Electronic Letters on Computer Vision and Image Analysis 24(2):285-311, 2025

Binary Feature Map-Splitting Architecture (BFMSA): A Computationally Efficient Approach for Plant Leaf Disease Classification

Amer Tabbakh*+ and Soubhagya Sankar Barpanda+

+ *School of Computer Science and Engineering, VIT-AP University, Amaravati, India.*
Received 29th of May, 2024; accepted 2nd of December, 2025

Abstract

This paper presents the Binary Feature Map-Splitting Architecture (BFMSA), a lightweight mechanism that reduces trainable parameters while maintaining or improving classification accuracy. BFMSA introduces a hierarchical splitting across L levels; for $L = 3$, feature maps are processed by $2^L = 8$ reduced-capacity branches whose outputs are concatenated before the final classifier. We instantiate BFMSA in a custom CNN and integrate it into pre-trained backbones (Inception-V3, ResNet50, VGG16, VGG19) for plant leaf-disease classification on the PlantVillage dataset. Across models, BFMSA yields an average parameter reduction of **87–88%** with consistent gains in validation accuracy—rising from 94.79% to **98.31%** for the custom CNN and from 65.56% to **79.14%** for ResNet50. We also report deployment-oriented metrics (FLOPs, latency, memory), provide a closed-form parameter formulation, and conduct an ablation over L. The results indicate that BFMSA acts as a structural regularizer that mitigates overfitting while substantially shrinking model size, making it well-suited to resource-constrained edge devices in smart agriculture.

Key Words: Trainable Parameters, Transfer Learning, Deep Learning, Image Processing, and Plant Disease.

1 Introduction

Plants have a crucial role in reducing the impact of climate change. They capture carbon dioxide from the atmosphere, store it in their tissues, and release oxygen through the process of photosynthesis. They also help to regulate local climates, prevent soil erosion, and protect against flooding. In addition, plants play a vital role in our everyday lives as they provide food, oxygen, and raw materials for various industries. The health of plants is crucial for their growth and productivity, and diseases affecting plants can significantly reduce crop quantity and quality. Moreover, with the increasing demand for food and other resources, it is more important than ever to maintain the productivity and health of agricultural yield. The detection of plant diseases is therefore a critical aspect of sustainable agriculture and food security. Therefore, early and accurate diagnosis of plant diseases is crucial to control their spread, reduce their impact on crop production, and prevent the loss of entire crops while minimizing the use of pesticides. Accurate disease detection also helps in the development of effective control strategies and the implementation of appropriate management practices. In the past, plant disease classification

*Correspondence to: mr.amertabbakh@gmail.com

Recommended for acceptance by Angel D. Sappa
https://doi.org/10.5565/rev/elcvia.2217
ELCVIA ISSN: 15108-5097
Published by Computer Vision Center / Universitat Autonoma de Barcelona, Barcelona, Spain

was typically done manually by human experts, which was time-consuming and often subject to human error. With Deep Learning (DL) and image processing technologies, it is possible to train algorithms to recognize specific patterns and features in images of plants, such as discoloration, spots, or lesions, that are indicative of various diseases. This advancement has enabled the development of automated systems for detecting and classifying plant diseases quickly and accurately for large amounts of data. Such systems have the potential to greatly improve crop yields by enabling farmers to identify and treat diseased plants before the diseases can spread and cause significant damage. Overall, the combination of DL and image processing technologies has revolutionized the field of plant disease classification, making it more efficient and accurate than ever before, and opening up new opportunities for research and development in this important area. Beyond model design, preprocessing (normalization, geometric stabilization, and feature selection) is widely used to handle illumination/viewpoint variability and to reduce computational load [1, 2, 3]; robust feature extraction further motivates invariance to real-world conditions [4, 5, 6]

In contrast, DL models can be computationally intensive due to a vast amount of parameters that require significant computation to train. During the training process, a DL model must go through many iterations to optimize its parameters to reduce the loss function. This process involves calculating gradients, which are used to update the model's weights, and can be computationally expensive. Additionally, the vast data is required to train a DL model, which makes the training process even more computationally demanding. To address these computational challenges, dedicated hardware such as Tensor Processing Units (TPUs) and Graphics Processing Units (GPUs) have been developed. These hardware platforms are designed to speed up the training process and reduce the time required for training a DL model. In addition, cloud-based services such as Google Cloud Platform (GCP) and Amazon Web Services (AWS) provide access to high-performance computing resources that can be used to train DL models. Despite these efforts to minimize the computational demands of DL models, they remain computationally intensive and require significant resources to train effectively. This can be a barrier to entry for individuals and organizations that do not have access to specialized hardware or cloud computing resources.

The main contribution of the proposed model is to address the challenge of the computational cost associated with DL models. Specifically, the model seeks to minimize the number of trainable parameters in a DL model by altering its structure, while maintaining the same level of accuracy. This is a significant contribution, as DL models are often computationally expensive and require substantial amounts of data and computational resources to train, which can hinder their applicability to a diverse range of domains. To address these challenges, we propose a Binary Feature Map-Splitting Architecture (BFMSA), a novel approach that strategically splits intermediate feature maps into binary partitions, enabling parallel yet lightweight feature extraction. This design reduces the number of parameters and floating-point operations (FLOPs), lowers memory requirements, and accelerates inference while preserving classification accuracy. Unlike conventional CNN-based architectures, BFMSA can be seamlessly integrated with existing CNN-based models, making it a scalable solution for computationally constrained environments.

The subsequent sections of this paper are organized in a structured manner to provide a clear and cohesive presentation of the research. Section 2 offers a brief review of pertinent literature on the topic under investigation. Section 4 elaborates on the workflow and methodology adopted in the research, while Section 5 presents the experimental parameters in detail. The outcomes of the experiments, along with a comprehensive discussion of their implications, are presented in Section 6. Finally, Section 7 provides a summary of the key findings of the study.

2 Related Work

Recent research has shown significant attention to utilizing ML and DL to determine plant diseases. This is due to the potential benefits of automated classification of plant diseases, which can aid farmers in promptly and effectively treating these diseases, ultimately leading to higher crop yields and reduced financial losses. The articles [7, 8, 9, 10] provided an overview of newly developed techniques as well as enhancements to

existing algorithms used for identifying plant diseases. Their main objective was determining the most effective method for classifying plant diseases. Various techniques of artificial intelligence (AI) have been implemented to identify and classify diverse diseases of plants, including betel vines in [11], rice [12], pomegranate [13], olive [14], cassava [15, 16, 17, 18], tomato [19, 20, 21, 22], potato [23, 24, 25, 26, 27], orange [28], Guava [29, 30], citrus [31], apple leaf [32, 33], tea leaf [34, 35], and mango [36].

Several techniques have been developed using Machine Learning (ML) to accurately categorize plant leaf diseases. One such approach by Sharif and colleagues [37] used an optimized weighted segmentation technique on the image acquired during the stage of preprocessing to detect leaf lesions. After detecting the lesions, they extracted features based on color (HSI, LUV, LAB, and HSV), texture (18 GLCM features), and geometric. Principal component analysis (PCA) was used to select most significant attributes of citrus leaf diseases, subsequently a Multi-Class Support Vector Machine (M- SVM) for classification. They concluded that improving the lesion area's contrast can enhance the accuracy of segmentation. Bhargava and Bansal [38] devised an approach aimed at segmenting lesions on monochromatic apples. This segmentation was accomplished through the utilization of two algorithms: k-means clustering and the 'Otsu' strategy. After segmenting the image, features are extracted including color coherence, Gabor wavelet, 13 statistical and textural, and 14 geometric features. They trained the linear SVM model and observed better performance compared to other kernels. Singh and Kaur [25] showed how the K-means algorithm can identify regions affected by infection on leaves and extract lesion features using GLCM. The feature extraction method is regarded as a form of dimensionality reduction. Two studies have proposed models for leaf feature extraction. Pahurkar and Deshmukh [39] combined convolutional feature sets, GLCM, color map, and edge map to create a featured ensemble. To extract the most significant features, Particle Swarm Optimization (PSO) is utilized, and GA is employed to tune its parametric variant classification techniques in order to optimize variance. Tabbakh and Barpanda [40] developed a machine- learning model that employs statistical features and the GLCM method along with wavelet transform to extract various leaf feature combinations. The extracted features were applied to train six machine-learning models, including AdaBoost and SVM, which were then compared. The GLCM method was modified to extract leaf-part features while ignoring the image's background. The SMOTE technique is used to address imbalanced dataset issue. Using LGBM, they attained the highest accuracy of 94.39%.

In contrast, DL demonstrated its effectiveness in image classification tasks, surpassing traditional techniques like Random Forests and SVM. However, to train DL models effectively, huge datasets are required. To address the issue of limited dataset size, various techniques have been used, such as creating new samples by cropping lesion portions from original images as described by Barbedo [41], or augmenting the dataset using methods like image flipping, scaling, rotation, color augmentation, noise injection, gamma correction, and PCA, as done by Geetharamani and Pandian [42]. The results have shown that these data augmentation techniques can enhance model performance. Researchers have employed an image segmentation method to accurately identify infected areas on leaves while minimizing processing effort. Dey et al. [11] utilized the HVS color space in their dataset. Their findings revealed that the hue component demonstrated significant efficacy in accurately detecting decayed sections of a leaf, effectively concealing the background and other non-affected regions of the leaf. Chaturvedi et al. [44] introduce an adapted firefly algorithm for fruit leaf segmentation. It uses multilevel thresholding techniques - including fuzzy, Tsallis, and Kapur's entropy methods. While Singh and Misra [43] achieved lesion segmentation by concealing green pixels. Johnson et al. [26] devised a system for automatically detecting patches on potato leaves of blight disease in field conditions. The system employs a residual network with a Mask R-CNN model as the backbone. The method effectively distinguishes the diseased regions on the potato leaf from the background, mitigating the potential confusion encountered in binary classification outcomes. They conducted experiments using various color spaces by reshape original dataset from RGB to HSL, HSV, LAB, XYZ, and YCrCb which resulted in a distinct model for individual color spaces. Meanwhile, Hassan and Maji [45] introduced an innovative deep- learning method incorporating residual connections and inception layers, achieving a remarkable 70% reduction in the parameter count using depthwise separable convolution and significantly decreasing computational costs. Khamparia et al. [23] used a convolutional encoder network that combined CNN with the encoding portion of an autoencoder model to extract meaningful features.

