent back to the clients for further iterations. Understanding the federated learning workflow is crucial for deploying pest image classification models on mobile devices.

Federated learning is a methodology that entails training statistical models on remote devices or isolated data centers, such as mobile phones or hospital servers, while ensuring that the data remains localized [11]. This approach presents new and distinctive challenges due to the heterogeneous and potentially extensive nature of the networks involved. As a result, it necessitates a departure from conventional techniques used in large-scale machine learning, distributed optimization, and privacy-preserving data analysis. In this context, we delve into the distinctive features and difficulties associated with federated learning, providing an extensive overview of current approaches while also outlining various avenues for future research that hold relevance for diverse research communities.

In the context of data distribution, "i.i.d." stands for "independent and identically distributed" [12]. In traditional machine learning settings, it is commonly assumed that data samples are independent of each other and are drawn from the same underlying distribution, making them identically distributed.

However, in certain scenarios, such as federated learning or distributed environments, the data collected from different sources may not satisfy the i.i.d. assumption. This means that the data samples are not independent and may come from different distributions. In such cases, the data generated on each device or participant in the network may vary significantly, leading to statistical heterogeneity.

Dealing with non-i.i.d. data poses challenges in algorithm design and optimization, as conventional machine learning algorithms often rely on the i.i.d. assumption. Federated learning, which involves training models on decentralized devices with data that is non-i.i.d., requires specialized techniques to handle the inherent complexities arising from the distribution differences among the devices. These techniques aim to ensure convergence to a global model while accounting for the varying data distributions across the network.

2.2.1.1 Federated Averaging and Gradient Descent

Federated averaging and gradient descent are key optimization techniques in federated learning [10]. Federated averaging aims to balance model updates from different clients to create a consensus model. Gradient descent, on the other hand, facilitates model updates by iteratively minimizing the loss function based on the local data of each client. Researchers have explored various improvements to federated averaging and gradient descent algorithms, such as adaptive learning rates and weight clipping, to enhance the convergence speed and robustness of federated learning on mobile phones.

2.2.1.2 Client-Server Communication in Federated Learning

Efficient communication between clients and the central server is crucial in federated learning. However, mobile devices often operate in unreliable and bandwidth-constrained networks [13]. Research has focused on developing communication-efficient protocols for federated learning, minimizing the transmission overhead while ensuring data privacy. Techniques such as quantization and compression of model updates, as well as differential privacy mechanisms, have been explored to facilitate seamless client-server communication.

2.2.1.3 Federated Learning in Unreliable Mobile Networks

Mobile devices can experience intermittent connectivity, making federated learning challenging in such scenarios. Researchers have proposed techniques to handle communication failures and latency issues in federated learning on mobile phones. For example, clients can store and buffer model updates during network disruptions, and the server can employ advanced synchronization methods to accommodate varying client participation rates.

2.2.2 Privacy and Security in Federated Learning

Preserving user privacy is a critical aspect of federated learning, especially when dealing with sensitive data on mobile devices. This subtopic explores various privacy and security mechanisms used in federated learning [14]:

2.2.2.1 Differential Privacy for Privacy Preservation

Differential privacy is a privacy-preserving technique that adds random noise to the model updates to prevent the identification of individual data samples. Implementing differential privacy in federated learning ensures that the contributions of individual clients remain confidential, safeguarding user data while maintaining model accuracy.

2.2.2.2 Secure Aggregation Protocols

Secure aggregation protocols are employed to protect model updates during aggregation at the central server. These protocols use cryptographic techniques, such as homomorphic encryption and secure multi-party computation, to perform aggregation without revealing sensitive client information.

2.2.2.3 Threats and Mitigation Strategies in Federated Learning

Federated learning faces security threats, such as model poisoning attacks and data leakage. Researchers have explored mitigation strategies to counter these threats, including robust aggregation methods, model verification, and federated learning-specific adversarial training.

2.2.3 Federated Learning Applications in Image Classification

Federated learning has demonstrated its effectiveness across multiple fields, with image classification being no exception. Federated transfer learning facilitates the exchange of insights among clients while upholding data privacy standards. Recent studies [15] have explored federated transfer learning techniques in the realm of image classification, allowing models to harness knowledge from a diverse array of mobile devices. When considering the accuracy of the proposed solution in the context of the MNIST dataset [16], it's noteworthy that even when members consent to sharing only 10% or 1% of their trained parameters at a time, the model achieves impressive accuracies of 99.14% and 98.71%, respectively. These are aligned with the performance of the centralized model, which attains an accuracy of 99.17% when trained on the entire dataset on a central server.

2.3 Convolutional Neural Networks

Convolutional Neural Networks or CNNs constitute the fundamental architecture in our research. While various CNN variations exist, the algorithms and derivations across these variations are remarkably similar [17].

2.3.1 Convolutional Neural Networks Architecture

A typical CNN is composed of multiple layers, falling into three main types: Convolutional Layers, Max-Pooling Layers, and Fully-Connected Layers as shown in Figure 3. Convolutional layers consist of a rectangular grid of neurons that take inputs from a corresponding rectangular section of the previous layer using shared weights. Max-pooling layers subsample small rectangular blocks from the preceding convolutional layer, taking the maximum value within each block. Finally, fully-connected layers facilitate high-level reasoning in the CNN, connecting all neurons from the previous layer, regardless of their spatial arrangement. After several convolutional and max-pooling layers, fully connected layers provide the core reasoning before any subsequent convolutional layers.

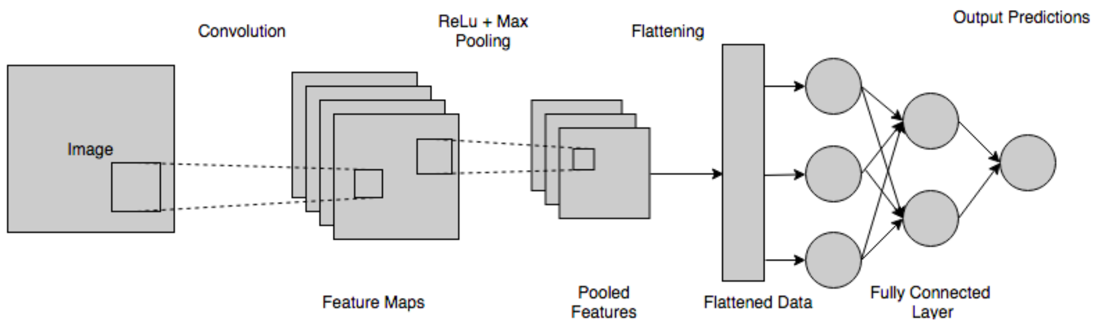


Figure 3: Convolutional Neural Network Components

2.3.2 Math Behind Convolutional Neural Networks

A typical CNN comprises several layers, falling into three distinct types: Convolutional Layers, Max-Pooling Layers, and Fully-Connected Layers. Convolutional layers consist of a rectangular grid of neurons, necessitating the previous layer to possess a similar rectangular grid structure. Each neuron within the convolutional layer receives inputs from a corresponding rectangular section of the previous layer, and the weights for this section remain uniform across all neurons in the convolutional layer. Consequently, the convolutional layer performs an image convolution operation on the previous layer, with the weights specifying the convolution filter. Furthermore, multiple grids may exist within each convolutional layer, with each grid receiving inputs from all grids in the preceding layer, potentially employing different filters.

Following each convolutional layer, a pooling layer may be introduced, wherein small rectangular blocks are extracted from the convolutional layer, and subsampling produces a single output value for each block [18, 19]. Finally, after multiple convolutional and max-pooling layers, the neural network's high-level reasoning occurs through fully connected layers. These layers connect all neurons from the previous layer, whether they originate from fully connected, pooling, or convolutional layers, and they no longer possess spatial arrangements (visualized as one-dimensional). As a result, no convolutional layers can follow a fully connected layer.

With the neural network's structure described, we proceed to analyze forward and backward propagation techniques to perform prediction and gradient computations in these neural networks.

Forward propagation involves three types of layers as specified earlier. The propagation process differs depending on the layer under consideration. For the purpose of this discussion, we shall focus on the convolutional layers and the max-pooling layers, leaving aside the fully connected networks.

In Convolutional Layers, consider a square neuron layer of size $N \times N$ followed by the convolutional layer. When utilizing an $m \times m$ filter ω, the output of the convolutional layer will be of dimensions $(N - m + 1) \times (N - m + 1)$. To compute the pre-nonlinearity input for a particular unit x_{ij}^{ℓ} in our layer, we sum up the contributions from the previous layer cells, with each contribution weighted by the corresponding filter components as shown in equation 1.

$$x_{ij}^{\ell} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{\ell-1} \tag{1}$$

Then, the convolutional layer applies its nonlinearity presented in equation 2:

$$y_{ij}^{\ell} = \sigma\left(x_{ij}^{\ell}\right) \tag{2}$$

Max-Pooling Layers are straightforward and do not involve learning. They take a $k \times k$ region and produce a single output, which represents the maximum value within that region. If the input layer is $N \times N$, the max-pooling layer's output will be $\frac{N}{k} \times \frac{N}{k}$, as each $k \times k$ block is condensed into a single value using the max function. Backward Propagation involves deriving algorithms for the two layer types, the Convolutional Layers and the Max-Pooling Layers. For the Convolutional Layers, assume that we have an error function E and we know the error values at our convolutional layer, the objective is to determine the error values at the layer before it and the gradient for each weight in the convolutional layer.

To compute the error values for the previous layer, we need to find the partial derivative of E with respect to each neuron output ($\frac{\partial E}{\partial y_{ij}^{\ell}}$). Employing the chain rule, we calculate the gradient component using equation 3 for each weight by summing the contributions from all expressions in which the variable occurs.

$$\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} \frac{\partial x_{ij}^{\ell}}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} y_{(i+a)(j+b)}^{\ell-1} \tag{3}$$

To calculate the gradient and propagate errors in the convolutional layer, we sum over all occurrences of x_{ij}^{ℓ} in which ω_{ab} is present, representing weight-sharing in the neural network. The relationship is deduced from the forward propagation equations 4 and 5.

$$\frac{\partial x_{ij}^{\ell}}{\partial \omega_{ab}} = y_{(i+a)(j+b)}^{\ell-1} \tag{4}$$

The computation of deltas ($\frac{\partial E}{\partial x_{ij}^{\ell}}$), often referred to as "deltas", is straightforward using the chain rule. Specifically,

$$\frac{\partial E}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial y_{ij}^{\ell}}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial}{\partial x_{ij}^{\ell}} \left(\sigma\left(x_{ij}^{\ell}\right)\right) = \frac{\partial E}{\partial y_{ij}^{\ell}} \sigma'\left(x_{ij}^{\ell}\right) \tag{5}$$

By leveraging the known error at the current layer ($\frac{\partial E}{\partial y_{ij}^{\ell}}$), we easily calculate the deltas ($\frac{\partial E}{\partial x_{ij}^{\ell}}$) at the current layer using the derivative of the activation function, $\sigma'(x)$. Having obtained the errors at the current layer, we possess the necessary components to compute the gradient concerning the weights used in this convolutional layer.

Furthermore, to compute the weights for this convolutional layer, we need to propagate errors back to the previous layer. By using the chain rule once more, we derive $\frac{\partial E}{\partial y_{ij}^{\ell-1}}$ as a sum over a range of a and b as presented in equation 6.

$$\frac{\partial E}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \frac{\partial x_{(i-a)(j-b)}^{\ell}}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \omega_{ab} \tag{6}$$

The relationship $\frac{\partial x^{\ell}_{(i-a)(j-b)}}{\partial y^{\ell-1}_{ij}}=\omega_{ab}$ is evident from the forward propagation equations.

This error propagation process resembles a convolution, with the filter ω applied to the layer but with $x_{(i-a)(j-b)}$ instead of $x_{(i+a)(j+b)}$. To ensure this process is well-defined for points at least m units away from the top and left edges, padding the top and left edges with zeros is necessary. Once implemented, this effectively performs a convolution using the ω filter flipped along both axes.

On the other side, the Max-Pooling Layers do not engage in learning themselves. Instead, they reduce the problem's size by introducing sparseness. During forward propagation, $k \times k$ blocks are condensed into a single value. This resulting value obtains an error from backward propagation through the previous layer and is then sent back to its origin. Since the error only originates from one location within the $k \times k$ block, backpropagated errors from max-pooling layers are notably sparse.

In conclusion, CNN offers a distinct approach to processing dimensioned and ordered data. Unlike fully connected layers, they enforce weight sharing translationally, acknowledging the relevance of data location in the input. This architectural modeling of the human visual cortex proves highly effective for tasks such as object recognition and image classification.

2.4 Image Classification of Pests

Image classification of pests using deep learning has emerged as a promising approach for automated pest detection and crop protection. This section explores various aspects of pest image classification:

2.4.1 Pest Identification and Crop Protection using Image Recognition

Automated pest detection through image recognition holds great potential for enhancing crop protection and increasing agricultural productivity. This subtopic focuses on the applications and benefits of using image recognition to identify pests in agricultural fields [20].

2.4.1.1 Challenges in Pest Detection

Pest detection poses several challenges, such as variations in pest appearance based on life stages, and diverse environmental conditions. Additionally, the presence of similar-looking pests and natural variations in plant features can lead to false positives or negatives in pest identification. Researchers have addressed these challenges by developing robust image processing techniques and leveraging advanced machine learning algorithms to improve the accuracy and reliability of pest detection systems.

2.4.1.2 Plant Pest Recognition Datasets and Benchmarks

The availability of high-quality datasets and benchmarks is crucial for training and evaluating pest recognition models [21]. Researchers have curated and shared datasets containing labeled images of various plant pests and diseases. These datasets serve as the foundation for developing and benchmarking state-of-the-art pest classification models.

2.4.1.3 State-of-the-Art Pest Classification Techniques

Advancements in deep learning techniques have led to state-of-the-art pest classification models [22]. Researchers have explored different neural network architectures, such as CNNs, to achieve high accuracy in pest identification. Additionally, feature engineering and image augmentation techniques have been utilized to enhance model performance. State-of-the-art pest classification models have demonstrated significant improvements over traditional image recognition methods.

2.4.2 Deep Learning Approaches for Pest Image Classification

Deep learning approaches have shown remarkable success in pest image classification tasks. This subtopic focuses on various deep learning methodologies applied to pest recognition:

2.4.2.1 CNN Architectures for Pest Recognition

CNNs have emerged as the backbone of many pest recognition models due to their ability to automatically learn hierarchical features from images. Researchers have experimented with various CNN architectures, such as VGG, ResNet, and DenseNet, to identify the most suitable architecture for pest image classification. Transfer learning has also been employed, where pre-trained CNNs on large datasets are fine-tuned for specific pest identification tasks.