They obtained an accuracy of 97.50% after 100 iterations. In another study, Liang et al. [46] proposed an approach that uses ResNet50 and a residual- based approach to identify diseases through a network- based system, incorporating the architecture of Shuffle- Net-network v2 to minimize computational complexity. Tabbakh and Barpanda [47] present TLMViT, a hybrid model that integrates a pre-trained model and a vision transformer for feature extraction in plant disease classification. Experimental results demonstrate its accuracy and superiority over approaches based on pre-trained models, highlighting usage of transfer learning model for extracting initial features of plant and the vision transformer's ability to extract deep features. DL models often contain millions of learnable weights, raising compute and memory cost during training. Schuler et al. [48] adapted an Inception-V3 architecture to incorporate two branches that process achromatic and chromatic data separately, leveraging the separation of these types of information. They subsequently partitioned the branches, resulting in a reduction of the number of trainable parameters and computational workload to approximately half of the original values, accomplished by modified layers. As a result, their approach attained a high level of accuracy, with a state-of-the-art classification accuracy of 76.91% for the Cropped-PlantDoc dataset, and 99.48% for the Plant Village dataset. Liu et al. [49] replaced the fully connected layers of the conventional AlexNet model with an Inception network in their CNN architecture for apple leaf disease detection, resulting in a reduction of trainable parameters. This substitution significantly decreased the number of parameters that needed to be trained, resulting in lower storage requirements. Wang and colleagues [50] proposed an improved multi-scale residual (Multi-scale ResNet) model based on ResNet18. Their approach involved several modifications to the network architecture, including adding a multi-scale feature extraction module, changing the residual layer connection method, decomposing the large convolution kernel, and using group convolution operations. These changes resulted in a significant reduction in computing overhead, storage space, and model parameters. As a result, the Multi-scale ResNet achieved an accuracy rate of 95.95% on their self-collected dataset and 93.05% on the PlantVillage dataset of seven real ecological diseases. Although prior techniques for identifying plant leaf diseases have achieved compelling outcomes, they still encounter various limitations.

However, many prior approaches rely on high-dimensional deep features fed into separate ML classifiers, which increases computational complexity and latency. Transfer learning still requires retraining large fully connected heads, inflating parameter count and memory, while training from scratch is even heavier and prone to overfitting on limited data. In light of these limitations, we propose an alternative architectural approach aimed at reducing parameters and FLOPs, all while maintaining accuracy.

3 Motivation & Objectives

Deep learning has revolutionized plant disease classification by enabling automated, high-accuracy diagnosis from leaf images. However, state-of-the-art models often rely on millions of trainable parameters, leading to high computational cost, memory footprint, and inference latency—key barriers to deployment in resource-constrained agricultural environments such as mobile or edge devices. Moreover, excessive model capacity increases the risk of overfitting, particularly on imbalanced datasets like PlantVillage, where certain disease classes are underrepresented.

While transfer learning mitigates the need to train deep feature extractors from scratch, it does not resolve the inefficiency of the final classifier head. Pre-trained models typically retain large, fully connected layers that introduce millions of trainable parameters, undermining efforts toward lightweight deployment. There is a pressing need for architectural innovations that intrinsically reduce complexity—without sacrificing accuracy or generalization.

To address this gap, we propose the **Binary Feature Map-Splitting Architecture (BFMSA)**, a novel, general-purpose module that hierarchically splits intermediate feature maps into parallel, lightweight branches. Each branch processes reduced-capacity features, while concatenation before the final classifier preserves global representational power. Unlike post-hoc compression techniques (e.g., pruning, quantization), BFMSA is a design-time structural optimization that acts as a built-in regularizer, promoting modular feature learning and enhancing generalization.

The primary objectives of this work are:

1. To design and validate BFMSA as a parameter-efficient architectural module that reduces trainable parameters by **87–88%** across diverse backbones while maintaining or improving classification accuracy.
2. To integrate BFMSA with both custom CNNs (trained from scratch) and pre-trained models (Inception-V3, ResNet50, VGG16/19), demonstrating its versatility across training paradigms.
3. To significantly reduce computational overhead—including FLOPs, inference latency, and memory footprint—making deep models viable for edge deployment in smart agriculture.
4. To demonstrate that BFMSA mitigates overfitting, especially in deeper networks (e.g., ResNet50), by acting as a structural regularizer, evidenced by improved validation accuracy and reduced train–validation gaps.
5. To conduct an ablation study over splitting depth L and derive a closed-form parameter formula, providing practical design guidelines for accuracy–efficiency trade-offs.

These objectives are rigorously evaluated on the PlantVillage dataset. Results show consistent accuracy improvements—ranging from +0.36% (Inception-V3) to +13.58% (ResNet50)—alongside dramatic efficiency gains: inference time and memory reduced by approximately $8\times$. BFMSA thus offers a practical, drop-in solution for efficient, scalable, and deployable plant disease diagnosis systems.

4 Methodology

This paper presents a **Binary Feature Map-Splitting Architecture (BFMSA)** for constructing efficient CNN models in plant disease classification. The proposed BFMSA was designed to significantly reduce the number of trainable parameters by hierarchically splitting feature maps across parallel branches, while preserving representational capacity through scalable and modular processing.

BFMSA is a **general-purpose architectural module** that can be applied in two key scenarios:

- **Custom-Designed CNNs** trained from scratch, where splitting is applied progressively during feature extraction.
- **Transfer Learning Frameworks**, where a pre-trained model serves as a fixed feature extractor and BFMSA is applied post-extraction to split and process the final feature maps efficiently.

4.1 Data Collection and Image Augmentation

We used the PlantVillage dataset (Kaggle), which contains 15,648 images of healthy and diseased plant leaves spanning 16 classes across five plant species: Apple, Corn (maize), Grape, Cherry (including sour), and Peach. Sample leaf images are presented in Figure 1. The class distribution, summarized in Table 1, ranges from 275 to 2,297 images per class, revealing significant class imbalance. This imbalance heightens the risk of overfitting—especially for underrepresented classes—where models may memorize training patterns instead of learning generalizable features, resulting in poor performance on unseen data.

To mitigate this, data augmentation is widely adopted to increase data diversity and improve robustness. Mohanty et al. [51] showed that augmentation significantly improves DL performance on PlantVillage by increasing tolerance to intra-class variability. Similarly, Shorten and Khoshgoftaar [52] surveyed augmentation strategies and emphasized that geometric transformations—such as shearing, zooming, and shifting—not only expand the effective dataset size but also act as a data-dependent regularizer that reduces overfitting.

Inspired by these findings, we applied the following augmentation techniques during training, including rotation ($\pm 25^\circ$), width and height shifts (10%), shearing (20%), zooming (20%), and horizontal flipping as

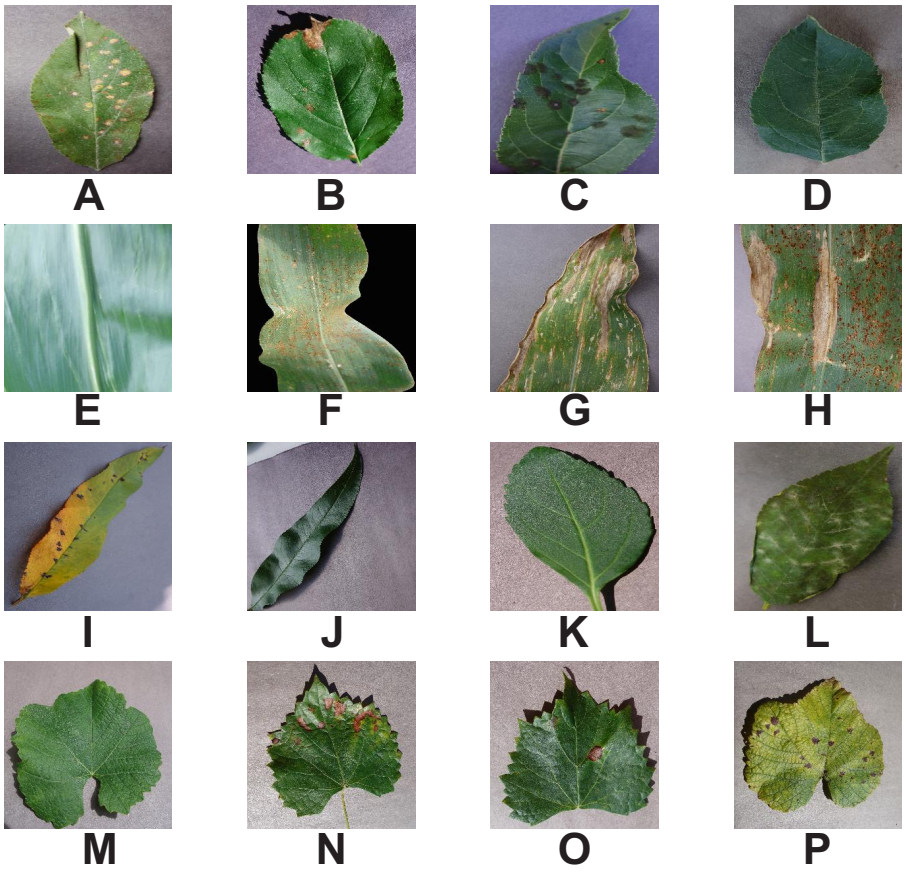


Figure 1: Example images of plant leaves from the PlantVillage dataset. A: Apple Cedar Apple Rust, B: Apple Black Rot, C: Apple Scab, D: Apple Healthy, E: Corn Healthy, F: Corn Common Rust, G: Corn Gray Leaf Spot, H: Corn Northern Leaf Blight, I: Peach Bacterial Spot, J: Peach Healthy, K: Cherry Healthy, L: Cherry Powdery Mildew, M: Grape Healthy, N: Grape Esca (Black Measles), O: Grape Black Rot, P: Grape Leaf Blight.

detailed in Table 5. Random rotations and horizontal flips promote orientation invariance, while zooming and shifting improve robustness to variations in leaf scale and position. Together, these transformations simulate real-world imaging conditions and enable the model to learn discriminative, invariant features, enhancing its reliability in plant disease classification.

4.2 Baseline Architecture (Traditional Topology)

Let us consider a traditional convolutional neural network (CNN) architecture as the baseline. Given an input RGB leaf image $I \in \mathbb{R}^{H \times W \times 3}$, where $H = W = 128$, the model processes the image through a series of convolutional and pooling layers, followed by fully connected layers for classification.

The baseline architecture consists of $C_L = 4$ convolutional stages:

- **Conv1**: 32 filters of size 5×5
- **Conv2**: 64 filters of size 3×3
- **Conv3**: 128 filters of size 3×3
- **Conv4**: 256 filters of size 3×3

Table 1: Distribution of PlantVillage dataset classes used in the proposed architecture.

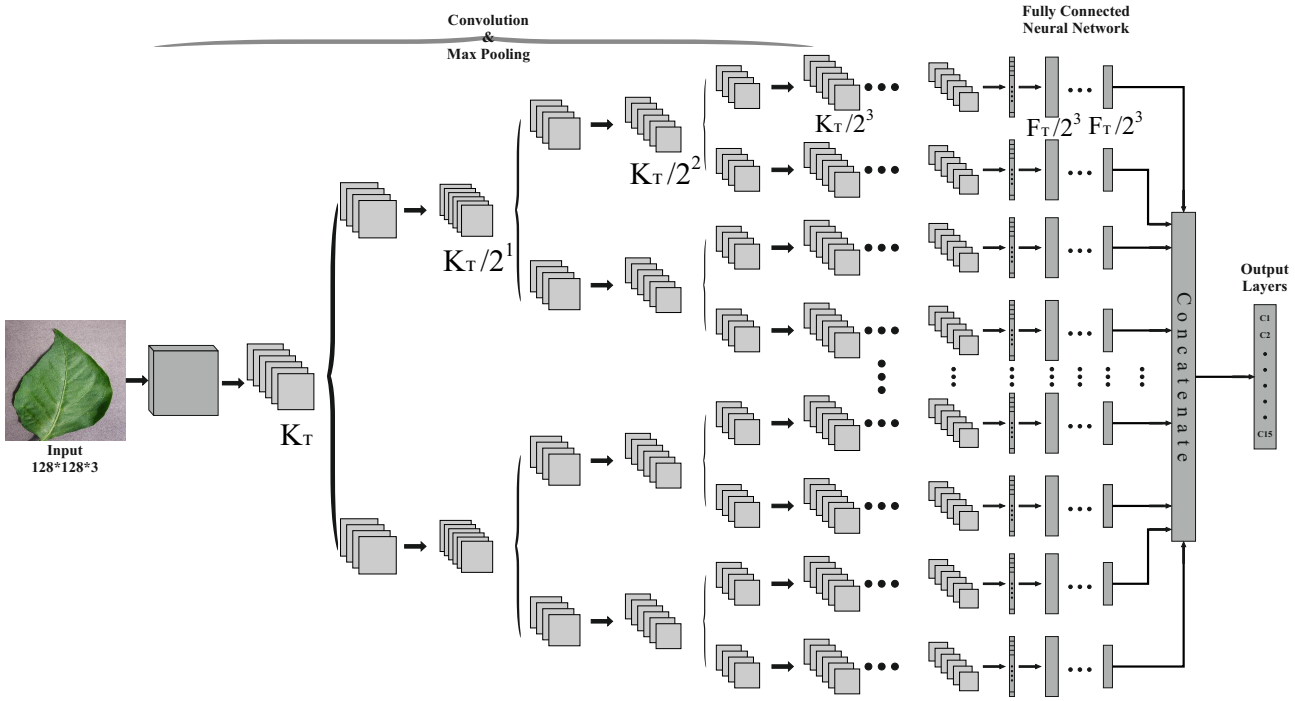
Sl. No.	Class Name	Number of Images
1	Apple Healthy	1,645
2	Apple Cedar Apple Rust	275
3	Apple Black Rot	621
4	Apple Scab	630
5	Peach Healthy	360
6	Peach Bacterial Spot	2,297
7	Cherry (Including Sour) Powdery Mildew	1,052
8	Cherry (Including Sour) Healthy	854
9	Grape Leaf Blight (Isariopsis Leaf Spot)	1,076
10	Grape Healthy	423
11	Grape Esca (Black Measles)	1,383
12	Grape Black Rot	1,180
13	Corn (Maize) Northern Leaf Blight	985
14	Corn (Maize) Healthy	1,162
15	Corn (Maize) Common Rust	1,192
16	Corn (Maize) Cercospora Leaf Spot (Gray Leaf Spot)	513
Total		**15,648**

Each convolutional layer is followed by a max-pooling layer (2×2, stride 2). Whereas each max-pooling layer reduces the spatial dimensions by half. Starting from 128×128, the feature map sizes become 64×64 after Conv1, 32×32 after Conv2, 16×16 after Conv3, and 8×8 after Conv4.

After the final convolutional stage, the feature maps are flattened and passed through three fully connected layers:

- First: 1024 units
- Second: 512 units
- Output: 16 units (corresponding to the number of plant disease classes)

This baseline configuration results in approximately $17,701,200$ trainable parameters.



K_T = The number of filters in the corresponding layer (stage) in the Baseline Architecture.

Figure 2: End-to-end framework of the BFMSA-based CNN backbone for plant leaf disease classification

4.3 BFMSA for Custom CNN Architectures

We adapt this architecture using the proposed **Binary Feature Map-Splitting Architecture (BFMSA)**. As depicted in Figure 2, this proposed topology introduces a hierarchical splitting mechanism based on the splitting level L. Let ℓ denote the index of the current convolutional stage, where $\ell = 1$ for **Conv1**, $\ell = 2$ for **Conv2**, $\ell = 3$ for **Conv3**, and $\ell = 4$ for **Conv4**. The splitting level L is defined as:

$$L = \ell - 1 \tag{1}$$

The splitting level L corresponds to the current convolutional stage: $L = 0$ for **Conv1**, $L = 1$ for **Conv2**, $L = 2$ for **Conv3**, and $L = 3$ for **Conv4**.

At each stage, the number of filters in each branch is scaled down according to:

$$K_{\text{BFMSA}} = \frac{K_T}{2^L} \tag{2}$$

where K_T is the number of filters in the corresponding layer of the traditional architecture.

Similarly, in the classification phase, the number of units in each fully connected layer per branch is scaled down by the same factor:

$$F_{\text{BFMSA}} = \frac{F_T}{2^L} \tag{3}$$

where F_T is the number of units in the corresponding dense layer of the traditional model.

This overall architecture consists of two primary components: the feature extraction phase, which involves hierarchical splitting of feature maps with K_{BFMSA}-sized filters, and the classification phase, where scaled-down, distributed fully connected layers (with F_{BFMSA} units) process the split pathways. The details of these components are described below.

4.3.1 Feature Extraction Phase

The feature extraction phase of the BFMSA-based CNN applies hierarchical feature map splitting across convolutional stages, starting after Conv1. At each stage, the number of filters per branch is scaled down by a factor of 2^L, where $L = \ell - 1$ and ℓ is the current stage index. The process is detailed as follows:

1. **Conv1 (No Splitting)**: The input image I is passed through the first convolutional layer with 32 filters (5×5), followed by max-pooling, yielding 32 feature maps of size 64×64. At this stage, no splitting has been applied, so the network operates with a single branch. The value $L = 0$ reflects the absence of any prior split.

2. **Splitting before Conv2** ($L = 1$): The 32 feature maps are split into 2 branches, each receiving 16 feature maps. Each branch processes its input through a local **Conv2** layer with:

$$K_{\text{BFMSA, Conv2}} = \frac{64}{2^1} = 32 \text{ filters } (3 \times 3) \tag{4}$$

   Each branch thus produces 32 feature maps after convolution and max-pooling.

3. **Splitting before Conv3** ($L = 2$): Each of the two existing branches is further split, resulting in $2^2 = 4$ total branches. Each new branch receives half of the previous feature maps ($16 \rightarrow 8$). For **Conv3**, the number of filters per branch is:

$$K_{\text{BFMSA, Conv3}} = \frac{128}{2^2} = 32 \text{ filters } (3 \times 3) \tag{5}$$

   Each of the 4 branches applies 32 filters independently, followed by max-pooling.

4. **Splitting before Conv4** ($L = 3$): Prior to the final convolutional stage, each of the 4 branches is split again, yielding $2^3 = 8$ independent branches. Each branch now receives 8 feature maps. For **Conv4**, the number of filters per branch is:

$$K_{\text{BFMSA, Conv4}} = \frac{256}{2^3} = 32 \text{ filters } (3 \times 3) \tag{6}$$

   Each branch applies 32 filters and performs max-pooling, resulting in 32 feature maps per branch.

After the final convolutional stage (Conv4), the network yields 8 independent branches, each producing 32 feature maps of size 8×8. This spatial dimension follows from the initial input size (128×128) and four downsampling operations via max-pooling. A per-branch comparison of filter counts between the traditional and BFMSA architectures is provided in Table 2. These hierarchical feature representations are processed independently in the subsequent classification phase.

Table 2: Filter count comparison per stage: Traditional vs. BFMSA (per branch)

Stage	L	2^L	**Traditional Filters**	**BFMSA Filters per Branch**
Conv1	0	1	32	$32/2^0 = 32$
Conv2	1	2	64	$64/2^1 = 32$
Conv3	2	4	128	$128/2^2 = 32$
Conv4	3	8	256	$256/2^3 = 32$

Note: The BFMSA filter count per branch is calculated as $K_T/2^L$. The constant value of 32 in this design is a result of the specific choices for K_T and L.

4.3.2 Classification Phase

The classification phase processes each of the $2^L = 8$ branch outputs independently through a series of dense layers, followed by concatenation and final classification. The steps are as follows:

1. **Flattening**: The 32 feature maps in each branch are flattened into a 1D vector. Given a spatial size of $S \times S$, here 8×8, the flattened dimension per branch becomes $8 \times 8 \times 32 = 2048$.

2. **First Dense Layer (F_{BFMSA})**: Each branch processes its flattened vector through a local dense layer. The number of units per branch is scaled down according to the splitting level $L = 3$:

$$F_{\text{BFMSA}} = \frac{F_T}{2^L} = \frac{1024}{2^3} = 128 \tag{7}$$

   where $F_T = 1024$ is the number of units in the corresponding layer of the traditional architecture. Thus, each branch applies a transformation to produce $F_{\text{BFMSA}} = 128$ output units. This significant reduction per-pathway contributes to overall parameter efficiency.

3. **Second Dense Layer (F_{BFMSA})**: The output from the first dense layer is passed to the second dense layer, again with scaled width:

$$F_{\text{BFMSA}} = \frac{F_T}{2^L} = \frac{512}{2^3} = 64 \tag{8}$$

   where $F_T = 512$ is the number of units in the traditional model. Each branch now outputs $F_{\text{BFMSA}} = 64$ units, maintaining proportionality with the original architecture while drastically reducing intermediate computation.

4. **Concatenation of Branch Outputs**: After the second dense layer, the outputs from all 8 branches are concatenated along the feature dimension:

$$F_{\text{concat}} = 8 \times 64 = 512 \tag{9}$$

   This recovers the exact number of units present in the corresponding layer of the traditional architecture (512), ensuring compatibility with the subsequent output layer.

5. **Final Classification Layer (Output)**: The concatenated F_T-unit vector (where $F_T = 512$) is fed into the final dense layer, which maps to the 16 output units corresponding to the number of plant disease classes. This layer is shared across the entire network and remains unchanged from the baseline:

$$\text{Output: } 512 \rightarrow 16 \text{ (class logits)}$$

This design ensures that:

- Each branch operates with significantly reduced capacity, lowering memory and computational cost.
- The concatenation step restores full representational power before the final classification, preserving the model's ability to discriminate between fine-grained disease categories.
- The total number of parameters in the dense layers is substantially reduced due to the distributed, scaled-down design.

As shown in Table 3, the BFMSA architecture achieves parameter efficiency during feature processing while preserving full capacity at decision time through concatenation. This makes it highly suitable for deployment in edge computing scenarios where computational resources are limited, yet high accuracy is required.

Table 4 provides a detailed comparison of trainable parameters between the traditional CNN and the proposed BFMSA-based architecture. The values in this table are computed using the generalized parameter

Table 3: Fully Connected layer configuration: Traditional vs. BFMSA (per branch)

Stage	**Traditional Architecture**	**BFMSA Architecture**	
		(per branch)	**After Concatenation**
F1	1024	$1024/2^L = 128$	-
F2	512	$512/2^L = 64$	$864 = 512$
Output	16	16	

Table 4: Trainable parameters comparison between traditional CNN and proposed BFMSA-based CNN

Phase	**Layer**	**Output Shape**	**Traditional Params**	***L***	**Branches**	**BFMSA Total Params**
Feature Extraction	Image	$128 \times 128 \times 3$	–	–	–	–
	Conv1	$64 \times 64 \times 32$	2,432	0	1	2,432
	Conv2	$32 \times 32 \times 64$	18,496	1	2	9,280
	Conv3	$16 \times 16 \times 128$	73,856	2	4	18,560
	Conv4	$8 \times 8 \times 256$	295,168	3	8	37,120
Classification	FC1	1024	16,778,240	–	8	2,098,176
	FC2	512	524,800	–	8	66,048
	Output	16	8,208	–	1	8,208
	Total		**17,701,200**			**2,239,824**

calculation framework introduced in Section 4.6, which decomposes the total trainable parameters into two components: the feature extraction phase and the classification phase.