2.4.2.2 Transfer Learning for Pest Image Classification

Transfer learning allows leveraging knowledge from pre-trained models on large datasets to improve pest classification models [23]. Researchers have investigated the transferability of features learned from generic image recognition tasks to domain-specific pest image classification. By fine-tuning pre-trained models, pest classification models can achieve better accuracy with limited labeled pest data.

2.4.2.3 Multi-scale Feature Learning for Pest Detection

Pest images can exhibit variations in scale, making it challenging to detect pests of different sizes [24]. Researchers have explored multi-scale feature learning approaches, such as pyramid pooling and multi-scale CNN architectures, to ensure that pest detection models are capable of recognizing pests at various scales.

2.5 TinyML and Federated Learning Integration

The integration of TinyML and federated learning offers several advantages and opens up new possibilities for decentralized machine learning on edge devices. This section explores the benefits and related studies of this combination:

2.5.1 Advantages of TinyML and Federated Learning Combination

2.5.1.1 Edge Computing and Local Inference with TinyML

The combination of TinyML and federated learning enables edge computing and local model inference on mobile devices [25]. TinyML models are lightweight and optimized for on-device execution, allowing real-time inference without relying on continuous cloud connectivity. Federated learning enhances this process by facilitating model training using the data stored locally on the device, eliminating the necessity to transmit raw data (images) to the cloud for central model updates. Instead, the model weights are shared, leading to updates in the central model. This edge computing paradigm reduces latency, conserves network bandwidth, and increases the central model accuracy and generalization.

2.5.1.2 Federated Learning for Decentralized Pest Data Analysis

In the context of pest image classification, federated learning allows for decentralized pest data analysis. Individual mobile devices (e.g., smartphones or IoT devices) can locally collect pest images specific to their geographical locations. These locally collected data can be effectively harnessed in a collaborative manner to continuously train the global pest image classification model using federated learning. In this approach, instead of centralizing all the data in one location, each device retains its data locally. Then, federated learning algorithms facilitate the sharing of model updates (weights) rather than raw data. These updates, derived from the locally collected data, are aggregated and used to refine the global model iteratively. This process ensures that the model benefits from insights across various geographical and that the pest data from different regions contribute to the central model training.

2.5.2 Related Studies on TinyML-Federated Learning in Different Domains

2.5.2.1 Empowering Farmers with AI: Federated Learning of CNNs for Wheat Diseases Multi-Classification [26]

In the domain of wheat disease detection and classification, this research contributes to the growing body of knowledge on leveraging collaborative learning CNNs and federated learning techniques. By addressing the challenges of decentralization, this study presents a novel approach to improve the precision and resilience of disease categorization models for wheat disease detection. Previous works have explored the importance of accurate wheat disease identification due to its significance as a staple food globally. In this particular research, the authors have introduced an approach that covers various critical stages in the development and implementation of a collaborative learning CNN model. This methodology not only outlines the process of gathering relevant data but also emphasizes the importance of preparing and refining the data through pre-processing techniques. Moreover, it highlights the significance of partitioning the data to create distinct sets for both training and testing the CNN model, ensuring its robustness and accuracy. What sets this research apart is its integration of federated averaging, a technique used to facilitate the distribution of data among different participants within the federated learning framework. This process allows for the collective utilization of data from various sources while preserving individual data privacy. The results demonstrate the effectiveness of the proposed approach, as the federated learning CNN model achieved high accuracy (0.948), precision, recall, and F1 scores (weighted-average F1: 0.946, macro-average F1: 0.944, micro-average F1: 0.946), outperforming state-of-the-art models. These findings hold potential implications for the agricultural sector, offering a promising solution to reduce harvest losses and boost crop outputs through precise and efficient disease identification, while maintaining data security and privacy. Moreover, the approach's adaptability to other domains facing data exchange constraints highlights its broader applicability in decentralized environments.

2.5.2.2 Evaluating the Potential of Federated Learning for Maize Leaf Disease Prediction [27]

The rapid evolution of technologies, coupled with the increasingly sophisticated demands of users, has led to an unprecedented surge in data. This vast amount of data contains valuable strategic information and knowledge, necessitating the use of sophisticated computational methods for extraction. In the context of maize crops, certain leaf diseases can significantly impact production and reduce quality and productivity. To address this limitation, CNNs offer intelligent applications to support crop disease diagnosis. In traditional machine learning, the dataset is typically required to be locally available for model training. This approach entails clients training the model locally with their data, ensuring privacy, and transmitting only the parameter weights of the trained model to a central server. However, Federated Learning presents an alternative, dynamic approach that accommodates heterogeneous client hardware capacity, making it suitable for decentralized training scenarios.

The experimental results validate the performance of each CNN trained with the Federated Learning paradigm. AlexNet demonstrated the shortest training time among the CNN models evaluated. Among them, VGG-11 followed by AlexNet achieved the highest accuracy, with 97.29% and 96.87% accuracy, respectively. However, VGG-11's training process required more time, making it less suitable for Federated Learning scenarios where training time is crucial. A weak negative correlation between accuracy and training time, except for VGG-11, was observed, indicating that distributed training approaches yield efficient models. Additionally, confusion matrices were analyzed to identify challenging classes in the training and generalization process of the models. The evaluation of network traffic used in the Federated Learning training process revealed that SqueezeNet exhibited lower network traffic volume despite its classification performance, owing to its trainable CNN parameters.

In conclusion, the study highlights the potential of Federated Learning to address data privacy concerns in the context of maize leaf disease classification using CNNs. AlexNet emerges as a suitable model for Federated Learning due to its structure, training time, and accuracy. The weak negative correlation between accuracy and training time suggests that distributed training approaches are efficient. Moreover, the number of CNN parameters significantly impacts the data exchanged during the Federated Learning training process. The results indicate that Federated Learning holds promise for enhancing data privacy in heterogeneous domains.

2.5.2.3 Image-based crop disease detection with federated learning [28]

The field of crop disease detection and management can significantly benefit from data science, offering decision tools to enhance productivity, reduce costs, and support environmentally friendly crop treatment methods [29, 30]. Precision agriculture aims to optimize crop yields while ensuring quality and environmental preservation, including reducing pesticide impact. Modern technologies, such as artificial intelligence, big data, image processing, and machine learning algorithms, have led to the development of systems for automatic crop disease detection and management. Deep neural networks, including CNNs, recurrent neural networks (RNNs), and Vision Transformers (ViTs) with attention mechanisms, have demonstrated outstanding performance in crop anomaly detection [31, 32, 33], providing promising opportunities for early detection and diagnosis of crop abnormalities [34]. The objective of this study is to highlight the strengths of federated learning in crop disease classification concerning user data security and confidentiality of sensitive information.

The study adopts a federated learning framework, where multiple clients contribute to training a robust global model that is shared while keeping data decentralized [35]. It involves multi-stage process that encompasses several key phases crucial for the successful implementation of the collaborative learning framework. The initial phase involves the setup and configuration of the learning process, including the establishment of communication channels and the initialization of the participating devices. Subsequently, the process proceeds to the parameter transfer phase, where the local model parameters are shared and updated among the participating devices. The next critical step is model aggregation, in which the combined knowledge from multiple devices is integrated to create an improved global model. Following this, the updated global model is transferred back to the local devices, ensuring that the collective insights are shared and disseminated for further enhancement. Finally, the local evaluation phase involves the assessment and validation of the updated model's performance on each individual device, allowing for the refinement and optimization of the learning process. These interconnected phases collectively form the foundation of the Federated Learning approach, enabling the collaborative development and enhancement of machine learning models across decentralized and privacy-sensitive environments.

After applying data augmentation techniques to enhance images, the proposed models achieve 92.24% and 91.28% accuracy for rice disease detection and classification, respectively. When testing on various plant leaf datasets, the proposed model outperforms pre-trained models like VGG-16, VGG-19, InceptionV3, ResNet50, and DenseNet201, achieving higher accuracy of 99.39%, 99.66%, and 76.59% for maize, potato and tomato, cassava leaves, and rice datasets, respectively, with reduced parameter numbers. The DenseNet121, VGG16, and MobileNetV2 models show improving performance with increasing rounds, while ViT B16 and ViT B32 also demonstrate enhanced performance with more rounds, though requiring longer computational time. InceptionV3 performs comparatively worse but peaks on the Grape dataset.

In conclusion, using machine learning technologies, particularly CNNs and Vision Transformers, for crop disease classification is a developing field. The performance of Federated Learning trained models is influenced by the number of clients. The performance of Federated Learning trained models is notably impacted by the number of participating clients within the network. As the number of clients increases, the complexity of the collaborative learning process intensifies, posing challenges such as communication overhead, increased model aggregation time, and potential issues related to data heterogeneity. With a larger number of clients, the aggregation of diverse and distributed data becomes more sophisticated, potentially leading to difficulties in achieving model convergence and performance consistency across the network. Moreover, an elevated client count can introduce variability in terms of data distribution and characteristics, potentially affecting the overall model's generalization capability and predictive accuracy. Consequently, maintaining a balance between the number of participating clients and the efficient coordination of model aggregation and updates becomes crucial for achieving optimal performance and scalability in Federated Learning setups. Furthermore, ResNet50 and MobileNetV2 have demonstrated greater robustness and suitability for Federated Learning scenarios. The number of communication rounds impacts the performance of deep architectures, and ResNet50 strikes a good balance between performance, computational cost, and complexity. While ViT B16 and ViT B32 offer better performance than some CNNs, they require more computational time, making them less suitable for Federated Learning scenarios. The performance of deep models varies with each dataset, depending on the data quality and the number of classes in each dataset.

2.5.2.4 Multiple Diseases and Pests Detection Based on Federated Learning and Improved Faster R-CNN [36]

In this context, Federated Learning is proposed as an efficient approach for improving model convergence speed and communication efficiency in orchard-related image data with no privacy protection requirements. This article presents an improved

Faster R-CNN model for detecting orchard diseases and pests based on Federated Learning. The application scenario involves multiple orchard farms collaborating with an AI company to develop a model capable of detecting various pests and diseases. However, the orchards have unbalanced and insufficient data for different pest categories, and Federated Learning is utilized to address this issue. The optimized FedAvg algorithm accelerates model training and enhances communication efficiency.

The Federated Learning algorithm offers the advantage of avoiding the uploading of large amounts of data, as frequent communication and unstable networks can significantly impact communication efficiency. Each participant uploads model parameters to the federated server for aggregation. The global model parameters for each round are obtained using a formula involving the total amount of data for all local devices and the data amount for each local device [N and n_k respectively].

The proposed improvement to FedAvg involves adding a restriction term to ensure local models do not deviate too much from the global model, promoting convergence. A fixed training period with global updates helps in selecting optimal parameters for convergence speed and communication cost. The Federated Learning process involves distributing model parameters, local training, uploading model parameters, and aggregation and update of the federated server model.

To enhance the accuracy and efficiency of orchard pest and disease detection, the article improves the network architecture of the original algorithm model and combines it with a sample expansion method. The sample expansion method plays a crucial role in augmenting the dataset by generating supplementary image samples. This technique involves the application of various image manipulation methods to diversify the existing data. Brightening involves adjusting the pixel values to enhance the overall brightness of the image, thereby expanding the spectrum of lighting conditions represented in the dataset. Noise addition introduces controlled variations in pixel values, simulating real-world imperfections and enhancing the model's robustness against noise. Mirroring or flipping horizontally and vertically aids in creating mirrored versions of the original images, contributing to a more comprehensive representation of diverse orientations and perspectives. Scaling allows for the transformation of the image size, enabling the model to learn from images with varying scales and dimensions. Lastly, random rotation involves rotating the image at random angles, promoting the model's ability to recognize objects from multiple viewpoints and angles. Collectively, these techniques aim to enrich the dataset, improve model generalization, and enhance the model's capacity to effectively handle diverse real-world scenarios. Moreover, the improved model incorporates a multi-pest detection model based on the Faster R-CNN framework. Instead of using VGG-16, the ResNet-101 structure is employed in the basic network for better detection accuracy, particularly for small target diseases.

Online Hard Example Mining (OHEM) is used to improve the accuracy of target detection based on deep convolution neural networks. It focuses on handling difficulty cases that may lead to inaccurate predictions by the network. By adjusting the threshold of negative samples and the proportion of positive and negative samples, the network can better adapt to training.

Soft Non-Maximum Suppression (Soft NM) is used to address the problem of partial occlusion of targets in the region proposal network (RPN). It ensures that important features are not overlooked due to the large field of perception in deep convolution feature maps.

The data processing phase involves sample expansion and dataset segmentation. The sample expansion method is employed to improve the generalization ability of the pest and disease detection model. The expanded dataset is divided into training, verification, and test sets.

The experiments were conducted using the deep learning open-source frameworks TensorFlow and Keras in the Python language. The improved Faster R-CNN model achieved an average accuracy of 90.27% on multiple pest detection, with a detection time of only 0.05 seconds per image. After employing federated learning, the model's mean average precision (mAP) reached 89.34%, and the training speed was improved by 59%.

In conclusion, the article proposes an improved Faster R-CNN method for detecting multiple pests in orchards based on Federated Learning. The incorporation of multiscale feature map fusion and OHEM enhances detection accuracy for different pest sizes. The application of ResNet-101 in the basic network further improves the accuracy of detecting subtle disease points, and the results demonstrate the effectiveness of the proposed method in improving pest and disease detection in orchards.

2.5.2.5 Convolutional Neural Network Applied to Plant Leaf Disease Classification [37]

This literature review explores the application of Deep Learning (DL) methods, specifically CNNs, in plant disease classification. Previous studies have used CNNs to classify plant diseases based on features like texture, type, and color of plant leaf images. This article covers the main works of the study, which include reviewing CNN networks for plant leaf disease classification, summarizing DL principles, discussing problems and solutions in CNN-based plant disease classification, and exploring future directions.

DL is a branch of machine learning used for image classification, object detection, and natural language processing. Popular CNN-based classification models include AlexNet, VGGNet, GoogLeNet, ResNet, MobileNet, and EfficientNet.

Data preparation and preprocessing involve dividing datasets into training, validation, and test sets. A suitable DL model architecture is essential for accurate classification results, and different hyperparameters are set for training and evaluation. The performance of the model is evaluated using metrics like accuracy, precision, recall, and F1 score.

Inference refers to the DL model's capability to apply its learning to new data, and deployment involves deploying the trained model for practical use, such as mobile applications for plant disease identification.