The main concept of BFMSA involves the hierarchical division of feature maps across L levels. In this case, we used L = 3 levels, which resulted in up to 8 parallel branches at the final convolutional stage. In each branch, both the number of filters and the number of dense units are reduced by a factor of 2^L, which significantly decreases the complexity of each pathway.

As shown, while the traditional model uses full-width convolutions (e.g., 64 filters in Conv2), the proposed model processes smaller feature subsets in parallel branches, applying only 64/2=32 filters per branch at L=1 . Despite this reduction, the total effective capacity is preserved through multi-branch processing. Crucially, the classification phase scales the fully connected layers proportionally: the first dense layer uses only 128 units per branch (vs. 1024 in baseline), but across 8 branches, the concatenated representation restores full dimensionality before the final output layer.

The proposed BFMSA architecture achieves a significant reduction in model complexity. As shown in Table 4, the total number of trainable parameters is reduced from $17,701,200$ in the traditional CNN to just $2,239,824$ in the proposed model. This corresponds to a parameter reduction of approximately 87.3%, calculated as:

$$\text{Reduction} = \frac{17,701,200 - 2,239,824}{17,701,200} \times 100\% \approx 87.3\%. \tag{10}$$

This dramatic decrease highlights the efficiency of the hierarchical feature map splitting strategy, making the proposed model highly suitable for deployment on resource-constrained edge devices in agricultural applications.

4.4 BFMSA for Transfer Learning Models

A transfer learning (TL)-based CNN model is designed using the $BFMSA$ as the second proposed topology, as depicted in Figure 3. In this approach, the pre-trained model (e.g., Inception-V3, VGG-16, VGG-19, or ResNet-50) serves as a fixed feature extractor, with all its layers frozen during training. As a result, no trainable parameters are introduced during the feature extraction phase.

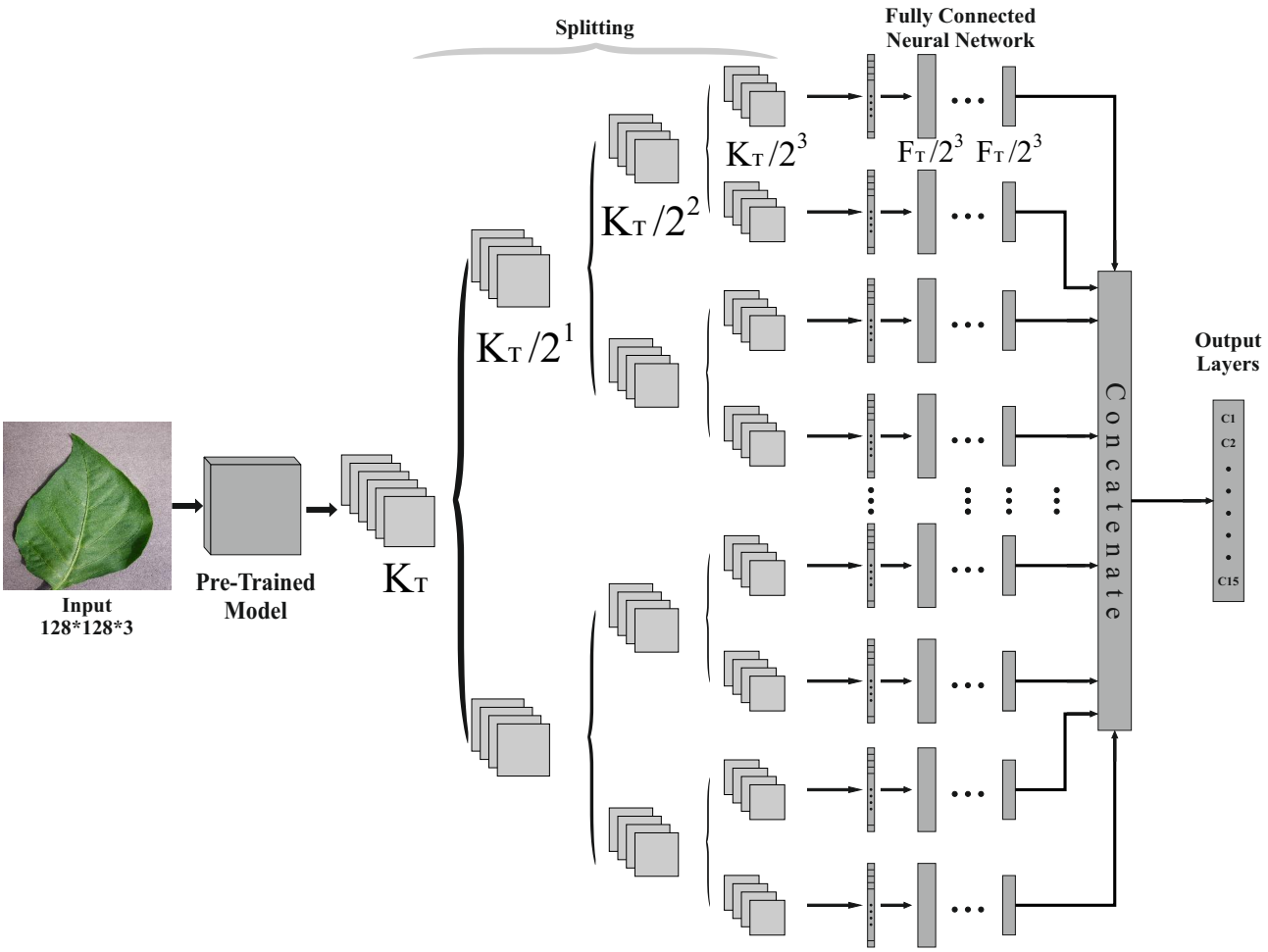


Figure 3: End-to-end framework of the BFMSA-based TL backbone for plant leaf disease classification

An input RGB plant image $I \in \mathbb{R}^{256\times256\times3}$ is first passed through the pre-trained network, which generates a high-level feature map of dimensions $H' \times W' \times k$. For consistency across models, the spatial dimensions are standardized to 8×8, resulting in feature maps of size $8 \times 8 \times k$, where $k = 2048$ for Inception-V3 and ResNet-50, and $k = 512$ for VGG-16 and VGG-19.

The BFMSA module is applied to this extracted feature map to reduce the computational load in the subsequent classification phase. The splitting process is performed hierarchically across $L = 3$ levels:

- **Level 1** ($\ell = 1$): The original feature map is split along the channel dimension into **2 branches**, each containing half of the channels, i.e., $8 \times 8 \times (k/2)$.
- **Level 2** ($\ell = 2$): Each of the 2 branches is independently split again into two, resulting in **4 branches** total, each with dimensions $8 \times 8 \times (k/4)$.
- **Level 3** ($\ell = 3$): Each of the 4 branches is again split into two, yielding **8 branches**, each with dimensions $8 \times 8 \times (k/8)$.

At each level ℓ, the number of branches doubles, following 2^ℓ, and the depth (number of channels) of each branch is halved compared to the previous level. This hierarchical splitting reduces the dimensionality of the data processed in each pathway, which directly lowers the number of trainable parameters in the downstream classification layers.

After splitting, the classification phase proceeds identically to the first proposed topology. Each of the $2^L = 8$ branches is processed independently:

1. The feature maps in each branch are flattened into a 1D vector.
2. The vector is passed through a series of fully connected layers (denoted as F1, F2, etc.), with the number of units in each layer scaled down by a factor of $2^L = 8$ compared to the traditional model.
3. The outputs from all 8 branches are concatenated into a single 1D vector.
4. This concatenated vector is fed into the final output layer (Dense with Softmax activation) to produce the 16-class prediction.

This design leverages the powerful feature extraction of pre-trained models while significantly reducing the number of trainable parameters in the classifier head through distributed, lightweight processing. As demonstrated in Section 4.6, this leads to up to an 85% reduction in trainable parameters, making the model highly efficient for edge deployment in smart agriculture applications.

4.5 Hierarchical Splitting with Depth L

$BFMSA$ is controlled by a single hyperparameter: the splitting depth L. This parameter determines both the number of parallel branches and the width of the layers in each branch, enabling a scalable trade-off between model efficiency and accuracy.

For a given value of L: - The feature maps are split across L levels, resulting in 2^L independent branches by the final convolutional stage. - From the first split onward, all subsequent layers (convolutional and fully connected) are applied independently in each branch, with their width (number of filters or dense units) scaled down by a factor of 2^L.

Specifically:

- $L = 0$**:** No splitting occurs. The model operates as a standard single-branch CNN, identical to the traditional architecture. All layers use the full width: K_T filters and F_T units.

- $L = 1$**:** One level of splitting is applied, resulting in $2^1 = 2$ branches. From the first split onward (i.e., from Conv2), each branch uses:

$$K_{\text{BFMSA}} = \frac{K_T}{2^1}, \quad F_{\text{BFMSA}} = \frac{F_T}{2^1}$$

For example, if the traditional model uses 64 filters in Conv2, each branch uses $64/2 = 32$ filters.

- $L = 2$**:** Two levels of splitting are applied, resulting in $2^2 = 4$ branches. From the second split onward (i.e., from Conv3), each branch uses:

$$K_{\text{BFMSA}} = \frac{K_T}{2^2}, \quad F_{\text{BFMSA}} = \frac{F_T}{2^2}$$

For example, if the traditional model uses 128 filters in Conv3, each branch uses $128/4 = 32$ filters.

- $L = k$**:** In general, for any $L = k$, the model is split into 2^k branches. From the k-th stage onward, each branch uses:

$$K_{\text{BFMSA}} = \frac{K_T}{2^k}, \quad F_{\text{BFMSA}} = \frac{F_T}{2^k}$$

This hierarchical scaling ensures that each branch processes a reduced-capacity pathway, significantly lowering computational cost.

After the final dense layer, the outputs from all 2^L branches are concatenated into a single vector of dimension $2^L \times F_{\text{final}}$, which is then fed into the final classification layer (in our case 16 units). This restores the full representational capacity of the original model at the decision boundary.

4.6 Trainable Parameter Calculation in BFMSA

To generalize the computation of trainable parameters in the proposed $BFMSA$ architecture, we divide the total into two primary components: the feature extraction phase and the classification phase.

4.6.1 Feature Extraction Phase

The feature extraction phase consists of $C_L = 4$ convolutional stages, indexed by $\ell = 1, 2, 3, 4$. At each stage ℓ, a splitting level L is defined as:

$$L = \ell - 1$$

This results in 2^L parallel branches at stage ℓ. For example:

- At $\ell = 1$ (Conv1): $L = 0$, so $2^L = 1 \rightarrow$ no splitting.
- At $\ell = 2$ (Conv2): $L = 1$, so $2^L = 2 \rightarrow 2$ branches.
- At $\ell = 3$ (Conv3): $L = 2$, so $2^L = 4 \rightarrow 4$ branches.
- At $\ell = 4$ (Conv4): $L = 3$, so $2^L = 8 \rightarrow 8$ branches.

At each stage, the number of filters per branch is scaled down as:

$$K_{\text{BFMSA}} = \frac{K_T}{2^L}$$

where K_T is the number of filters in the corresponding layer of the traditional architecture.

For a convolutional layer with kernel size $k \times k$, input channels C_{in}, and output channels K_{BFMSA}, the number of trainable parameters per branch is:

$$P_{\text{conv}} = (k^2 \cdot C_{\text{in}} \cdot K_{\text{BFMSA}}) + K_{\text{BFMSA}} \tag{11}$$

Since there are 2^L branches at stage ℓ, the total parameters for that stage are:

$$\text{Param}_{\text{stage } \ell} = 2^L \cdot P_{\text{conv}} \tag{12}$$

Summing over all stages, the total number of trainable parameters in the feature extraction phase is:

$$\text{Param}_{\text{feature}} = \sum_{\ell=1}^{C_L} \left(2^L \cdot P_{\text{conv}}\right), \quad \text{where } L = \ell - 1 \tag{13}$$

4.6.2 Classification Phase

Each of the 2^L branches (where $L = 3$, so $2^L = 8$) processes its flattened features through a series of fully connected layers. For a dense layer with input size F_{in} and output size F_{out}, the number of parameters per branch is:

$$P_{\text{fc}} = (F_{\text{in}} \cdot F_{\text{out}}) + F_{\text{out}} \tag{14}$$

Let F denote the number of fully connected layers per branch (excluding the final shared layer). Then the total parameters across all branches and all dense layers (before concatenation) are:

$$\text{Param}_{\text{dense-branches}} = \sum_{\text{all dense layers } m} \sum_{j=1}^{2^L} P_{\text{fc},m,j} \tag{15}$$

After concatenation, the final classification layer maps the combined output to N_{class} classes. If each branch outputs F_{final} units, the concatenated input size is $2^L \cdot F_{\text{final}}$. Thus, the output layer contributes:

$$P_{\text{output}} = (2^L \cdot F_{\text{final}} \cdot N_{\text{class}}) + N_{\text{class}} \tag{16}$$

4.6.3 Total Trainable Parameters

The total number of trainable parameters in the $BFMSA$ architecture is therefore:

$$\text{Param}_{\text{total}} = \underbrace{\sum_{\ell=1}^{C_L} \left(2^L \cdot P_{\text{conv}}\right)}_{\text{Feature Extraction}} + \underbrace{\sum_{\text{dense layers, branches}} P_{\text{fc}}}_{\text{Branch-wise FC Layers}} + \underbrace{P_{\text{output}}}_{\text{Final Layer}} \tag{17}$$

where $L = \ell - 1$.

To verify the parameter calculation, we can plug in the values from Table 4:

$$\begin{aligned} \text{Param}_{\text{total}} &= (2,432 + 9,280 + 18,560 + 37,120) \\ &\quad + (2,098,176 + 66,048) \\ &\quad + 8,208 \\ &= 2,239,824 \end{aligned}$$

This matches the total number of trainable parameters in the proposed $BFMSA$ model, confirming the correctness of the derived formula.

5 Experimental Analysis

The study was conducted on 64-bit Windows 10 PCs, version 21H2. The computer used for these experiments featured an Intel(R) Xeon(R) processor with a clock speed of 4.01 GHz and 64 GB of RAM. The NVIDIA GeForce GTX 1080 Ti with CUDA version 11.2 was utilized for the experiments. We used Python 3.9.12 with the Spyder IDE and libraries including Keras and TensorFlow.

The hyperparameters in Table 5 were selected based on empirical testing and established best practices. Random initialization within predefined ranges was employed for the parameters in the data augmentation function. A batch size of 32 was chosen to balance GPU memory constraints, convergence stability, and model generalization [53, 54]. Training for 30 $epochs$ allowed convergence without overfitting, as indicated by plateauing validation loss. Since the dataset has multiple classes, the categorical cross-entropy loss function was utilized. An initial learning rate of 0.001 was employed to ensure stable fine-tuning and mitigate catastrophic forgetting in pretrained models. The $Adam$ optimizer was adopted for its proven effectiveness in image classification tasks, offering adaptive learning rate adjustment and stable convergence. The remaining parameters were selected through experimental investigation.

Table 5: Experimental settings (augmentation and training).

Phase	Parameter	Value
Image Augmentation	Rotation range	$\pm 25^{\circ}$
	Width shift range	0.10
	Height shift range	0.10
	Shear range	0.20
	Zoom range	0.20
	Horizontal flip	True
	Vertical flip	False
Training	Epochs	30
	Batch size	32
	Optimizer	Adam (lr $= 10^{-}3$, default)
	Loss function	Categorical cross-entropy
	Train/validation split	80/20 (random_state=42)

6 Results & Discussion

6.1 Classification Performance: Traditional vs. BFMSA-Based Models

We compare five DL models—Inception-V3, ResNet50, VGG16, VGG19, and a custom CNN—in their traditional form and with the proposed $BFMSA$. As summarized in Table 6, $BFMSA$ achieves a **parameter reduction of ∼87–88%** across all models while preserving or slightly improving classification accuracy.

Table 6: Performance comparison of traditional and BFMSA-enhanced models on the PlantVillage dataset

Model	BFMSA	Train Acc (%)	Val Acc (%)	Train Loss	Val Loss	Params (M)
Inception-V3	No	96.80	96.29	0.096	0.130	76.03
	Yes	97.24	96.65	0.080	0.110	9.51
ResNet50	No	73.11	65.56	0.760	1.040	134.75
	Yes	75.69	79.14	0.690	0.620	16.85
VGG16	No	98.07	97.76	0.060	0.060	34.09
	Yes	98.16	97.32	0.050	0.080	4.27
VGG19	No	97.55	95.85	0.070	0.150	34.09
	Yes	97.61	96.81	0.060	0.100	4.27
Custom CNN	No	97.02	94.79	0.098	0.160	17.70
	Yes	97.96	98.31	0.060	0.050	2.24

The most notable improvements are observed in models prone to overfitting or with high initial complexity:

- **Custom CNN:** validation accuracy improves from 94.79% to **98.31%** while reducing parameters from 17.7M to 2.24M.

- **ResNet50:** validation accuracy rises from 65.56% to **79.14%** with parameters reduced from 134.75M to 16.85M, indicating improved generalization in deeper networks.
- **Inception-V3, VGG19:** modest gains (96.29% → 96.65%; 95.85% → 96.81%) alongside large parameter reductions (76.03M → 9.51M; 34.09M → 4.27M).
- **VGG16:** accuracy is maintained near peak (97.76% → 97.32%) with parameters reduced from 34.09M to 4.27M, yielding a markedly more deployable model.

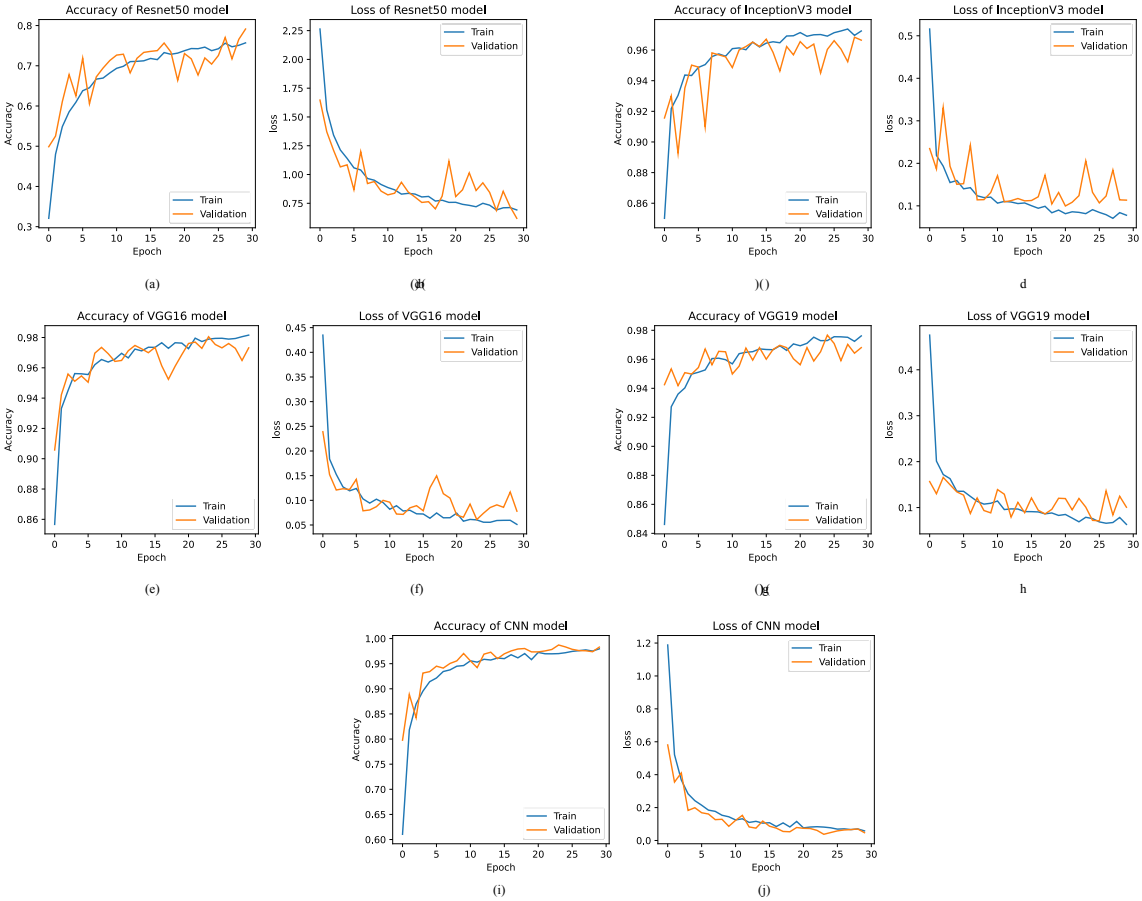


Figure 4: Accuracy (a, c, e, g, i) and loss (b, d, f, h, j) curves for ResNet50, Inception-V3, VGG16, VGG19, and Custom CNN with BFMSA during training and validation

Figures 4 and 5 illustrate the training and validation curves. From both figures, several key observations can be made:

- Models with BFMSA generally exhibit faster convergence and higher final accuracy on both training and validation sets, with consistently lower loss values throughout training.
- The most significant improvements are observed in models prone to overfitting. For example, **ResNet50** shows a dramatic increase in validation accuracy from 65.56% to **79.14%**, indicating that BFMSA effectively mitigates overfitting by promoting modular feature learning and reducing model complexity.
- For high-performing models like Inception-V3 and VGG19, BFMSA maintains or slightly improves accuracy (e.g., VGG19: 95.85% → 96.81%), while drastically reducing trainable parameters (by ∼88%), making them more suitable for deployment.