The review highlights various problems and solutions in plant disease classification. Insufficient datasets with limited size and diversity hinder classification accuracy, and solutions like data augmentation and few-shot learning are proposed. Nonideal robustness occurs when the model fails to perform well in practical conditions, and increasing dataset diversity and model robustness can address this issue. Symptom variations due to plant characteristics and environmental factors challenge disease recognition, and enriching dataset diversity is suggested. Image background complexity may affect classification, and leaf segmentation techniques are proposed to handle this.

CHAPTER 3

OVERVIEW OF TINYML FRAMEWORKS

3.1 TinyML Frameworks

TinyML frameworks play a crucial role in enabling machine learning models to run efficiently on resource-constrained devices. This section provides an overview of some popular TinyML frameworks specifically designed for image classification tasks.

3.1.1 TensorFlow Lite

TensorFlow Lite is a specialized version of the popular machine learning framework, TensorFlow, designed specifically to meet the demands of mobile and edge devices with limited computational resources [38, 39]. Its primary focus is on enabling efficient execution of machine learning models on devices like smartphones, tablets, IoT devices, and other edge devices. By catering to these resource-constrained environments, TensorFlow Lite allows AI applications to be run directly on the device, eliminating the need for constant connectivity to cloud servers.

One of the key strengths of TensorFlow Lite lies in its support for various neural network architectures. It can handle a wide range of model types, including feedforward neural networks, CNNs, RNNs, and transformer-based models, making it versatile for various use cases. This flexibility allows developers to choose the most suitable architecture for their specific application, whether it's image classification, natural language processing, object detection, or other tasks.

To ensure optimal performance on low-power devices, TensorFlow Lite offers several tools and techniques for model optimization. Model conversion is an essential step where TensorFlow models are transformed into a format that can be efficiently executed on mobile and edge devices. This conversion process often involves quantization, which reduces the precision of model weights, thereby reducing memory and computational requirements without significantly impacting accuracy. Quantized models are especially well-suited for edge devices with limited memory and processing capabilities.

Inference optimization is another critical aspect of TensorFlow Lite. During inference, the model processes input data to generate predictions. TensorFlow Lite employs a variety of techniques to accelerate inference, such as hardware acceleration using specialized co-processors or neural processing units (NPUs) available on some devices. These hardware accelerators are designed to speed up matrix computations commonly used in neural networks, resulting in faster and more power-efficient inferencing.

TensorFlow Lite's lightweight nature and efficiency make it a popular choice for developers working on mobile and edge AI applications. It empowers a wide range of real-time, on-device AI tasks, from image recognition and voice processing to personalized recommendations and natural language understanding.

3.1.2 TensorFlow Lite Micro

TensorFlow Lite for Microcontrollers (TFLite Micro) is a specialized version of TensorFlow Lite designed specifically for microcontrollers and other devices with very limited computational resources [40]. It is designed to be as small and efficient as possible, while still providing the flexibility and power of TensorFlow Lite.

TFLite Micro supports a wide range of neural network architectures, including feedforward neural networks, CNNs, and RNNs. It also supports quantization, which can significantly reduce the size and complexity of a model without significantly impacting accuracy.

TFLite Micro is tailored with a range of features optimized for microcontrollers, addressing their unique requirements. Notably, it offers a remarkably low memory footprint, allowing TFLite Micro models to occupy just a few kilobytes. This attribute is particularly advantageous for microcontrollers with severe resource constraints. Moreover, the framework emphasizes energy efficiency, ensuring that TFLite Micro models are exceptionally power-frugal. This attribute renders them exceptionally well-suited for devices reliant on battery power. Additionally, TFLite Micro extends support for hardware acceleration on select microcontrollers, further boosting overall performance.

The user-friendly nature, efficiency, and comprehensive support of TFLite Micro for various neural network architectures establish it as an invaluable asset in the realm of edge computing. This tool finds its applicability across diverse domains, including IoT devices, as TFLite Micro facilitates the creation of intelligent devices capable of data aggregation, predictive insights, and independent decision-making, all without necessitating a constant cloud connection.

3.1.3 PyTorch Mobile

PyTorch Mobile serves as an extension of the PyTorch deep learning library, bringing the power of machine learning directly to mobile devices. By enabling the deployment of machine learning models without the necessity of continuous cloud connectivity, it effectively addresses the limitations posed by restricted internet access. This empowers mobile devices to perform complex computational tasks independently, opening a world of possibilities for real-time, on-device machine learning applications [41]. Moreover, PyTorch Mobile facilitates the execution of neural networks on devices with limited computational resources. By optimizing the inference process, it enhances the speed of decision-making and conserves energy, contributing to improved battery life and overall device performance. The straightforward conversion of models and the support for quantization techniques further simplify the integration of AI functionalities into various mobile applications. This seamless integration facilitates the development of intelligent features, including accurate image classification, natural language processing, and responsive speech recognition, enabling users to interact with their devices in more intuitive and efficient ways. Moreover, by prioritizing on-device inference, PyTorch Mobile reduces reliance on external cloud services, thereby offering offline use of the machine learning models.

3.1.4 Edge Impulse

Edge Impulse is a platform designed to simplify the process of developing and deploying machine learning models on edge devices [42]. As a user-friendly and cloud-based solution, it enables developers to harness the power of machine learning for their edge devices without the need for extensive expertise in AI or data science.

Edge Impulse has a vast library of pre-trained models. These pre-built models cover a wide range of applications, from image and sound classification to anomaly detection and predictive maintenance. Additionally, Edge Impulse allows users to train custom machine learning models using their own data. This data can be easily uploaded and processed through a web interface, eliminating the need for complex local installations or infrastructure. Furthermore, Edge Impulse offers seamless integration with various popular development boards and microcontrollers, streamlining the deployment process. This integration enables prototyping and real-world testing of machine learning models on edge devices, facilitating a smooth transition from development to deployment.

3.1.5 uTensor

uTensor is a specialized open-source deep learning framework tailored explicitly for resource-constrained devices, with a primary focus on microcontrollers [43]. The framework's main objective is to address the unique challenges posed by these devices, such as limited memory and processing capabilities, and to enable the deployment of machine learning models on IoT devices and other edge devices with constrained resources.

One of the key strengths of uTensor lies in its ability to optimize memory usage and computational efficiency. Microcontrollers often have strict limitations on available RAM and flash memory, making it crucial to develop models that are efficient in terms of memory footprint. uTensor employs various techniques, such as model quantization and compression, to reduce the memory requirements of machine learning models without significantly sacrificing accuracy. This optimization process ensures that models can be easily deployed and run on microcontrollers with limited resources.

Moreover, uTensor's focus on computational efficiency ensures that the execution of machine learning models on microcontrollers is both fast and energy-efficient. This is particularly important for battery-powered IoT devices, where energy consumption must be minimized to extend the device's operational life. uTensor opens new possibilities for deploying machine learning capabilities in edge devices that were previously considered as not suitable for resource-limited devices. For example, uTensor can be utilized in sensor nodes to perform data preprocessing and filtering, enabling smarter and more localized decision-making at the edge. It can also be applied to applications such as gesture recognition, voice detection, and environmental monitoring in resource-constrained environments.

3.1.6 CMSIS-NN

CMSIS-NN, which stands for Cortex Microcontroller Software Interface Standard - Neural Network kernels, is a specialized collection of neural network kernels that have been accurately optimized for ARM Cortex-M processors [44]. This standard is developed and maintained by ARM, a semiconductor and software design company, with a focus on providing efficient and high-performance solutions for running neural networks on microcontrollers.

The primary objective of CMSIS-NN is to facilitate the deployment of TinyML models on ARM-based microcontrollers that typically have limited computational power and memory resources. By offering a collection of low-level functions, CMSIS-NN allows developers to efficiently implement the building blocks of neural networks on ARM Cortex-M processors. These kernels are optimized to take full advantage of the specific features and capabilities of these microcontrollers, enabling faster and more power-efficient execution of machine learning operations.

The optimized kernels provided by CMSIS-NN cover essential operations commonly found in neural networks, such as convolution, pooling, fully connected layers, and activation functions. These operations are crucial for tasks like image recognition, sound processing, and sensor data analysis – all of which can be performed on microcontrollers using TinyML techniques.

The efficiency of CMSIS-NN is of great importance for TinyML applications, as it ensures that machine learning models can run in real-time and consume minimal resources. This is particularly significant for edge devices in IoT applications or wearable devices, where power consumption and processing speed are critical considerations.

By standardizing these optimized neural network kernels, CMSIS-NN facilitates the development of TinyML applications on a wide range of ARM Cortex-M processors from various vendors. It provides a consistent and reliable platform for deploying machine learning models across different microcontroller architectures, fostering interoperability and ease of development for AI-driven embedded systems.

3.2 Comparison of TinyML Frameworks

The advancements in model compression, hardware acceleration, and specialized architectures have shown great potential in making TinyML a practical reality. However, addressing the associated challenges and open research questions will be critical to unlocking its full potential and ensuring its seamless integration into various industries. As the field continues to grow, TinyML has the potential to revolutionize the way we interact with edge devices and shape the future of AI at the edge and play a pivotal role in enabling efficient machine learning on resource-limited devices. Table is a concise table of comparison of popular TinyML frameworks.

Table 1: Comparison of Machine Learning Frameworks for Embedded Devices

	Federated Learning Support	**Target Platforms**	**Model Size**	**Latency**	**Ease of Use**	**Community Support**	**Cost**
TensorFlow Lite	Yes	Android, iOS, Raspberry Pi, Arduino, etc.	Up to 50 MB	Up to 100 ms	Easy to use	Large community (over 1 million users)	Free
TensorFlow Lite Micro	No (Need to be built from scratch)	Microcontrollers with limited memory	Up to 10 KB	Up to 5 ms	Easy to use	Small community (around 10,000 users)	Free
PyTorch Mobile	No (Need to be built from scratch)	Mobile devices	Up to 50 MB	Up to 100 ms	More difficult to use	Small community (around 100,000 users)	Free
Edge Impulse	No	Microcontrollers, embedded devices	Up to 10 MB	Up to 100 ms	Easy to use	Small community (around 10,000 users)	Free for basic
uTensor	No	Cortex-M microcontrollers	Up to 100 KB	Up to 10 ms	Easy to use	Small community (around 10,000 users)	Free
CMSIS-NN	No	Cortex-M microcontrollers	Up to 1 MB	Up to 20 ms	Easy to use	Small community (around 10,000 users)	Free

CHAPTER 4

DATASETS USED AND IMAGES PRE-PROCESSING

In this chapter, we delve into the crucial aspects of the datasets used in our machine learning experiments and the preprocessing techniques applied to the images. A high-quality dataset is fundamental for training accurate and robust machine learning models. Therefore, we carefully selected the PlantVillage dataset [46], a well-established and diverse collection of plant images that encompasses healthy plants and various diseases affecting different plant species. To this dataset, we appended another dataset we gathered through a website we constructed, which contains images of healthy and infected cucumber plant images with their infection type and severity.

To ensure the overall dataset is in the optimal form for model training, we perform essential preprocessing and cleaning steps on the images. This includes resizing the images to a consistent resolution, applying data augmentation techniques, and addressing noise and artifacts that might affect the model's performance. Additionally, the dataset is strategically split into training, validation, and testing subsets to evaluate the model's accuracy on unseen data.

To calculate the accuracy of our model, we have employed the confusion matrix that furnishes a comprehensive assessment of the model's performance, yielding five pivotal metrics for evaluating its validity. These metrics are essential in gauging the model's accuracy and efficiency in classification tasks. Firstly, Accuracy quantifies the proportion of correctly classified instances out of all instances, expressed as the sum of true positives (TP) and true negatives (TN) divided by the total number of instances, as shown in Equation (7).

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{7}$$

Misclassification, conversely, measures the proportion of incorrectly classified instances relative to all instances, calculated as the sum of false positives (FP) and false negatives (FN) divided by the total number of instances, as delineated in Equation (8).

$$Misclassification = \frac{(FP + FN)}{(TP + TN + FP + FN)} \tag{8}$$

Moreover, Precision, also known as Positive Predictive Value, assesses the accuracy of positive predictions relative to all positive predictions made by the model, formulated as TP divided by the sum of TP and FP, as depicted in Equation (9).

$$Precision = \frac{TP}{(TP + FP)} \tag{9}$$

In addition, Sensitivity, often referred to as Recall, evaluates the model's capability to correctly identify positive instances from all actual positive instances, calculated as TP divided by the sum of TP and FN, as illustrated in Equation (10).

$$Sensitivity = \frac{TP}{(TP + FN)} \tag{10}$$

Lastly, Specificity gauges the model's ability to correctly identify negative instances from all actual negative instances, expressed as TN divided by the sum of TN and FP, as denoted in Equation (11).

$$Specificity = \frac{TN}{TN + FP} \tag{11}$$

These equations collectively provide indispensable insights into the model's performance across various dimensions of classification, enabling a thorough assessment of its efficacy in real-world scenarios.

The following sections provide a comprehensive overview of the PlantVillage dataset, the dataset that we collected, and the preprocessing techniques applied to the images, and the rationale behind the dataset split. A robust and well-prepared dataset serves as the foundation for our machine learning endeavors, ultimately contributing to the development of an accurate and reliable plant disease detection system.

4.1 Overall Dataset (Augmented PlantVillage Dataset)

The overall dataset we've curated and named Augmented PlantVillage dataset is a combination of the widely recognized PlantVillage dataset and our own gathered collection of healthy and infected cucumber plants, resulting in a well-rounded resource. This combination will serve as the cornerstone for training our predictive model, harnessing the collective knowledge within both datasets to create a robust and adaptable tool. This overall dataset promises to enhance the accuracy and effectiveness of our model's predictions, empowering us to tackle more diseases in a more comprehensive way and offer valuable insights to aid in agricultural management decisions.

4.1.1 PlantVillage Dataset

In this section, we present a comprehensive and crucial overview of the PlantVillage dataset, which assumes a central role as the primary dataset in our machine learning experiments. With the pressing need to increase food production by an estimated 70% by 2050 to feed a projected population of over 9 billion people, addressing yield losses caused by infectious diseases becomes paramount [46]. Currently, infectious diseases reduce potential yields by an alarming average of 40%, with some farmers in the developing world experiencing devastating yield losses as high as 100%.