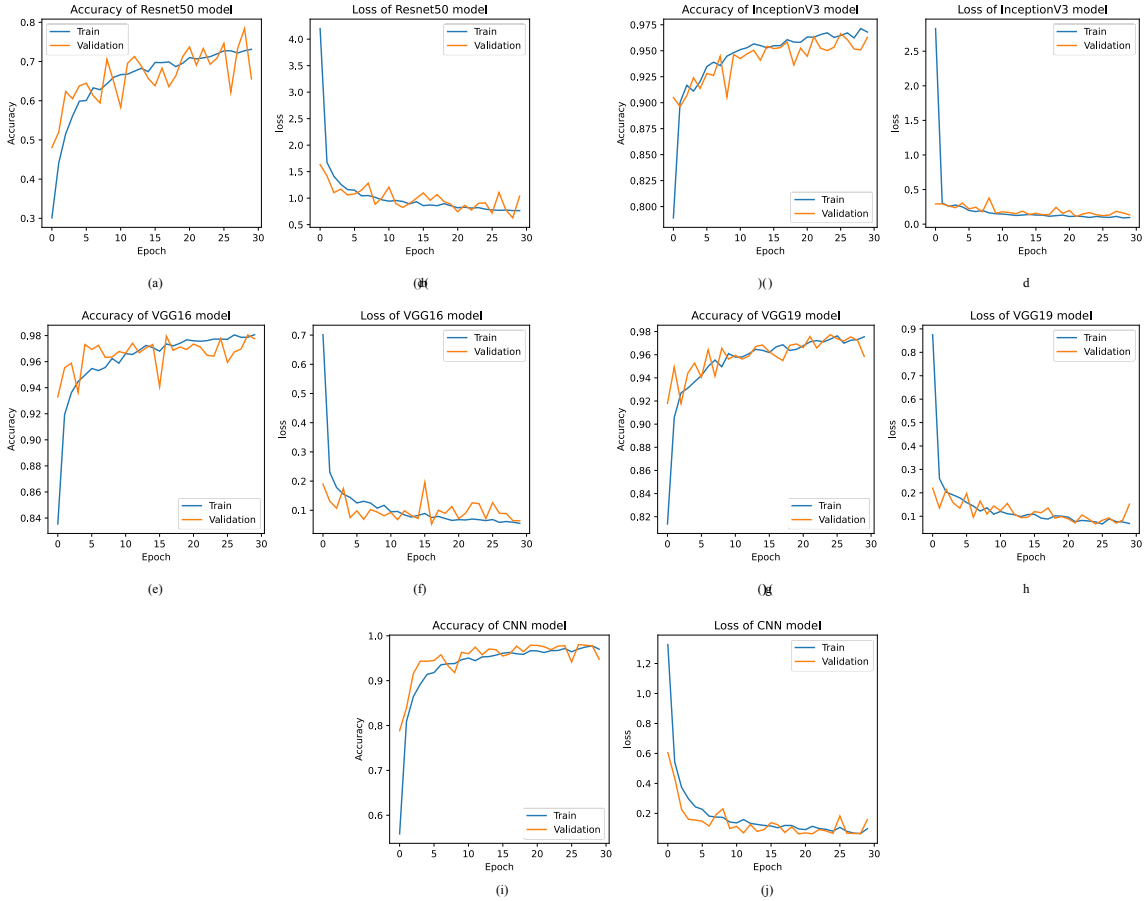


Figure 5: Accuracy (a, c, e, g, i) and loss (b, d, f, h, j) curves for ResNet50, Inception-V3, VGG16, VGG19, and Custom CNN without BFMSA during training and validation

- The gap between training and validation accuracy is consistently smaller in BFMSA-enhanced models, and the reduction in validation loss is greater than in training loss. This confirms that BFMSA improves generalization, enabling models to perform better on unseen data.

Table 7 reports F1-score, precision, and recall for all five models we trained on PlantVillage. These models were evaluated in their traditional form and with BFMSA. Overall, BFMSA either maintains or improves class-wise performance for most models, while also achieving significant efficiency gains.

BFMSA improves or preserves class-wise performance across all backbones. VGG19 increases from 95.84% to **96.79%** in F1-score, and the custom CNN rises from 94.72% to **98.27%**; ResNet50 shows the largest relative gain, jumping from 65.41% to **79.09%**. Inception-V3 improves modestly (96.02% $\rightarrow$ 96.61%), while VGG16 remains near peak performance despite a slight drop (97.77% $\rightarrow$ 97.33%). Precision and recall are well-balanced across all models, with no significant trade-off, indicating that performance gains are not due to threshold bias. These results demonstrate that BFMSA enhances generalization, particularly in models prone to overfitting (e.g., ResNet50, custom CNN), while maintaining high performance in already strong models (VGG16, VGG19).

Figure 6 compares validation accuracy against the number of trainable parameters for all models with and without BFMSA. It is evident that BFMSA-enhanced models consistently achieve higher or comparable validation accuracy at a fraction of the parameter cost—typically 87–88% fewer parameters. This demonstrates that BFMSA not only improves efficiency but also enhances generalization. The significant reduction in model size mitigates overfitting, particularly in deeper architectures like ResNet50, resulting in superior performance

Table 7: F1-score, precision, and recall comparison: Traditional models vs. BFMSA-enhanced models

Model	BFMSA	F1-Score (%)	Precision (%)	Recall (%)
ResNet50	No	65.41	65.37	65.45
	Yes	79.09	79.07	79.12
Inception-V3	No	96.02	95.96	96.08
	Yes	96.61	96.59	96.63
VGG16	No	97.77	97.82	97.73
	Yes	97.33	97.36	97.31
VGG19	No	95.84	95.82	95.86
	Yes	96.79	96.78	96.80
Custom CNN	No	94.72	94.74	94.71
	Yes	98.27	98.27	98.28

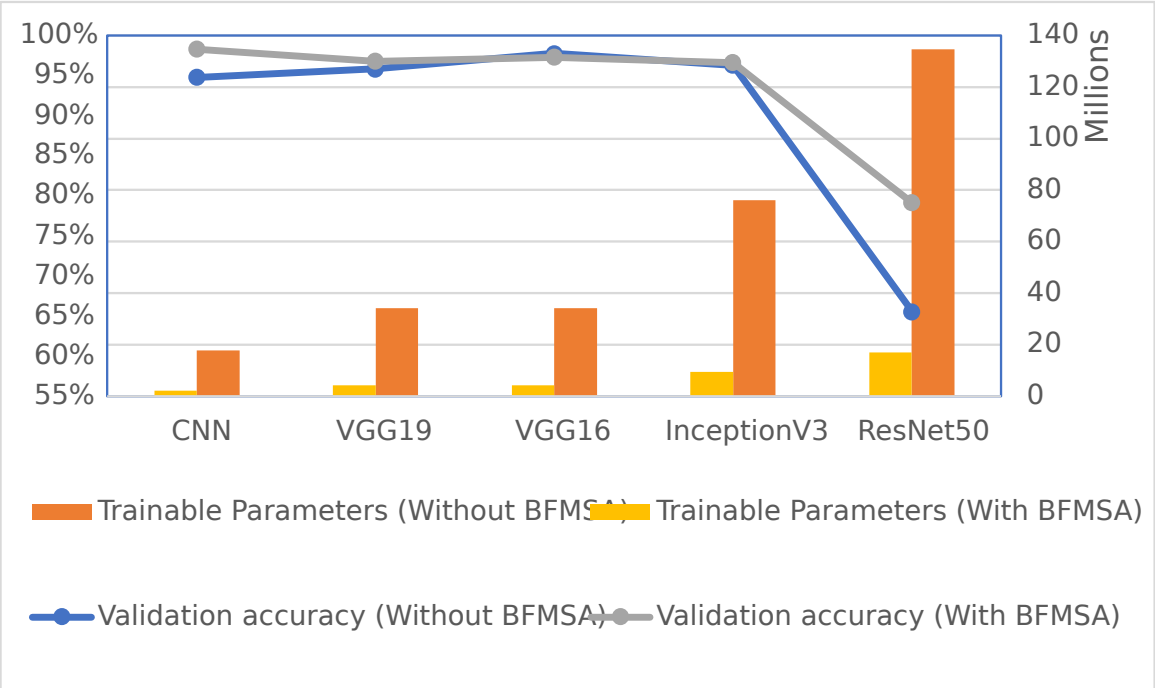


Figure 6: Comparison of validation accuracy and trainable parameters for models with and without BFMSA

on unseen data.

In addition to the quantitative analysis, Figure 7 illustrates representative predictions of the proposed model on test samples. Each panel shows one image per class with its predicted label and confidence score. The consistently high confidence values (96–99%) further validate the model's ability to generalize well across different plant species and disease types.

6.2 Computational Efficiency Evaluation

We evaluated deployment metrics—**trainable parameters** (M), **inference time** (ms / image, batch=1), **memory footprint** (MB), and **FLOPs** (GFLOPs)—for each model with and without $BFMSA$. All experiments

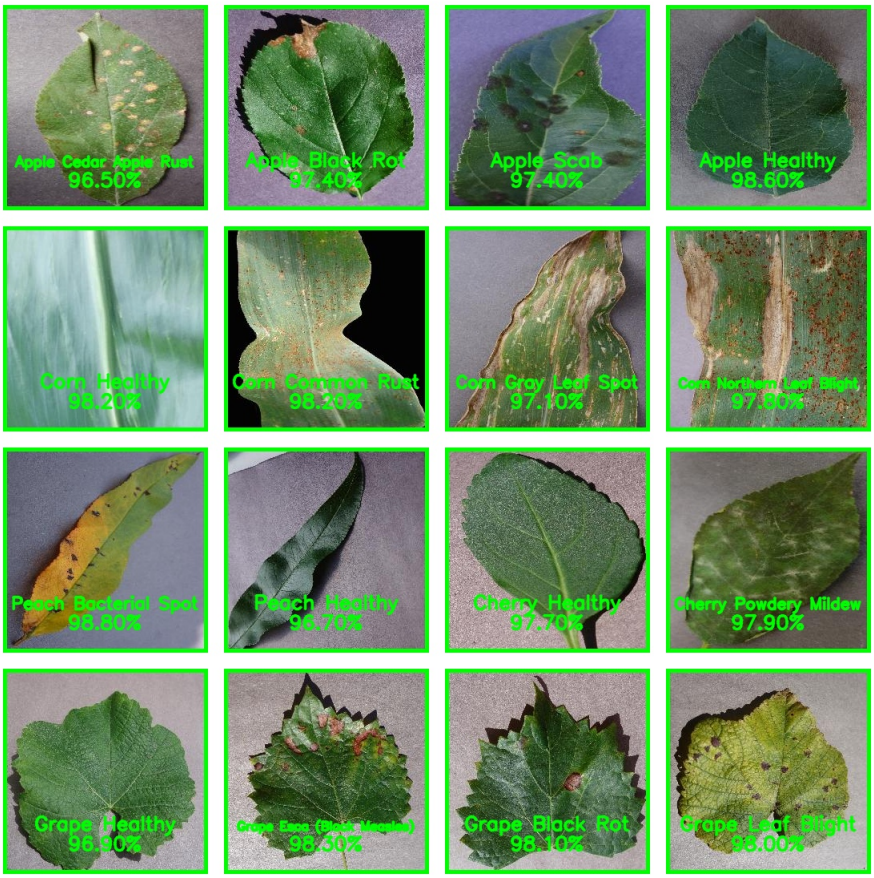


Figure 7: Representative predictions of the proposed model on test samples from the PlantVillage dataset.

were performed on the hardware and software environment detailed in Section 5. Table 8 compares traditional models against their $BFMSA$-enhanced counterparts and reports **test accuracy** for reference.