The PlantVillage dataset is thoughtfully structured to provide a comprehensive and organized repository of invaluable agricultural information. It encompasses a diverse collection of over 50,000 images of 38 different classes, carefully curated to include both healthy and diseased leaves from various crop plants. Each image within the dataset is enriched with careful annotations, detailing essential information such as the specific plant species and the type of disease depicted. This structured labeling ensures the dataset's reliability and utility for machine learning endeavors. This detailed labeling ensures the accuracy and reliability of our model's training, empowering it to learn distinct patterns and features associated with different plant diseases. In addition, the PlantVillage dataset encompasses a diverse array of crops and associated diseases, providing valuable insights for the identification and management of various agricultural challenges. Among the crops included are apple, blueberry, cherry, corn, grape, orange, peach, pepper, potato, tomato, raspberry, soybean, squash, and strawberry. The dataset also covers an extensive range of diseases, such as Bacterial spot, Early blight, Late blight, Black rot, Cercospora leaf spot, Gray leaf spot, Leaf mold, Powdery mildew, and Septoria leaf spot.

4.1.2 Collected Dataset from the Agricultural Department at the AUB

The PlantVillage dataset, although comprehensive, notably lacks information on cucumber plant diseases. Recognizing this gap, we have taken the initiative to develop a dedicated web portal. This platform aims to bridge the information void by facilitating the collection of data specifically related to cucumber healthy and infected plants. By focusing on this critical area of agricultural concern, we are prepared to compile a comprehensive dataset that will significantly assist in combating cucumber diseases and enhancing crop disease management strategies in the region.

We developed a comprehensive web portal using the ReactJS library for the frontend and the .NET framework for the backend, aimed at addressing the escalating cucumber disease infestation prevalent across Lebanon. The platform serves as a valuable tool for gathering images of cucumber healthy and infected plants. Through this web portal, users can conveniently indicate the severity level of the infestation and record the precise count of pests present on each leaf. This dataset of images and corresponding data points are stored within our databases.

The web portal is crafted to serve as an exclusive hub for authorized users. The portal is accessible solely through a secure sign-in and sign-up process, seamlessly integrated using the Identity Framework. This guarantees that only authorized individuals can access and add images to the database.

Our web portal is hosted on Microsoft Azure platform, ensuring a seamless user experience. Hosting a web portal on Microsoft Azure offers several practical advantages. Azure's scalable infrastructure efficiently manages varying levels of traffic, ensuring quick access even during peak usage. The platform's reliable services and robust infrastructure minimize downtime, establishing a stable environment for users. Moreover, Azure's comprehensive security features, including data encryption and identity management, contribute to the protection of user information. In addition, integration with Microsoft services simplifies authentication processes. Furthermore, the platform's support and community contribute to efficient issue resolution and continuous improvement. You can have a preview of the user-friendly sign-in and sign-up interfaces in Figure 4, showcasing the simplicity of the authentication process.

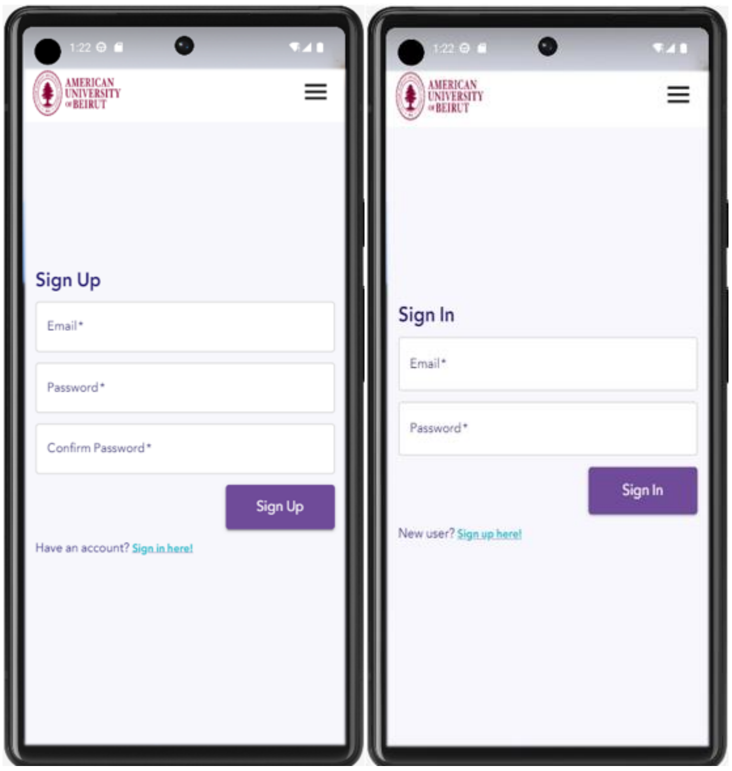
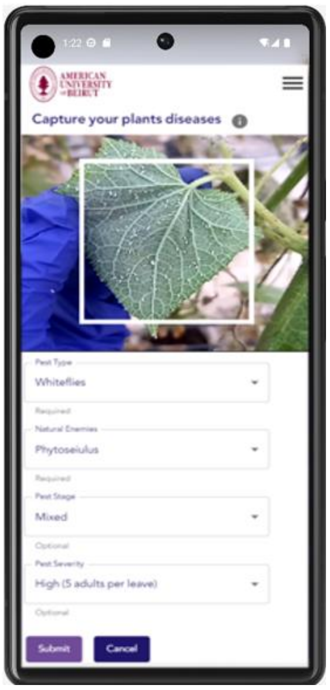


Figure 4: Sign up and Sign in Pages of the Web Portal

The image-capturing feature serves as the main component of the portal, enabling users to capture images of various plant diseases swiftly and conveniently. Moreover, this feature allows users to annotate these images accurately by selecting predefined options from intuitive dropdown menus. The illustration in Figure 5 provides a visual representation of this process, underlining the emphasis we've placed on creating a user-friendly and accessible interface. By simplifying the data collection process through intuitive design and efficient functionality, we aim to enhance the overall user experience and facilitate a more streamlined approach to data management within the platform.



Capture your plants diseases

Pest Type: Whiteflies (Required)

Natural Enemies: Phytoseiulus (Required)

Pest Stage: Mixed (Optional)

Pest Severity: High (5 adults per leave) (Optional)

Figure 5: Diseases Images Acquisition Screen

Our collaboration with the Agricultural Department at the American University of Beirut (AUB) yielded a comprehensive set of healthy and infected cucumber plant images. This collaboration provided us with an extensive dataset of approximately 1700 images of cucumber plants categorized into five classes, including healthy specimens, cucumbers affected by whiteflies with varying degrees of severity, and cucumbers affected by spider mites with varying degrees of severity. The dataset, categorized based on infection severity and corresponding quantity, supplements the PlantVillage dataset, enhancing the Augmented PlantVillage dataset's diversity and predictive capabilities. The integration of this diverse dataset enlarged the dataset and significantly enhanced the model's predictive capabilities.

4.2 Image Preprocessing and Cleaning Techniques

Data preprocessing is a crucial step in every machine learning pipeline, aimed at preparing the dataset for model training. This section discusses the various preprocessing and cleaning techniques applied to the overall dataset.

The preprocessing phase involves a series of essential steps aimed at optimizing the dataset for effective model training. One of these crucial steps involves resizing all images to a standardized resolution, ensuring uniformity across all samples. This measure is particularly vital for seamless neural network training, as these networks necessitate images of identical dimensions as inputs. Moreover, we implement various data augmentation techniques, such as rotation, flipping, and adjustments in brightness, to artificially expand the dataset. These techniques serve the purpose of enhancing the dataset's diversity, thereby strengthening the model's resilience and improving its ability to generalize patterns effectively. In addition, to address noise and artifacts present in some images, we perform cleaning operations like denoising and removing irrelevant background information. Noise reduction enhances the clarity of the images, making it easier for the model to learn relevant features. Removing irrelevant background information also enhances the model's ability to focus on the plant regions, increasing the overall accuracy of disease detection.

4.3 Dataset Splitting Strategy for Robust Machine Learning Model Training and Evaluation

In the first experiment conducted for training the TinyML models described in chapter 6. We partitioned the Augmented PlantVillage dataset into three distinct subsets: the training set (80%), the validation set (10%), and the testing set (10%). This division adhered to standard machine learning practices and facilitated comprehensive model evaluation.

The training set constituted the largest portion of the dataset and was utilized to train the machine learning models. It comprised labeled images representing both healthy and infected plants, enabling the models to learn intricate patterns and features associated with each class.

Throughout the training process, the validation set played a pivotal role in fine-tuning the models and preventing overfitting. Serving as an unseen dataset during training, it enabled us to assess the models' performance on new data and make necessary adjustments to optimize their effectiveness.

Finally, the testing set served as an independent dataset to evaluate the final performance of the trained models. It consisted of images that were not used during the training or validation phases, providing an unbiased measure of the models' generalization capability. The accuracy achieved on the testing set reflected the true effectiveness of the models in real-world scenarios.

In the second experiment, we followed a similar protocol to the first experiment, but with a focus on developing machine learning models tailored to each type of plant disease. The Augmented Plant Village dataset was initially divided into separate subsets associated with each plant type including corn, grape, peach, strawberry, tomato, potato, cherry, bell pepper, apple, and cucumber crops. This preprocessing step ensured that each subset contained healthy and infected images specific to a particular plant, facilitating targeted model training and evaluation.

Subsequently, each subset was further split into three subsets: the training set (80%), the validation set (10%), and the testing set (10%). This division maintained consistency with standard machine learning practices and ensured the availability of sufficient data for training, validation, and testing purposes.

The training set comprised the largest portion of each subset and was used to train the machine learning models with labeled images, enabling pattern learning. In addition, the validation set aided in fine-tuning and preventing overfitting. Hyperparameter tuning was based on insights from its evaluation. Finally, the testing set served as an independent dataset to assess the models' real-world effectiveness, reflecting their true generalization capability.

CHAPTER 5

FEDERATED LEARNING SETUP AND SIMULATION

Within this chapter, we will elaborate on the setup and simulation particulars of our federated learning approach. This distributed machine learning technique facilitates the training of these models by each model users across multiple nodes or devices without the need for centralized data storage. Our primary goal is to leverage federated learning to develop robust models for the classification of plant diseases. To enable this, we have employed the comprehensive dataset introduced in the previous chapter, encompassing images that represent various plant diseases as well as healthy plant samples.

5.1 Federated Learning Architecture

Federated Learning is an innovative architecture that revolutionizes the way machine learning models are trained. Unlike traditional centralized approaches, where data is collected and stored in a central server, Federated Learning enables training models directly on decentralized devices. This architecture allows devices such as smartphones, IoT devices, and edge servers to collaboratively learn from their local data while keeping it securely stored and private. By aggregating model updates instead of raw data, Federated Learning preserves privacy, reduces communication costs, and enables efficient distributed learning. Federated Learning architecture presented in Figure 6 represents a powerful paradigm shift in the machine learning landscape, fostering collaboration, privacy, and scalability in model training.

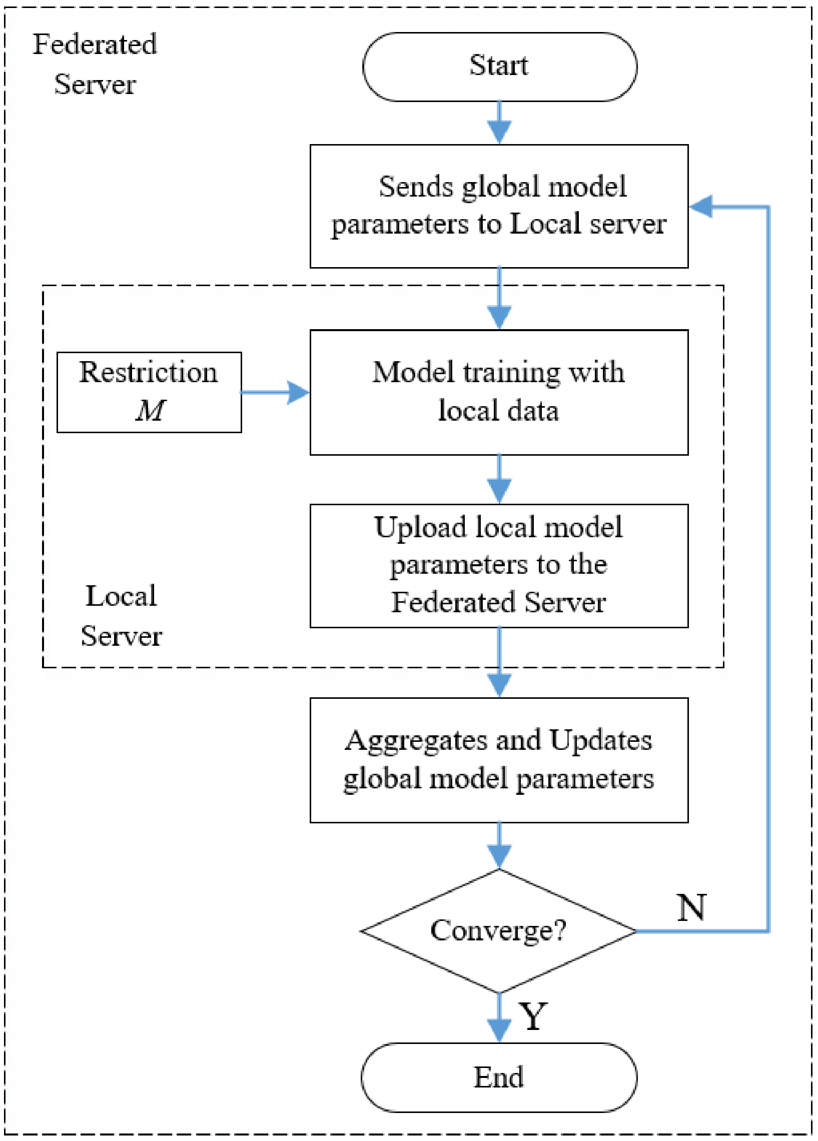

Federated Server
Start
Sends global model parameters to Local server
Restriction M
Model training with local data
Upload local model parameters to the Federated Server
Local Server
Aggregates and Updates global model parameters
Converge?
N
Y
End

Figure 6: Federated Learning Iterative Process [35]

To simulate federated learning, we deployed our architecture in a controlled environment. We set up a network of virtual clients, each emulating a mobile device with limited computational resources. During each round of federated learning, clients perform local model training on their respective data partitions. In addition, we outline the simulation setup for our federated learning experiment, using the TensorFlow Federated (TFF) API. The purpose of TFF is to facilitate federated learning, enabling users to seamlessly integrate their own TensorFlow models into the process. Throughout this section, we will describe the key steps of our simulation and highlight the important components that contribute to its success.

1. Environment Setup: To ensure the proper execution of our federated learning simulation, we first set up the required environment by installing essential packages such as "tensorflow-federated".
2. Data Preparation: The overall dataset reflects the features of real-world federated data, where each client holds a distinct subset of data, resulting in non-identically distributed data (non-i.i.d.) behavior. To prepare the input data for training, we execute data pre-processing tasks, which encompass image flattening, shuffling, batching, and image augmentation as previously outlined.
3. Model Construction: For this simulation, we utilized a comprehensive neural network architecture implemented via Keras. The model incorporates multiple layers, including an input layer, multiple hidden layers, and an output layer with a softmax function, specifically tailored to effectively accommodate the necessities of federated learning protocols.
4. Federated Proximal Algorithm: Our federated learning approach hinges on the Federated Proximal (FedProx) algorithm. This algorithm aggregates model updates from participating clients during each round of training as presented in equation 12.

   $$\min_{w} f(w) \approx \frac{1}{K}\sum_{k=1}^{K} p_k F_k(w) \tag{12}$$

   this ensures the convergence of the global model. FedProx is a collaborative learning approach across distributed devices. FedProx involves performing a specified number of epochs of Stochastic Gradient Descent (SGD) on each of the K devices involved in the federated learning process. On each device, local surrogate functions (F_k) are employed to compute model updates. To ensure consistent learning across all participating devices, we enforced the use of uniform learning rates and a consistent number of local epochs. Additionally, model updates from a subset of devices are averaged during each round, contributing to the collaborative refinement of the global model. This approach not only facilitates the convergence of the model but also mitigates the challenges associated with training on non-identically distributed data across decentralized devices.

In addition, to address the challenge of varying local updates and accommodate statistical heterogeneity, we implemented a technique known as B-Bounded Dissimilarity. Denoted in equation 13 as

$$\min_{w} \; h_k(w; w^t) \approx F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 \tag{13}$$

this method guides local updates to be closer to the initial global model. By enforcing a bounded dissimilarity constraint, denoted as B, we mitigate the divergence of local updates, promoting convergence towards a consistent global model across decentralized devices. This approach not only facilitates collaboration among distributed participants but also enhances the stability and efficiency of the federated learning process.

5. Federated Training Rounds: In the simulation, we run 100 rounds of federated training. At the end of each round, the global model is updated based on contributions from individual clients' local models. Throughout the training process, we closely monitor crucial training metrics, including loss and accuracy.

Loss refers to the measure of error between the predicted outcome and the actual target, helping us assess the model's precision in making predictions. On the other hand, accuracy signifies the model's ability to provide correct predictions in relation to the total number of predictions made, allowing us to gauge the overall effectiveness and reliability of the model. By closely monitoring these crucial metrics, we ensure a comprehensive evaluation of the model's performance and make informed decisions to enhance its training accuracy.

6. Model Evaluation on Federated Data: Lastly, we evaluate the trained model on federated data using TFF's "build_fed_eval" function. This evaluation process provides valuable metrics, such as loss and accuracy as discussed earlier, enabling us to assess the model's performance in a federated setting.

In conclusion, the insights gained from this simulation encompass a deeper understanding of managing diverse and non-identically distributed data in a federated learning setting. Additionally, the evaluation of models within the context of federated learning provides valuable knowledge about the intricacies of model performance and adaptability in decentralized environments.

5.2 Experiment 1 Simulation (One Model for All Plants)

This section presents the setup and methodology of our federated learning experiment, focusing on the development of robust models for plant disease classification through distributed machine learning techniques. Our simulation replicates the federated learning framework in a controlled environment, showcasing its effectiveness in training global models across distributed data sources.

5.2.1 Initial Training and Testing

Initially, the model was trained using 50% of the entire dataset containing all plant disease classes. Specifically, 40% of the data was distributed across 8 clients (5% each), simulating diverse data sources. To mimic real-world scenarios and enhance robustness, an additional set of noise images was added to each client dataset. The remaining 10% of the data was reserved for testing and reporting the final accuracy of the model. Achieving a testing accuracy of 92.77%, this phase established a baseline for assessing improvements in federated learning.

5.2.2 Federated Training Rounds

We conducted 100 rounds of federated training, closely monitoring key training metrics such as loss and accuracy to monitor model performance. Following data distribution among clients, federated learning commenced. Central models were initialized with parameters derived from initial training, while clients executed local training iterations, updating model weights based on their respective dataset characteristics.

5.2.3 Evaluation of Optimal Image Count per Federated Learning Round

In order to enhance the efficiency and convergence speed of federated learning, we conducted a series of experiments to determine the optimal number of images to be used prior to commencing each federated training round. Our investigation involved distributing the available images among 8 clients in a random manner, with varying participation rates in each round. The objective was to find a balance between minimizing communication rounds and achieving the shortest convergence time.

The experiments were conducted on the Augmented PlantVillage dataset. And through extensive analysis and evaluation, we observed that employing 800 images achieved a remarkable tradeoff between the least communication rounds required and the shortest convergence time as presented in Figure 7. This finding demonstrates the significance of carefully selecting the appropriate image count in federated learning scenarios.

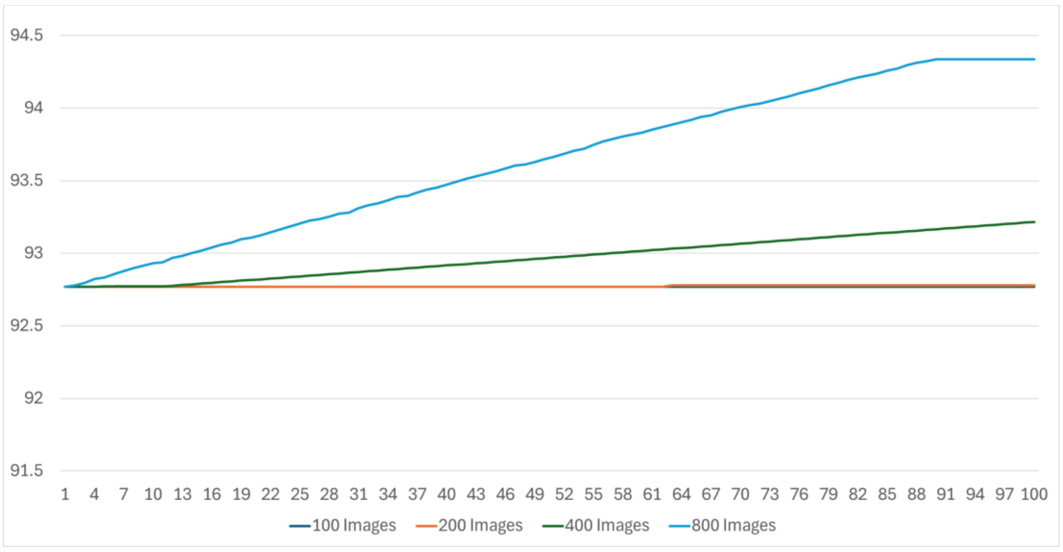


Figure 7: Evaluation of Optimal Image Count per Federated Learning Round

By utilizing 800 images, we were able to strike an optimal balance that reduced the overall communication overhead while simultaneously accelerating the convergence process. This optimal tradeoff allows for efficient utilization of computational resources and ensures that federated learning can be effectively deployed in real-world applications.

The choice of 8 clients in our federated learning model is derived from the dataset size and the operational constraints imposed by the requirement to distribute a minimum of 800 images per federated round to each client. This criterion ensures that each client receives a substantial and representative subset of the data for training, promoting model generalization and robustness. Given the dataset's characteristics and the need to balance computational efficiency with data diversity, the number 8 emerged as the optimal choice to meet these requirements. By distributing the dataset across multiple clients in this manner, we can effectively leverage the collective intelligence of diverse data sources while accommodating practical considerations such as communication overhead and resource utilization. Therefore, the selection of 8 as the number of clients is informed by the dataset size and the need to ensure adequate data distribution for successful federated learning.

Finally, the significance of this finding lies in its potential to enhance the scalability and efficiency of federated learning systems. By reducing the number of communication rounds required for convergence, computational resources can be utilized more effectively, leading to improved training efficiency and reduced training time.

5.2.4 Aggregation and Model Improvement

In our study, we employed the FedProx optimization function to enhance the performance of federated learning models. FedProx integrates proximal terms into the optimization process to mitigate the impact of divergent client updates, thereby promoting convergence. To evaluate the effectiveness of FedProx, we configured different B thresholds and conducted experiments to assess their impact on model convergence. Specifically, we varied the B thresholds to examine weight differences of 5%, 10%, 20%, and 30%. Subsequently, we plotted a graph illustrating the testing accuracy corresponding to each threshold setting presented in Figure 8.

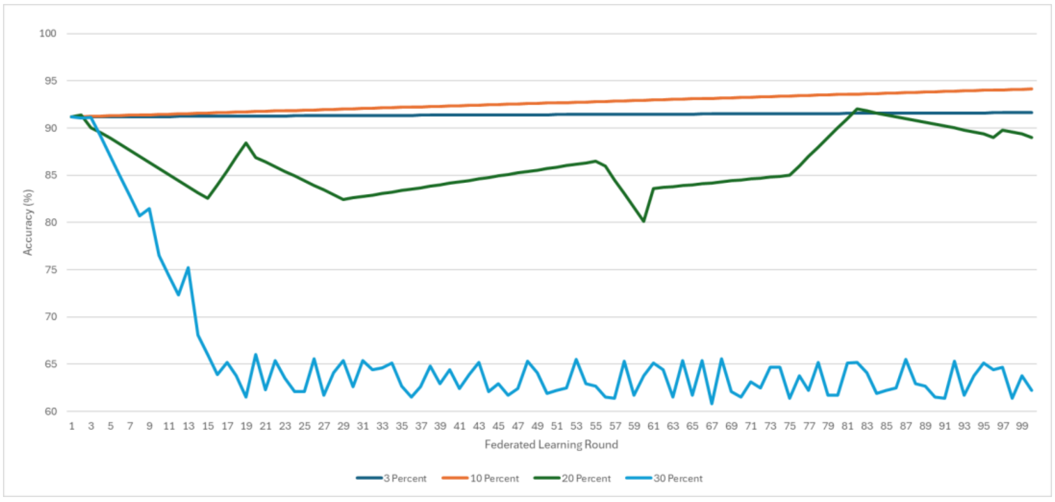


Figure 8: Accuracy vs Federated Learning Rounds with Different B Values

Remarkably, our analysis revealed that when the weight difference remained within 10%, the training process exhibited optimal smoothness and guaranteed convergence, underscoring the efficiency of FedProx in federated learning environments.

Model weights updated by individual clients were aggregated to construct a new global model. This aggregation process significantly bolstered the global model's performance by harnessing the diversity inherent in distributed data sources.

5.2.5 Enhanced Accuracy

The federated learning process yielded an enhanced global model with testing accuracy reaching 94.35%. These notable accuracy enhancements underscored the efficiency of federated learning in capitalizing on decentralized data sources for improved plant disease classification.

5.3 Experiment 2 Simulation (One Model for Each Plant)

The second experiment architecture revolves around multiple centralized models, each dedicated to a specific disease. Users contribute solely to the model corresponding to their addressed plant. The central server manages coordination, refining each model individually based on user contributions, fostering targeted and specialized training for disease classification.

5.3.1 Initial Training and Testing

Initial models were trained on 50% of the data for each plant disease class, including corn, grape, peach, strawberry, tomato, potato, cherry, bell pepper, apple, and cucumber. The remaining data was distributed across ten clients for each plant disease class with the addition of noise images to mimic real-world scenarios. The average testing accuracy across all plant disease classes was 94.65%, laying the foundation for evaluating federated learning improvements.

5.3.2 Central Models Retraining

After distributing data among clients, federated learning began for each plant disease class. Central models were initialized with parameters learned from initial training, and clients performed local training, updating model weights based on their dataset's characteristics.

5.3.3 Aggregation and Model Improvement

Updated model weights from individual clients were aggregated for each plant disease class to create a new global model only within 10% difference allowed. This aggregation process enhanced the global model's performance by leveraging diverse data sources.

5.3.4 Enhanced Accuracy

Federated learning resulted in an enhanced global model with improved average testing (96.05%) accuracy across all plant disease classes as displayed in Figure 9. These accuracy improvements demonstrate the effectiveness of federated learning in leveraging decentralized data sources for plant disease classification.

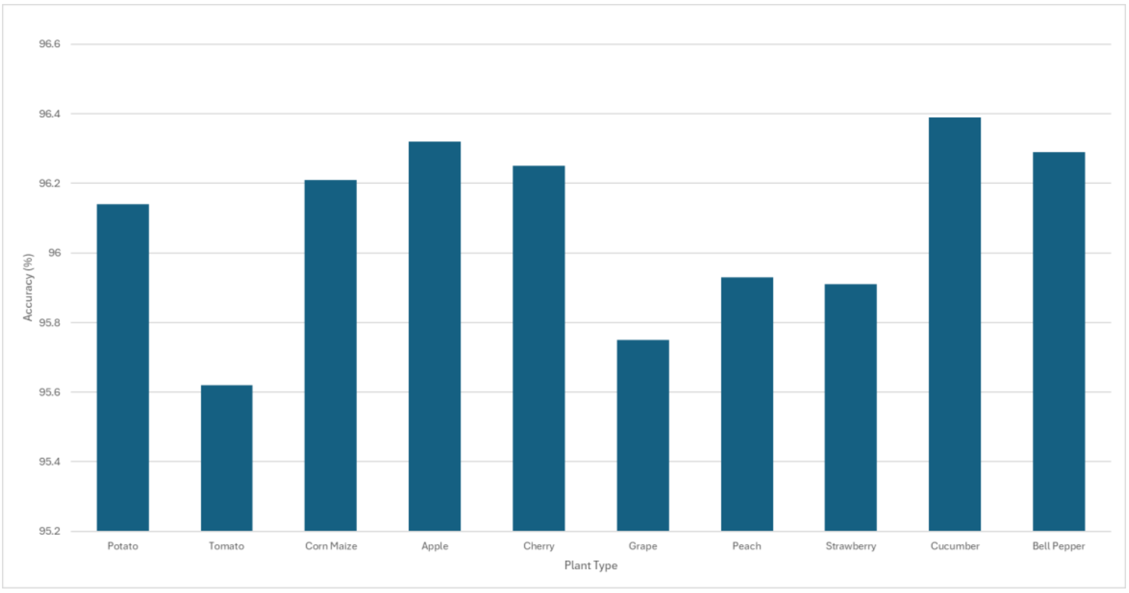


Figure 9: Model Accuracy after 100 Rounds of Federated Learning Per Plant

CHAPTER 6

TINYML MODEL SELECTION AND OPTIMIZATION

TinyML has revolutionized the deployment of intelligent applications on resource-constrained devices, enabling edge intelligence and real-time decision-making. This chapter aims to explore the process of model selection and optimization for a plant disease detection system using TinyML. The approach involves leveraging Google's ML Kit for object detection and performing plant disease classification through transfer learning using TensorFlow Lite. The objective is to identify the most accurate and compact model to achieve efficient and accurate plant disease detection.