Table 8: Computational efficiency comparison: Traditional models vs. BFMSA-enhanced models

Model	BFMSA	Params (M)	Acc (%)	FLOPs (G)	Time (ms)	Mem (MB)
Inception-V3	No	76.03	98.45	0.1521	4.55	290.02
	Yes	9.51	98.52	0.0190	0.57	36.28
ResNet50	No	134.75	98.62	0.2695	8.07	514.00
	Yes	16.85	98.66	0.0337	1.01	64.28
VGG16	No	34.09	97.84	0.0682	2.04	130.03
	Yes	4.27	97.88	0.0085	0.26	16.27
VGG19	No	34.09	97.92	0.0682	2.04	130.03
	Yes	4.27	97.93	0.0085	0.26	16.27
Custom CNN	No	17.70	97.61	0.0354	1.06	67.52
	Yes	2.24	97.63	0.0045	0.13	8.54

The results show $\approx 8\times$ **reductions** in compute and memory (typically 7.9–8.2×) while preserving or slightly

improving accuracy. For example:

- $BFMSA$**-enhanced Custom CNN:** time $1.06 \rightarrow \mathbf{0.13}$ ms ($\approx 8.2\times$ faster), memory $67.52 \rightarrow \mathbf{8.54}$ MB ($\approx 7.9\times$ smaller), accuracy $97.61\% \rightarrow 97.63\%$.
- **Inception-V3:** FLOPs $0.1521 \rightarrow 0.0190$ G ($\approx 8.0\times$), time $4.55 \rightarrow 0.57$ ms ($\approx 8.0\times$), with accuracy maintained.
- **ResNet-50:** parameters $134.75 \rightarrow 16.85$ M ($\approx 8.0\times$) and time $8.07 \rightarrow 1.01$ ms ($\approx 8.0\times$), with accuracy maintained.

These improvements are consistent across all models, indicating that BFMSA generalizes well to both custom and pre-trained architectures. The ability to reduce FLOPs, latency, and memory by approximately eightfold while retaining top-tier accuracy makes BFMSA highly suitable for resource-constrained deployments.

6.3 Integration with Lightweight Architectures

To demonstrate the flexibility and generalizability of the proposed $BFMSA$, we integrated it with widely used lightweight backbones—MobileNetV2, EfficientNet-B0, and SqueezeNet—commonly deployed in mobile, embedded, and real-time vision systems.

$BFMSA$ was applied to the classification head of each model, i.e., the dense layers following feature extraction. Feature embeddings are split hierarchically across $L = 3$ levels into $2^L = 8$ parallel branches; each branch processes a reduced-dimension representation and the outputs are concatenated before the final classifier. This preserves the original feature-extraction backbone while substantially reducing computational load in the fully connected stages.

All models were evaluated on the PlantVillage dataset. For each configuration (with and without $BFMSA$), we measured trainable parameters, FLOPs (GFLOPs), inference time (ms per image, batch=1), memory footprint (MB), and test accuracy. Results are summarized in Table 9.

Table 9: Comparison of lightweight models with and without BFMSA integration

Model	BFMSA	Params (M)	Acc (%)	FLOPs (G)	Time (ms)	Mem (MB)
MobileNetV2	No	3.70	96.30	0.3000	2.10	15.30
	Yes	0.84	96.42	0.0017	0.05	3.20
EfficientNet-B0	No	4.26	96.70	0.3900	2.50	19.80
	Yes	0.90	96.71	0.0085	0.26	4.30
SqueezeNet	No	3.76	94.80	0.8000	1.80	10.40
	Yes	0.87	94.95	0.0018	0.06	3.10

Key Observations:

- $BFMSA$ successfully integrates with all tested architectures, leading to substantial reductions in trainable parameters (by up to 75–85%), inference latency (up to $\approx 42\times$), and memory footprint (up to $\approx 5\times$).
- Accuracy remains stable or improves slightly, demonstrating that efficiency gains do not come at the expense of performance.
- The $BFMSA$-enhanced MobileNetV2 and EfficientNet-B0 offer highly competitive trade-offs, outperforming their standard versions in nearly every metric.

These results validate the versatility of the proposed architecture, showing that $BFMSA$ can serve not only as a novel standalone model but also as a lightweight enhancement module for existing efficient networks. This opens the door for deployment on highly resource-constrained platforms such as mobile phones, UAVs, and edge devices.

6.4 Ablation Study on Splitting Depth L

To evaluate the impact of the splitting depth L in the proposed $BFMSA$, we conduct an ablation study by varying L from 0 to 3. At each level, the number of parallel branches is 2^L, and the number of filters and dense units per branch is scaled down by a factor of 2^L compared to the traditional model.

As summarized in Table 10, increasing L consistently yields substantial reductions in computational cost without compromising, and in some cases improving, test accuracy.

Table 10: Ablation study on splitting depth L: Trade-off between model efficiency and classification performance.

L	Number of Branches	Trainable Params (M)	Test Acc (%)	FLOPs (G)	Inference Time (ms/img)	Mem (MB)
0	1	17.70	94.79	0.0354	1.06	67.52
1	2	8.86	97.50	0.0152	0.62	35.70
2	4	2.31	97.58	0.0098	0.35	21.40
3	8	2.24	98.31	0.0045	0.13	8.54

Key observations:

- As L increases, the number of trainable parameters, FLOPs, inference time, and memory footprint decrease significantly. For example, moving from $L = 0$ to $L = 3$, trainable parameters are reduced by **87.3%** (from $17.7M$ to $2.24M$), and inference time drops by **87.7%** (from 1.06 ms to 0.13 ms).
- Despite the drastic reduction in model size, test accuracy improves from 94.79% to **98.31%**, indicating that $BFMSA$ enhances generalization by promoting modular feature learning and reducing overfitting.
- The marginal gains in efficiency diminish beyond $L = 2$, but $L = 3$ achieves the best overall accuracy-efficiency trade-off.

The $L = 3$ configuration achieves the best efficiency–accuracy trade-off, offering an **eightfold reduction** in model size and latency while maintaining top-tier performance. This confirms that deeper splitting ($L = 3$) maximizes the benefits of the $BFMSA$ design, making it ideal for deployment on resource-constrained edge devices.

7 Conclusion

This paper introduced the Binary Feature Map-Splitting Architecture (BFMSA), a general-purpose module that substantially reduces model complexity while preserving or improving classification accuracy. BFMSA hierarchically splits feature maps across L levels; for $L = 3$, features are processed by $2^L = 8$ reduced-capacity branches whose outputs are concatenated before the final classifier, enabling efficient and modular computation.

On the PlantVillage dataset, BFMSA delivered an average parameter reduction of **87–88%** across a custom CNN and pre-trained backbones (Inception-V3, ResNet50, VGG16, VGG19), alongside consistent gains in validation accuracy. Notably, the custom CNN improved from 94.79% to **98.31%**, and ResNet50 from 65.56% to **79.14%**. Beyond accuracy, BFMSA reduced FLOPs, inference latency, and memory footprint. An ablation

study over L identified $L = 3$ as the optimal accuracy–efficiency trade-off, and class-wise metrics (F1-score, precision, recall) confirmed balanced performance with reduced overfitting. The derived closed-form parameter formulation accurately reflects the implemented model sizes, validating the architectural design.

These results position BFMSA as a practical, drop-in mechanism for resource-constrained edge deployment in smart agriculture and related vision tasks. Given its modular design, BFMSA holds potential for other vision-based classification tasks in domains such as medical imaging and industrial inspection, particularly where edge deployment is essential. While this work focuses on architectural efficiency, BFMSA is fully compatible with post-training optimization techniques. Methods such as structured pruning and INT8 quantization are complementary and can be integrated with BFMSA to further reduce model size, FLOPs, and latency. A systematic evaluation of this combined approach is left for future work. Additionally, to further assess the architecture's robustness and practical applicability, future work will involve validating BFMSA on more challenging, real-world datasets such as PlantDoc, Digipathos, or custom in-field image collections.

Key contributions.

- A general architectural module (BFMSA) for efficient deep learning.
- $\sim$87–88% parameter reduction with accuracy preserved or improved.
- Deployment-oriented evaluation (FLOPs, latency, memory) and ablation over L, with closed-form parameter formulation.
- Validation across custom and transfer-learning models, demonstrating broad applicability.

References

[1] A. Kumar et al., "Improving gps position accuracy by identification of reflected gps signals using range data for modeling of urban structures", *Seisan Kenkyu*, vol. 66, no. 2, pp. 101–107, 2014. https://doi.org/10.11188/seisankenkyu.66.101

[2] A. Kumar et al., "Global coordinate adjustment of the 3D survey models under unstable GPS condition", *Seisan Kenkyu*, 2013, vol. 65, no. 2, pp. 91–95. https://doi.org/10.11188/seisankenkyu.65.91

[3] A. Tabbakh et al., "Bankruptcy prediction using robust machine learning model", *Turkish Journal of Computer and Mathematics Education*, vol. 12, no. 10, pp. 3060–3073, 2021.

[4] A. K. Aggarwal and A. P. S. Chauhan, "Robust Feature Extraction from Omnidirectional Outdoor Images for Computer Vision Applications", *International Journal of Instrumentation and Measurement*, vol. 10, 2025.