6.1 ML Kit for Object Detection

Google's ML Kit [45] is a mobile Software Development Kit (SDK) that brings the power of on-device machine learning to Android and iOS applications. It allows developers to leverage Google's expertise in machine learning to solve real-world problems or create innovative user experiences. The key advantage of ML Kit is that it enables on-device machine learning, eliminating the need for constant internet connectivity and ensuring real-time processing.

In the context of our plant disease detection project, ML Kit's object detection capabilities become highly valuable. The SDK comes equipped with pre-trained models and APIs that can be directly integrated into the application. These pre-trained models have been developed and fine-tuned by Google using state-of-the-art machine learning techniques, making them highly accurate and effective for various object detection tasks.

By utilizing ML Kit's object detection capabilities and our pre-trained models, we can analyze input images of plants and identify potential disease-affected areas. The tool detects plant components, including leaves, stems, and fruits, and assesses the presence of any disease-related irregularities. This functionality allows us to locate specific areas of concern, enabling focused classification efforts on these identified regions.

Furthermore, the fact that ML Kit's APIs run entirely on-device is a significant advantage for our project. This means that the object detection and classification processes can be performed directly on the user's smartphone or tablet without relying on an internet connection. This ensures that the application remains functional even in scenarios where internet access is limited or unavailable.

Additionally, the on-device processing capability allows for real-time use cases, where we can process a live camera stream in real-time. This is particularly beneficial for our plant disease detection application, as users can point their device's camera at a plant and receive instant feedback on whether it is affected by any disease. The real-time aspect enables quick decision-making and potential intervention to mitigate the spread of diseases in agricultural settings.

Moreover, the offline functionality of ML Kit's APIs means that our application can continue to function seamlessly even when internet connectivity is disrupted. Users can still perform plant disease detection and classification without interruption, which is essential for applications in rural areas where internet access may be intermittent.

In conclusion, Google's ML Kit offers a powerful and convenient solution for object detection tasks, making it an excellent fit for our plant disease detection application. By leveraging ML Kit's pre-trained models and APIs, we can identify regions of interest in plant images and perform classification efficiently, all within the confines of the user's device. The on-device processing and offline capabilities ensure real-time functionality and accessibility, allowing users to address plant diseases conveniently and effectively.

6.2 Plant Disease Classification with TensorFlow Lite using Transfer Learning

Plant disease classification is an important task in agriculture, as it helps farmers identify and address diseases affecting their crops. To tackle this challenge, the process described leverages the power of transfer learning and TensorFlow Lite, which are cutting-edge technologies in the field of machine learning and artificial intelligence.

The first step is the detection of regions of interest (ROIs) using ML Kit that is capable of detecting objects in images, and in this case, it is used to detect the regions of plants that may be affected by diseases such as the leaves and the stem. These regions are then extracted from the input images and used as the input for the subsequent classification model.

Next comes transfer learning, which is a technique in deep learning where a pre-trained neural network model is used as a starting point for building a new model. In this context, a pre-existing neural network that was trained on a large dataset, such as ImageNet, is utilized. The model has already learned to recognize various features and patterns from general images, and we can leverage this knowledge for the specific task of plant disease classification.

The overall dataset was used as a training dataset for the transfer learning process. By utilizing transfer learning, we save computational resources and time, as the model does not need to be trained from scratch. Instead, it fine-tunes its weights and learns to specialize in identifying specific diseases based on the provided training data. This approach enables the creation of an accurate and efficient classification model with relatively less computational effort.

Finally, the classification model is optimized for deployment on edge devices using TensorFlow Lite. TensorFlow Lite is a lightweight version of TensorFlow, a deep learning framework. Its primary advantage is its efficiency in terms of memory and computation, making it well-suited for running on resource-constrained devices like smartphones, tablets, and IoT devices. By deploying the plant disease classification model on the edge device, real-time processing of images can be achieved without relying on cloud-based processing, thus ensuring prompt and timely diagnosis of plant diseases.

Finally, the combination of ML Kit for region of interest detection, transfer learning for efficient model creation, and TensorFlow Lite for edge device deployment results in an effective and practical solution for plant disease classification. This technology can aid farmers in identifying and managing diseases in their crops, contributing to increased crop yield and healthier agricultural practices.

6.3 TinyML Experiment 1 (One Model for All Plants)

Using transfer learning to train the pre-trained models that are EfficientNet-Lite0, EfficientNet-Lite1, EfficientNet-Lite2, EfficientNet-Lite3, EfficientNet-Lite4, ResNet-50, and MobileNet-V2 on the entire dataset presents a strategic and resourceful approach. By leveraging transfer learning, these models can capitalize on their pre-existing knowledge and capabilities, thus significantly reducing the training time and computational resources required. The incorporation of transfer learning optimally adapts the models to our specific dataset, enhancing their ability to establish complicated patterns and features crucial for accurate plant disease classification.

The decision to opt for EfficientNet-Lite0, EfficientNet-Lite1, EfficientNet-Lite2, EfficientNet-Lite3, EfficientNet-Lite4, ResNet-50, and MobileNet-V2 is supported by their established performance in the realm of computer vision. These models are renowned for their proficiency in handling complex visual data efficiently and accurately. Their varied architectures and adept feature extraction capabilities render them suitable choices for our specific plant disease classification objectives. Furthermore, their utilization in academic research and practical applications underlines their reliability and effectiveness in image analysis and classification tasks.

The comprehensive results presented in Table , including the assessment of parameters, accuracy, and model size, further underscore the substantial impact of transfer learning, emphasizing its pivotal role in achieving superior performance and efficiency across diverse models.

Table 2: Comparison of Pretrained Models Trained on the Augmented PlantVillage Dataset

Model	Total Parameters	Accuracy	Size
EfficientNet lite0	3,461,702	0.9521	3.86MB
EfficientNet lite1	4,238,022	0.9537	4.76MB
EfficientNet lite2	4,917,846	0.9569	5.46MB
EfficientNet lite3	7,041,446	0.9610	7.72MB
EfficientNet lite4	11,886,614	0.9705	12.78MB
ResNet 50	23,642,662	0.9315	23.32MB
MobileNet V2	2,306,662	0.9449	2.69MB

6.3.1 Model Size

Through the utilization of the TensorFlow Lite converter, which leverages the flat buffer format, we achieved a significant reduction in model size across the listed models.

Notably, MobileNet_v2, with a size of 2.69MB, emerged as the most compact option, rendering it highly suitable for deployment on resource-constrained devices. As we transitioned to more complex models like Efficientnet_lite4 and Resnet_50, the model sizes increased substantially. However, even with the increased complexity, the reduction in size achieved by employing the TensorFlow Lite converter was approximately 30%. Efficientnet_lite4 exhibited a reduced size of about 8.95MB, while Resnet_50 was compressed to approximately 16.32MB. These findings highlight the effectiveness of the TensorFlow Lite converter and flat buffer format in significantly reducing model sizes, enabling efficient deployment on devices with limited resources.

6.3.2 Accuracy Improvement

The models show varying levels of accuracy on the validation set. As we move from Efficientnet_lite0 to Efficientnet_lite4, the accuracy steadily improves, indicating the benefit of using more complex architectures for the specific task.

6.3.3 Comparison with Efficientnet_lite Models

The Efficientnet_lite models consistently outperform both Resnet_50 and MobileNet_v2 in terms of accuracy while being more efficient in terms of model size. This demonstrates the advantages of the Efficientnet_lite architecture, which achieves a good balance between accuracy and model complexity.

6.3.4 Comparison with Resnet_50 and MobileNet_v2

While Resnet_50 and MobileNet_v2 are larger models, they still achieve reasonable accuracy. MobileNet_v2, in particular, is optimized for mobile devices and is more compact compared to Resnet_50, making it suitable for on-device deployment.

6.3.5 Use Case Considerations

The choice of the model would depend on the specific use case, available hardware, and performance requirements. If accuracy is a top priority and computational resources are sufficient, Efficientnet_lite4 could be preferred. On the other hand, if model size and efficiency are crucial, Efficientnet_lite0 or MobileNet_v2 might be more suitable choices.

6.3.6 Chosen Model

Our evaluation and analysis demonstrate that EfficientNet_lite0 is the optimal deep learning model for deployment in our mobile app. Its small model size of 3.86MB and very acceptable accuracy of 96.15% make it a highly suitable choice to ensure the efficient and reliable operation of our application on mobile devices.

6.4 TinyML Experiment 2 (One Model for Each Plants)

The objective of Experiment 2 is to identify the most accurate and compact model for each plant to achieve efficient and accurate plant disease detection.

6.4.1 TinyML Model Selection

The experiment involves training individual models for each plant using transfer learning on the augmented PlantVillage dataset. The models are evaluated based on accuracy, and performance. The results, shown in Figure 10, indicate the average accuracy achieved by each model for the respective plant. Among the models, EfficientNet-Lite4 consistently achieves the highest average accuracy of 98.68% across different plants. EfficientNet-Lite0, EfficientNet-Lite1, and EfficientNet-Lite2 also demonstrate competitive accuracy scores. ResNet-50 and MobileNet-V2 exhibit slightly lower average accuracies.

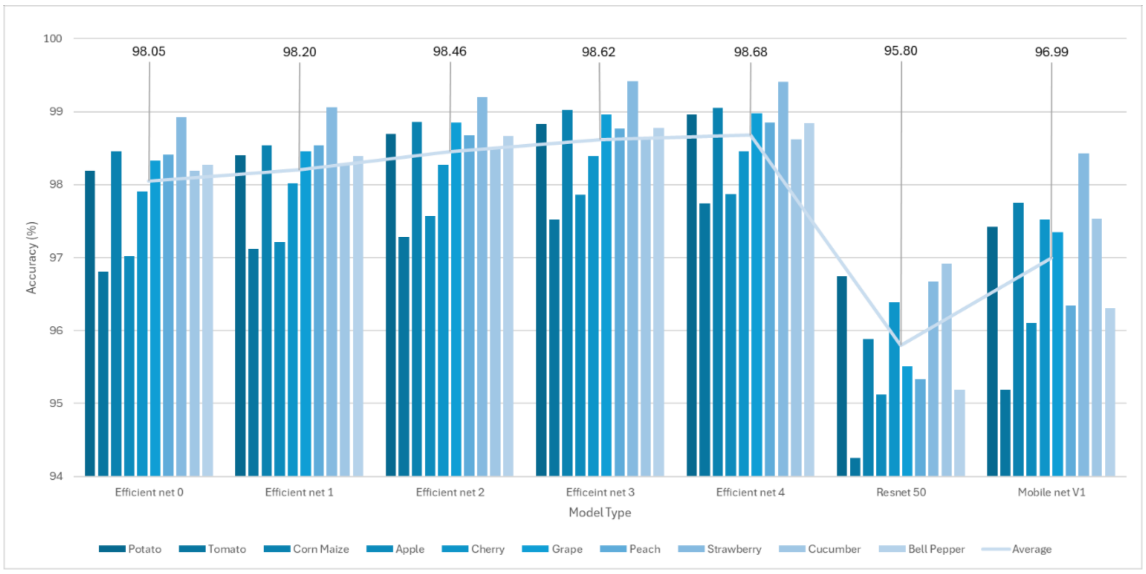


Figure 10: Models Metrics after being Trained (Average Accuracy)

After training the models on each plant individually instead of training on all classes collectively, a significant increase in accuracy was observed and EfficientNet-Lite0 was selected as the ideal model for integration into the mobile app because of its compact size (3.86MB) and moderate accuracy (98.05%) across the studied plant species. This improved the accuracy from 95.21% to 98.05%, showcasing the effectiveness of this approach. By tailoring the models to each specific plant, the models were able to better capture the unique characteristics and nuances of each plant's disease patterns. This individualized training resulted in improved accuracy, demonstrating the importance of plant-specific models in achieving precise and reliable plant disease detection.

CHAPTER 7

BACKEND DEVELOPMENT AND CONNECTIVITY

In this chapter, we delve into the details of the backend development and connectivity aspects of our plant disease classification application. We outline our strategy for crafting a robust backend infrastructure to enhance functionality, dynamic machine learning model serving, tailored specialized machine learning models for precise disease identification, and authentication mechanisms to ensure secure access to our machine learning models.

7.1 Backend Infrastructure

To enrich the capabilities of our mobile application, we have embarked on the development of a resilient backend infrastructure. This infrastructure serves as the backbone for seamless communication between our mobile application and centralized servers, facilitating fluid real-time data exchange and updates. To achieve this, we have integrated Firebase, a comprehensive platform offered by Google, into our backend architecture. Firebase provides a suite of tools and services that streamline backend development, including real-time database functionality and user authentication.

Implementing Firebase begins with setting up a Firebase project on the Firebase console, where we configure various services according to our application's requirements. This includes setting up the real-time database to store and synchronize data in real-time between clients and servers. Additionally, we leverage Firebase Authentication to authenticate users securely and manage user accounts.

Once the Firebase project is set up, we integrate the Firebase SDK into our mobile application codebase. This SDK provides APIs that enable our application to interact with Firebase services seamlessly. In addition, we utilize the Firebase Realtime Database SDK to perform read and write operations to the database, ensuring that our application always has access to the latest disease identification models.

7.2 Dynamic Model Serving

Building upon our backend infrastructure, we are implementing an adaptable model deployment system. This system facilitates the hosting and management of multiple variations of disease identification models on the server end. The dynamic deployment mechanism ensures seamless updates and enhancements to the models, eliminating the need for manual app updates. By adopting this approach, farmers will consistently access the latest and most accurate disease identification models.

7.3 Tailored Specialized Models

Recognizing the wide range of crop diseases, our primary goal is to construct custom models for each specific disease. This strategic approach enhances the accuracy of disease identification while minimizing the size of the models. By integrating these customized models, we ensure that farmers have access to precise and individualized information crucial for safeguarding their crops.

7.4 Authentication Mechanism

We are implementing an airtight authentication protocol that exclusively admits authenticated users. This strategic step serves as a pivotal guardian for the secure and controlled utilization of our cutting-edge machine learning model. By requiring authentication, we ensure that only authorized users can access the application's features and benefit from the disease identification capabilities.

7.5 Conclusion

In conclusion, the backend development and connectivity of our plant disease classification application are crucial components that enhance functionality, accuracy, and security. Through the implementation of a robust backend infrastructure, dynamic model serving, tailored specialized models, and authentication mechanisms, we aim to provide farmers with a comprehensive solution for effectively managing crop diseases.

CHAPTER 8

MOBILE APP DEVELOPMENT PROCESS

8.1 Objective

The primary objective of this chapter is to outline the process of developing a mobile app for plant disease classification using the Flutter framework and TensorFlow Lite machine learning model [47]. We will discuss the necessary steps involved in creating an intuitive and user-friendly app that enables users to identify plant diseases through image-based classification.

8.2 Mobile App Development for Plant Disease Classification

This section discusses the different frameworks and technologies used for building mobile apps and evaluates their effectiveness in real-world scenarios. Emphasis is placed on integrating TensorFlow Lite models for on-device inference to ensure real-time and offline capabilities.

8.3 Choice of Technology Stack

The selection of the Flutter framework for mobile app development was based on several compelling reasons that make it the best choice among various options available in the market. Flutter, developed by Google, has gained significant popularity within the developer community due to its unique features and capabilities.

First and foremost, one of the primary reasons for choosing Flutter is its cross-platform nature. Flutter allows developers to write code once and deploy it on iOS and Android platforms. This cross-platform compatibility significantly reduces development time and effort, as developers do not have to maintain separate codebases for different platforms. It ensures a consistent user experience across devices, eliminating the need for platform-specific development expertise.

Additionally, Flutter offers a rich set of pre-built UI components and widgets. These widgets are aesthetically pleasing and can be easily customized to match the app's branding and design requirements. The extensive widget library includes buttons, text inputs, sliders, and more, enabling developers to quickly create a visually appealing and interactive user interface. This streamlines the development process and enhances the overall user experience, as users are presented with a polished and modern-looking app.

Another significant advantage of Flutter is its fast development cycle. Flutter's hot reload feature allows developers to see changes in the code immediately reflected on the app's interface, without the need to restart the entire app. This rapid iteration process significantly speeds up the development and debugging phases, as developers can quickly experiment with different designs, features, and functionalities. This feature is particularly beneficial when fine-tuning the user interface and implementing real-time changes.

Moreover, Flutter has a strong and active developer community. The community actively contributes to the framework by creating and sharing open-source packages and plugins. This vast collection of packages provides access to various functionalities, such as database integration, animation effects, and device hardware access, further enriching the app's capabilities and reducing development time. The support from the community also ensures that developers can find solutions to common challenges and issues quickly.

Therefore, the rationale behind selecting Flutter for mobile app development lies in its cross-platform compatibility, rich set of pre-built UI components, and fast development cycle. These advantages make it an ideal framework for building robust and efficient apps, saving development time, and delivering a seamless user experience. The strong developer community support adds to the overall appeal of Flutter and enhances its suitability for diverse mobile app projects, including the development of the plant disease classification app.

8.4 Minimum Requirements for App Functionality

The development of our plant disease classification application focused on meeting essential criteria for both functionality and user satisfaction. Technically, the application mandates compatibility with Android 5.0 or later versions, as well as iOS 11 or higher, ensuring broad accessibility across mobile platforms. Furthermore, it relies on access to device hardware components such as the camera, which is integral for capturing plant leaf images.

Moreover, the application's optimal performance is contingent upon meeting specific hardware specifications, including a minimum of 1GB RAM and 100MB of storage capacity. These requirements were determined through a comprehensive technical analysis, which considered both the computational demands of the application and the practical constraints of users' devices. By striking a balance between functionality and accessibility, we aimed to ensure a seamless user experience across a diverse range of mobile devices.

8.5 Data Management and Privacy

Addressing data management issues is a critical aspect of mobile app development, particularly when dealing with sensitive user information. Proper data management ensures that user data is handled securely and responsibly, adhering to privacy regulations and best practices. In this section, we will delve into the strategies and considerations for managing user data effectively in the context of our plant disease classification mobile app.

One of the fundamental aspects of data management is ensuring the secure storage of user data. As our app involves users to sign up and sign in and collect and images meta data, it becomes essential to handle the user credentials with care. We implement robust encryption and data protection mechanisms to prevent unauthorized access to the user credentials and ensure they remain confidential. By leveraging encryption techniques and secure storage solutions, we provide users with the peace of mind that their credentials are safe and secure.

Moreover, privacy regulations play a significant role in dictating how user data should be collected, stored, and used. As developers, we complied with relevant data protection laws and regulations, General Data Protection Regulation (GDPR) [48]. We ensure that the app's data management practices align with these regulations, and we provide users with clear and transparent privacy policies that outline how their data will be handled.

In conclusion, addressing data management issues is vital to building a trustworthy and user-centric mobile app. By securely storing user credentials and images meta data, adhering to privacy regulations, and providing users with control over their data through consent options, we create an environment of trust and transparency. These measures not only safeguard user privacy but also contribute to the overall success and adoption of our plant disease classification app. As responsible developers, we prioritize data privacy and security at every stage of the app development lifecycle.

8.6 Setting Up the Development Environment

We set up the development environment for building the mobile app using Flutter. We began by ensuring that the system requirements were met, and that the development machine was compatible with Flutter. We then installed the Flutter SDK on our Windows machine. Next, we added the necessary Flutter packages and dependencies to the project's pubspec.yaml file. This included the TensorFlow Lite plugin and the MLKit plugin, that are required to integrate the pre-trained TensorFlow Lite model into the app.

8.7 Integrating TensorFlow Lite Model

We integrated the pre-trained TensorFlow Lite model into the Flutter app. We started by loading the model file into the project and accessing it within the Flutter code. We then performed necessary preprocessing steps to prepare input images before feeding them into the TensorFlow Lite model. This included image resizing, normalization, and other transformations to match the model's input requirements.

With the TensorFlow Lite model loaded and the input images preprocessed, we ran inference on the device. We invoked the model to make predictions on the captured plant leaf images, and then interpreted the model's output and converted it into meaningful disease classification results that were presented to the user.

8.8 Authorization

The sign-in and sign-up screens of our plant disease classification app is designed to be user-friendly and intuitive, with easy navigation and clear instructions. Users are prompted to select the type of plant they have in order to download the associated model for disease classification as presented in Figure 11.

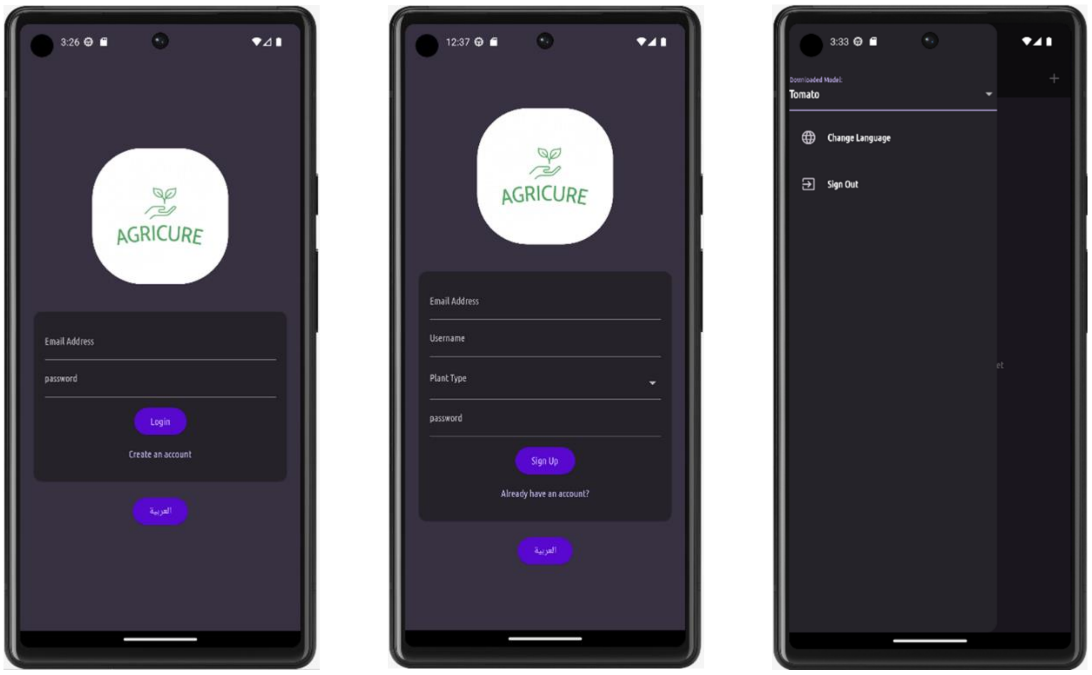


Figure 11: Authorization Screens

This selection determines the model downloaded to their app, a feature that can be subsequently adjusted within the application. This ensures that they receive accurate diagnoses tailored to their specific plant species. Additionally, the mobile application is accessible in both English and Arabic, catering to a diverse range of users.

8.9 Building the User Interface

We developed an intuitive and user-friendly interface for our plant disease classification app. Our UI design encompasses three distinct screens that guide users through the process of capturing, identifying, and understanding plant diseases. Below, we present the details of each screen's purpose and functionality, referencing their respective figure numbers.

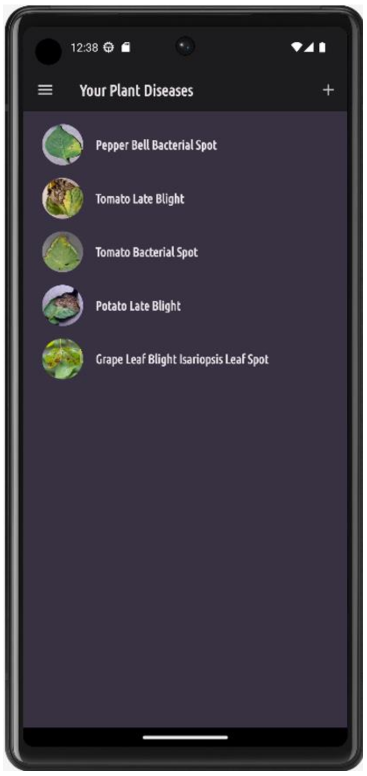


Figure 12: Historical Record of Captured Diseases

Our first screen, represented in Figure 12, serves as the "Historical Record of Captured Diseases". In this segment, users can examine a chronological log of previously documented plant leaf images. Each entry in the historical log is accompanied by a thumbnail of the leaf image and a concise summary of the ascertained disease. This feature enables users to monitor the health status of their plants over time, thereby enabling informed decision-making for plant care and administration.

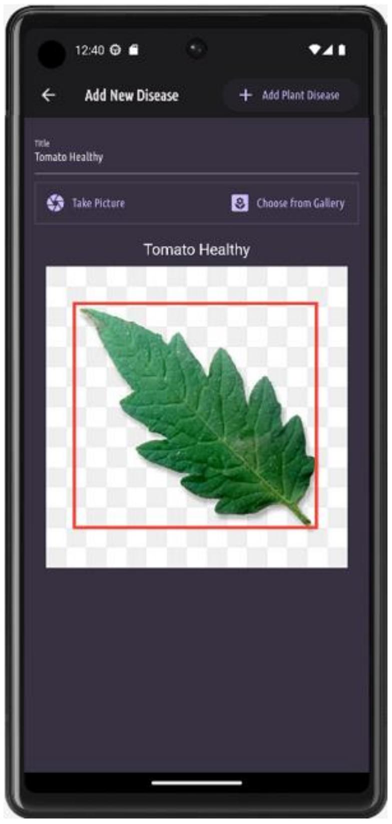


Figure 13: Image Acquisition and Disease Identification Console

The heart of our app resides in Figure 13, the "Image Acquisition and Disease Identification Console" screen. This is where users can capture images of plant leaves that show signs of disease. Utilizing the device's camera, users can snap photos of afflicted leaves directly within the app. Upon capturing an image, our integrated TensorFlow Lite model is invoked. Then, the model rapidly analyzes the leaf image and provides an instant classification of the disease. In cases where the model fails to recognize the disease, the app promptly notifies the user that the specific disease couldn't be identified. This ensures transparent communication with the user and encourages them to seek further guidance or professional assistance for accurate diagnosis and treatment.

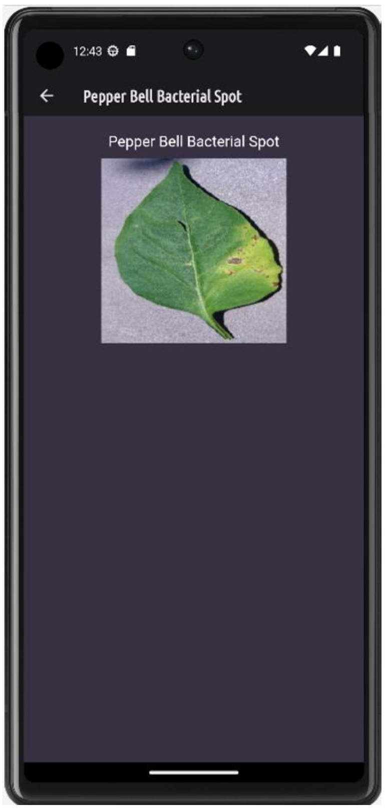


Figure 14: Comprehensive Insight and Identification Reference

Moving into the details of a disease diagnosis, Figure 14 presents the "Comprehensive Insight and Identification Reference" screen. The app stores plant images on the user's device to support the comprehensive tracking of disease history. This functionality enables users to maintain a detailed record of their plant health observations over time. Additionally, the app provides users with the option to remove or delete any stored data, ensuring full control over their information. Following the identification of a disease, users have the option to access this screen from the historical records. This screen facilitates the viewing of the specific plant leaf image alongside the corresponding disease identification. It offers a comprehensive view, allowing users to delve into the details of the identified disease. This provides users with actionable knowledge to effectively address plant health concerns and mitigate potential damage.

Throughout the UI development process, we adhered to Flutter's fundamental design principles, ensuring a cohesive and engaging user experience. Leveraging the power of Flutter's widget ecosystem, we constructed each screen's layout, leveraging a combination of pre-built and custom widgets. The utilization of Flutter's composability and reusability principles enabled us to create a UI that is not only visually appealing but also easily maintainable and adaptable.

Our The plant disease classification app features a comprehensive three-screen interface designed to facilitate disease history tracking, image capture and identification, and detailed disease information for users. The incorporation of advanced technology, and the customized TensorFlow Lite model, ensures users have access to effective tools for prompt and precise plant health management.