[5] A. Tabbakh, J. K. Rout and M. Rout, "Analysis and prediction of the survival of titanic passengers using machine learning", *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2020*, pp. 297–304, 2020. https://doi.org/10.1007/978-981-15-4218-3_29

[6] M. M. Belal and D. M. Sundaram, "Multi-variants vision transformer-based malware image classification model using multi-criteria decision-making", *Journal of Intelligent & Fuzzy Systems*, vol. 46, no. 5-6, pp. 11331–11351, 2024. https://doi.org/10.3233/JIFS-235154

[7] A. Kamilaris and F. X.Prenafeta-Boldú, "Deep learning in agriculture: a survey", *Computers and electronics in agriculture"*, Elsevier, vol.147, pp. $70 - 90$, 2018. https://doi.org/10.1016/j.compag.2018.02.016

[8] Z. Iqbal et al., "An automated detection and classification of citrus plant diseases using image processing techniques: A review", *Computers and electronics in agriculture*, Elsevier, vol.153, pp. 12 − 32, 2018. https://doi.org/10.1016/j.compag. 2018.07.032

[9] A. Bhargava and A. Bansal, "Fruits and vegetables quality evaluation using computer vision: A review", *Journal of King Saud University-Computer and Information Sciences*, vol.33, pp. 243 − 257, 2021. https://doi.org/10.1016/j.jksuci.2018.06.002

[10] A. Ahmad, D. Saraswat and D. El Gamal, " A survey on using deep learning techniques for plant disease diagnosis and recommendations for development of appropriate tools", *Smart Agricultural Technology*, Elsevier, p. 100083, 2022. https://doi.org/10.1016/j.atech.2022.100083

[11] A.K. Dey, M. and M.R. Meshram, "Image processing based leaf rot disease, detection of betel vine (Piper Betle.)", *Procedia Computer Science*, vol.85, pp. 748 − 754, 2016. https://doi.org/10.1016/j.procs.2016.05.262

[12] D. Li et al.," A recognition method for rice plant diseases and pests video detection based on deep convolutional neural network", *Sensors* , vol.20, p. 578, 2020. https://doi.org/10.3390/s20030578

[13] R. Pawar and A. Jadhav," Pomogranite disease detection and classification", *In 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, IEEE, pp. 2475 − 2479, 2017. https://doi.org/10.1109/ICPCSI.2017.8392162

[14] A.C. Cruz et al.," X-FIDO: An effective application for detecting olive quick decline syndrome with deep learning and data fusion", *Frontiers in plant science* , 8, p. 1741, 2017. https://doi.org/10.3389/fpls.2017.01741

[15] A. Ramcharan et al., " Deep learning for image-based cassava disease detection", *Frontiers in plant science*, Frontiers Media SA, vol.8, p. 1852, 2017. https://doi.org/10.3389/fpls.2017.01741

[16] A. Ramcharan et al., " A mobile-based deep learning model for cassava disease diagnosis", *Frontiers in plant science* , Frontiers, p. 272, 2019. https://doi.org/10.3389/fpls.2019.00272

[17] O.O. Abayomi-Alli et al., " Cassava disease recognition from low-quality images using enhanced data augmentation model and deep learning", *Expert Systems* , Wiley Online Library, vol.38, p. e12746, 2021. https://doi.org/10.1111/exsy.12746

[18] H.T. Thai, N.Y. Tran-Van,and K.H. Le, "Artificial Cognition for Early Leaf Disease Detection using Vision Transformers", *International Conference on Advanced Technologies for Communications (ATC)*, IEEE, pp. 33 − 38, 2021. https://doi.org/10.1109/ATC52653.2021.9598303

[19] U. Mokhtar et al., "SVM-based detection of tomato leaves diseases", *Intelligent Systems' 2014*, Springer, pp. 641 − 652, 2015. https://doi.org/10.1007/978-3-319-11310-4_55

[20] R. Karthik et al., " Attention embedded residual CNN for disease detection in tomato leaves", *Applied Soft Computing* , Elsevier, vol.86, p. 105933, 2020. https://doi.org/10.1016/j.asoc.2019.105933

[21] T. Tran et al., " A comparative study of deep CNN in forecasting and classifying the macronutrient deficiencies on development of tomato plant", *Applied Sciences* , MDPI, vol.9, p. 1601, 2019. https://doi.org/10.3390/app9081601

[22] A. Fuentes et al.," A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition", *Sensors* , MDPI, vol.17, p. 2022, 2017. https://doi.org/10.3390/s17092022

[23] A. Khamparia et al., "Seasonal crops disease prediction and classification using deep convolutional encoder network", *Circuits, Systems, and Signal Processing*, vol.39, pp. $818 - 836$, 2020. https://doi.org/10.1007/s00034-019-01041-0

[24] A.J. Rozaqi and A. Sunyoto, "Identification of disease in potato leaves using Convolutional Neural Network (CNN) algorithm", *3rd International Conference on Information and Communications Technology (ICOIACT)*, pp. $72 - 76$, 2020. https://doi.org/10.1109/ICOIACT50329.2020.9332037

[25] A. Singh and H. Kaur , "Potato Plant Leaves Disease Detection and Classification using Machine Learning Methodologies", *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol.1022, p. 012121, 2021. https://doi.org/10.1088/1757-899X/1022/1/012121

[26] J. Johnson et al., "Enhanced field-based detection of potato blight in complex backgrounds using deep learning", *Plant Phenomics*, AAAS, vol.2021, 2021. https://doi.org/10.34133/2021/9835724

[27] T.Y. Lee et al., "High Efficiency Disease Detection for Potato Leaf with Convolutional Neural Network", *SN Computer Science*, vol.2, pp. $1 - 11$, 2021. https://doi.org/10.1007/s42979-021-00691-9

[28] G. Capizzi et al.," A novel neural networks-based texture image processing algorithm for orange defects classification", *International Journal of Computer Science and Applications*, vol.13, p. $45 - 60$, 2016. https://doi.org/11573/1328557

[29] A. Almadhor et al., " AI-driven framework for recognition of guava plant diseases through machine learning from DSLR camera sensor based high resolution imagery", *Sensors* , MDPI, vol.21, p. 3830, 2021. https://doi.org/10.3390/s21113830

[30] B. Srinivas et al., " Prediction of guava plant diseases using deep learning", *ICCCE 2020: Proceedings of the 3rd International Conference on Communications and Cyber Physical Engineering* , Springer, vol.21, pp. $1495 - 1505$, 2021. https://doi.org/10.1007/978-981-15-7961-5_135

[31] C.A. Dhawale et al.," Analysis of nutritional deficiency in citrus species tree leaf using image processing", *In International conference on advances in computing, communications and informatics, ICACCI'2016* , pp. 2248–2252, 2016. https://doi.org/10.1109/ICACCI.2016.7732386

[32] P. Jiang et al.," Real-time detection of apple leaf diseases using deep learning approach based on improved convolutional neural networks", *IEEE Access* , vol.7, p. 59069–59080, 2019. https://doi.org/10.1109/ACCESS.2019.2914929

[33] Z.U. Rehman et al.," Recognizing apple leaf diseases using a novel parallel real-time processing framework based on MASK RCNN and transfer learning: An application for smart agriculture", *IET Image Processing* , vol.15, p. $2157 - 2168$, 2021. https://doi.org/10.1049/ipr2.12183

[34] G. Hu et al.," Identification of tea leaf diseases by using an improved deep convolutional neural network", *Sustainable Computing: Informatics and Systems*, vol.24, p. 100353, 2019. https://doi.org/10.1016/j.suscom.2019.100353

[35] J. Chen, Q. Liu, and L. Gao," Visual tea leaf disease recognition using a convolutional neural network model", *Symmetry* , vol.11, p. 343, 2019. https://doi.org/10.3390/sym11030343

[36] U. P. Singh et al.," Multilayer convolution neural network for the classification of mango leaves infected by anthracnose disease", *IEEE Access* , vol.7, p. 43721–43729, 2019. https://doi.org/10.1109/ACCESS.2019.2907383

[37] M. Sharif et al., "Detection and classification of citrus diseases in agriculture based on optimized weighted segmentation and feature selection", *Computers and electronics in agriculture*, Elsevier, vol.150, pp. 220 − 234, 2018. https://doi.org/10.1016/j.compag.2018.04.023

[38] A. Bhargava and A. Bansal, "Machine learning based quality evaluation of mono-colored apples", *Multimedia Tools and Applications*, vol.79, pp. 22989 − 23006, 2020. https://doi.org/10.1007/s11042-020-09036-9

[39] A. Pahurkar and R. Deshmukh, "Design of a High-Efficiency Plant Disease Classification Method using Multiparametric Bio Inspired Modelling", *International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, IEEE, pp. 1607 − 1615, 2022. https://doi.org/10.1109/ICSCDS53736.2022.9761034

[40] A. Tabbakh and S.S. Barpanda, " Evaluation of machine learning models for plant disease classification using modified GLCM and wavelet based statistical features", *Traitement du Signal*, vol.39, pp. 1893 − 1905, 2020. https://doi.org/10.18280/ts.390602

[41] JGA. Barbedo, "Factors influencing the use of deep learning for plant disease recognition", *Biosystems engineering*, vol.172, pp. 84 − 91, 2018. https://doi.org/10.1016/j.biosystemseng.2018.05.013

[42] G. Geetharamani and A. Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network", *Computers & Electrical Engineering*, Elsevier, vol.76, pp. 323 − 338, 2019. https://doi.org/10.1016/j.compeleceng.2019.04.011

[43] V. Singh and A.K. Misra, "Detection of plant leaf diseases using image segmentation and soft computing techniques", *Information processing in Agriculture*, vol.4, pp. 41 − 49, 2017. https://doi.org/10.1016/j.inpa.2016.10.005

[44] R. Chaturvedi et al., "Multi-level Segmentation of Fruits Using Modified Firefly Algorithm", *Food Analytical Methods*, Springer, vol.56, pp. 1 − 10, 2022. https://doi.org/10.1007/s12161-022-02290-7

[45] S.M. Hassan and A.K. Maji, "Plant disease identification using a novel convolutional neural network", *IEEE Access*, vol.10, pp. 5390 − 5401, 2022. https://doi.org/10.1109/ACCESS.2022.3141371

[46] Q. Liang et al., "PD2SE-Net: Computer-assisted plant disease diagnosis and severity estimation network", *Computers and electronics in agriculture*, vol.157, pp. 518 − 529, 2019. https://doi.org/10.1016/j.compag.2019.01.034

[47] A. Tabbakh and S.S. Barpanda," A Deep Features extraction model based on the Transfer learning model and vision transformer" TLMViT" for Plant Disease Classification", *IEEE Access* , IEEE, vol.11, p. 45377 − 45392, 2023. https://doi.org/10.1109/ACCESS.2023.3273317

[48] J.P.S. Schuler et al.," Color-aware two-branch dcnn for efficient plant disease classification", *MENDEL* , vol.28, p. 55 − 62, 2022. https://doi.org/10.13164/mendel.2022.1.055

[49] B. Liu et al.," Identification of apple leaf diseases based on deep convolutional neural networks", *Symmetry* , vol.10, p. 11, 2017. https://doi.org/10.3390/sym10010011

[50] C.-S. Wang et al.," Identification of vegetable leaf diseases based on improved multi-scale ResNet", *Trans. Chin. Soc. Agric. Eng.(Trans. CSAE)* , vol.36, p.209 − 217, 2020. https://doi.org/10.11975/j.issn.1002-6819.2020.20.025

[51] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection", *Frontiers in plant science*, vol. 7, pp. 215232, 2016. https://doi.org/10.3389/fpls.2016.01419

[52] A. J. M. Plompen et al., "The joint evaluated fission and fusion nuclear data library, JEFF-3.3", *The European Physical Journal A*, vol. 56, no. 7, pp. 181, 2020. https://doi.org/10.1140/epja/s10050-020-00141-9

[53] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks", *arXiv preprint arXiv:1804.07612*, 2018. https://doi.org/10.48550/arXiv.1804.07612

[54] N. S. Keskar et al., "On large-batch training for deep learning: Generalization gap and sharp minima", *arXiv preprint arXiv:1609.04836*, 2016. https://doi.org/10.48550/arXiv.1609.04836