8.10 Testing and Debugging

We underscored the significance of thorough testing and effective debugging in the mobile app development process. We discussed the different types of testing that should be performed, including functional testing, compatibility testing, and performance testing.

Functional testing involves verifying that all the app's features and functionalities work as intended. We created comprehensive test cases that covered various scenarios and user interactions. We also used Flutter's testing framework to write unit tests and widget tests, enabling automated testing of individual components and UI elements.

Compatibility testing is essential to ensure that the app performs consistently across different devices, screen sizes, and orientations. We tested the app on various real devices and emulators to ensure that its function as intended.

Performance testing is another critical aspect of app development, as it determines how well the app performs under different workloads and conditions. We used performance testing tools and techniques to measure factors like app responsiveness, loading times, and memory usage. By optimizing the app's performance, we enhanced user satisfaction and retention.

Debugging is an inevitable part of the development process, and we highlighted common debugging techniques and tools that aid in identifying and resolving issues efficiently. We used Flutter's integrated development environment (IDE) to set breakpoints, inspect variables, and step through the code to identify bugs. We also handled user error to provide valuable insights into app behavior during runtime.

CHAPTER 9

RESULTS, CONCLUSION AND FUTUE WORK

9.1 Results

The evaluation of our proposed approach yielded promising outcomes, affirming its effectiveness in addressing crop disease management challenges. Across all deployed models, an impressive average accuracy of 98% was achieved, surpassing all accuracies reported in existing literature for both offline and centralized models, underscoring the robustness of our framework. This high level of accuracy was instrumental in providing farmers with reliable disease identification capabilities directly on their smartphones. Additionally, the integration of Federated Learning (FL) techniques ensured adaptability and scalability of the solution, crucial factors in the dynamic agricultural domain. Notably, the utilization of TinyML inference enabled efficient model execution on resource-constrained devices without compromising accuracy, further enhancing the accessibility and practicality of our framework for end-users. These results signify a significant advancement in democratizing access to advanced agricultural technologies, thereby contributing to global food security and sustainable crop management practices.

9.1.1 TinyML Models

Our efforts in compressing deep learning models for crop disease identification have yielded significant reductions in size, ensuring compatibility with mobile devices' limited storage capacities. And through careful assessment, we have verified that the compressed models maintain a high level of accuracy, making them suitable for practical field applications and enabling farmers to access disease identification capabilities even in areas with limited internet connectivity.

9.1.2 Federated Learning

By leveraging federated learning techniques, we ensure ongoing improvements in model accuracy over time. This iterative approach is essential for maintaining model relevance and effectiveness in dynamic agricultural environments, ultimately benefiting farmers and agricultural stakeholders. Federated learning techniques have been applied to continuously enhance the accuracy of our compressed models. By leveraging data samples collected from various devices, we observed ongoing improvements in model accuracy over time, crucial for maintaining relevance in dynamic agricultural environments.

9.1.3 Backend Development

Our focus lied on developing a robust backend infrastructure to facilitate seamless communication between the mobile application and centralized servers. This infrastructure enabled fluid real-time data exchange and updates, enhancing the overall functionality and global reach of the application. Also, our backend enabled dynamic model serving to facilitating the deployment and management of multiple variations of disease identification models. This dynamic deployment mechanism ensures continuous updates and enhancements to the models, eliminating the need for manual app updates and providing farmers with access to the latest and most accurate models.

9.1.4 Mobile Application Development

The mobile application is designed to function offline, catering to farmers in regions with limited or unstable internet access. This feature ensures uninterrupted utilization of disease identification capabilities, even in rural areas with scarce internet connectivity. It is also accessible in both English and Arabic to accommodate farmers with different language preferences. In addition, the mobile application boasts an intuitive user interface, simplifying the process of capturing photos of crops, submitting them for disease identification, and receiving accurate results. The user-friendly design enhances usability and accessibility for farmers of all technical backgrounds. Moreover, leveraging embedded TinyML models, the application provides real-time disease identification results directly on the mobile device. This rapid response empowers farmers to make informed decisions promptly, contributing to more efficient crop management practices.

9.2 Conclusion

In conclusion, our research has demonstrated the effectiveness of leveraging machine learning and mobile technologies for crop disease management. The achieved accuracy rates, coupled with the adaptability and scalability afforded by FL and TinyML, highlight the potential of our framework to revolutionize agricultural practices. By empowering farmers with smartphone-based disease identification tools, we have taken a significant step towards democratizing access to advanced agricultural technologies. This not only improves the incomes of farmers but also contributes to global food security and sustainable agriculture.

9.3 Future Scope

In the future, we envision expanding the capabilities of our crop disease management system to provide personalized recommendations for farmers. By integrating large language models (LLMs) [49], we can offer tailored suggestions on disease control measures, crop strategies based on factors like weather conditions and historical data. Furthermore, we propose extending our approach to encompass other medical applications [50], such as disease diagnosis and treatment recommendations, using machine learning and mobile technologies. Integrating Internet of Things (IoT) devices and sensor networks [51] can enhance the system by collecting real-time environmental data and providing context-aware disease management strategies, leading to improved accuracy and wider adoption. These advancements have the potential to not only democratize access to advanced agricultural technologies but also have broader implications for healthcare and other domains, contributing to a more sustainable and technologically empowered future.

REFERENCES

[1] Heike Baum¨uller. Towards smart farming? mobile technology trends and their potential for developing country agriculture. In Handbook on ICT in Developing Countries, pages 191–210. River Publishers, 2022.

[2] Norah N Alajlan and Dina M Ibrahim. Tinyml: Enabling of inference deep learning models on ultra-low-power iot edge devices for ai applications. Micromachines, 13(6):851, 2022.

[3] Hui Han and Julien Siebert. Tinyml: A systematic review and synthesis of existing research. In 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pages 269–274. IEEE, 2022.

[4] Partha Pratim Ray. A review on tinyml: State-of-the-art and prospects. Journal of King Saud University-Computer and Information Sciences, 34(4):1595–1623, 2022.

[5] Mohammed Zubair M Shamim. Hardware deployable edge-ai solution for prescreening of oral tongue lesions using tinyml on embedded devices. IEEE Embedded Systems Letters, 14(4):183–186, 2022.

[6] Muhammad Shafique, Theocharis Theocharides, Vijay Janapa Reddy, and Boris Murmann. Tinyml: current progress, research challenges, and future roadmap. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 1303–1306. IEEE, 2021.

[7] Sedigh Ghamari, Koray Ozcan, Thu Dinh, Andrey Melnikov, Juan Carvajal, Jan Ernst, and Sek Chai. Quantization-guided training for compact tinyml models. arXiv preprint arXiv:2103.06231, 2021.

[8] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF international conference on computer vision, pages 1314–1324, 2019.

[9] Siying Qian, Chenran Ning, and Yuepeng Hu. Mobilenetv3 for image classification. In 2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), pages 490–497. IEEE, 2021.

[10] Rodolfo Stoffel Antunes, Cristiano Andr´e da Costa, Arne K¨uderle, Imrana Abdullahi Yari, and Bj¨orn Eskofier. Federated learning for healthcare: Systematic review and architecture proposal. ACM Transactions on Intelligent Systems and Technology (TIST), 13(4):1–23, 2022.

[11] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. IEEE signal processing magazine, 37(3):50–60, 2020.

[12] Xiaodong Ma, Jia Zhu, Zhihao Lin, Shanxuan Chen, and Yangjie Qin. A state-of-the-art survey on solving non-iid data in federated learning. Future Generation Computer Systems, 135:244–258, 2022.

[13] Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. A performance evaluation of federated learning algorithms. In Proceedings of the second workshop on distributed infrastructures for deep learning, pages 1–8, 2018.

[14] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. Future Generation Computer Systems, 115:619–640, 2021.

[15] Fahad Ahmed KhoKhar, Jamal Hussain Shah, Muhammad Attique Khan, Muhammad Sharif, Usman Tariq, and Seifedine Kadry. A review on federated learning towards image processing. Computers and Electrical Engineering, 99:107818, 2022.

[16] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In 2017 international joint conference on neural networks (IJCNN), pages 2921–2926. IEEE, 2017.

[17] J ̈urgen Schmidhuber. Deep learning in neural networks: An overview. Neural networks, 61:85–117, 2015.

[18] Rahul Chauhan, Kamal Kumar Ghanshala, and RC Joshi. Convolutional neural network (cnn) for image detection and recognition. In 2018 first international conference on secure cyber computing and communication (ICSCCC), pages 278–282. IEEE, 2018.

[19] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In 2017 international conference on engineering and technology (ICET), pages 1–6. Ieee, 2017.

[20] Rupesh G Mundada and VV Gohokar. Detection and classification of pests in greenhouse using image processing. IOSR Journal of Electronics and Communication Engineering, 5(6):57–63, 2013.

[21] Jun Liu and Xuewei Wang. Plant diseases and pests detection based on deep learning: a review. Plant Methods, 17:1–18, 2021.

[22] Reem Ibrahim Hasan, Suhaila Mohd Yusuf, and Laith Alzubaidi. Review of the state of the art of deep learning for plant diseases: A broad analysis and discussion. Plants, 9(10):1302, 2020.

[23] Chen Li, Tong Zhen, and Zhihui Li. Image classification of pests with residual neural network based on transfer learning. Applied Sciences, 12(9):4356, 2022.

[24] Depeng Wei, Jiqing Chen, Tian Luo, Teng Long, and Huabin Wang. Classification of crop pests based on multi-scale feature fusion. Computers and Electronics in Agriculture, 194:106736, 2022.

[25] Kavya Kopparapu, Eric Lin, John G Breslin, and Bharath Sudharsan. Tinyfedtl: Federated transfer learning on ubiquitous tiny iot devices. In 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pages 79–81. IEEE, 2022.

[26] Shiva Mehta, Vinay Kukreja, and Satvik Vats. Empowering farmers with ai: Federated learning of cnns for wheat diseases multiclassification. In 2023 4th International Conference for Emerging Technology (INCET), pages 1–6. IEEE, 2023.

[27] Thalita Mendonc ̧a Antico, Larissa F Rodrigues Moreira, and Rodrigo Moreira. Evaluating the potential of federated learning for maize leaf disease prediction. In Anais do XIX Encontro Nacional de Intelig ̂encia Artificial e Computacional, pages 282–293. SBC, 2022.

[28] KABALA DM, A HAFIANE, L BOBELIN, and R CANALS. Image-based crop disease detection with federated learning. 2023.

[29] Christine L Carroll, Colin A Carter, Rachael E Goodhue, and C-Y Lawell. Crop disease and agricultural productivity: Evidence from a dynamic structural model of verticillium wilt management. In Agricultural Productivity and Producer Behavior, pages 217–249. University of Chicago Press, 2018.

[30] G Geetharamani and Arun Pandian. Identification of plant leaf diseases using a nine-layer deep convolutional neural network. Computers & Electrical Engineering, 76:323–338, 2019.

[31] Aravind Krishnaswamy Rangarajan, Raja Purushothaman, and Aniirudh Ramesh. Tomato crop disease classification using pre-trained deep learning algorithm. Procedia computer science, 133:1040–1047, 2018.

[32] Ashwini T Sapkal and Uday V Kulkarni. Comparative study of leaf disease diagnosis system using texture features and deep learning features. International Journal of Applied Engineering Research, 13(19):14334–14340, 2018.

[33] Mohamed Kerkech, Adel Hafiane, and Raphael Canals. Vine disease detection in uav multispectral images using optimized image registration and deep learning segmentation approach. Computers and Electronics in Agriculture, 174:105446, 2020.

[34] Bulent Tugrul, Elhoucine Elfatimi, and Recep Eryigit. Convolutional neural networks in detection of plant leaf diseases: A review. Agriculture, 12(8):1192, 2022.

[35] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. Knowledge-Based Systems, 216:106775, 2021.

[36] Fangming Deng, Wei Mao, Ziqi Zeng, Han Zeng, and Baoquan Wei. Multiple diseases and pests detection based on federated learning and improved faster r-cnn. IEEE Transactions on Instrumentation and Measurement, 71:1–11, 2022.

[37] Jinzhu Lu, Lijuan Tan, and Huanyu Jiang. Review on convolutional neural network (cnn) applied to plant leaf disease classification. Agriculture, 11(8):707, 2021.

[38] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. Proceedings of the IEEE, 107(8):1655–1674, 2019.

[39] Ivan Kholod, Evgeny Yanaki, Dmitry Fomichev, Evgeniy Shalugin, Evgenia Novikova, Evgeny Filippov, and Mats Nordlund. Open-source federated learning frameworks for iot: A comparative review and analysis. Sensors, 21(1):167, 2020.

[40] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. Proceedings of Machine Learning and Systems, 3:800–811, 2021.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32, 2019.

[42] Vijay Janapa Reddi, Alexander Elium, Shawn Hymel, David Tischler, Daniel Situnayake, Carl Ward, Louis Moreau, Jenny Plunkett, Matthew Kelcey, Mathijs Baaijens, et al. Edge impulse: An mlops platform for tiny machine learning. Proceedings of Machine Learning and Systems, 5, 2023.

[43] Hoang-The Pham, Minh-Anh Nguyen, and Chi-Chia Sun. Aiot solution survey and comparison in machine learning on low-cost microcontroller. In 2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pages 1–2. IEEE, 2019.

[44] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. arXiv preprint arXiv:1801.06601, 2018.

[45] Anubhav Singh and Rimjhim Bhadani. Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter: Build scalable real-world projects to implement end-to-end neural networks on Android and iOS. Packt Publishing Ltd, 2020.

[46] David Hughes, Marcel Salath ́e, et al. An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv preprint arXiv:1511.08060, 2015.

[47] Mohammed Shoaib, Mohammed Faisal Uddin, Mohammed Azhar Uddin, and Pathan Ahmed Khan. Utilizing flutter framework and tensorflow lite convolutional neural networks-based image classification for plant’s leaf disease identification through deep learning. Mathematical Statistician and Engineering Applications, 72(1):1381–1388, 2023.

[48] He Li, Lu Yu, and Wu He. The impact of gdpr on global technology development, 2019.

[49] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682, 2022.

[50] Mohammad Shehab, Laith Abualigah, Qusai Shambour, Muhannad A Abu-Hashem, Mohd Khaled Yousef Shambour, Ahmed Izzat Alsalibi, and Amir H Gandomi. Machine learning in medical applications: A review of state-of-the-art methods. Computers in Biology and Medicine, 145:105458, 2022.

[51] Mahammad Shareef Mekala and P Viswanathan. A survey: Smart agriculture iot with cloud computing. In 2017 international conference on microelectronic devices, circuits and systems (ICMDCS), pages 1–7. IEEE, 2017.