Utilizing Adam Optimizer with Hybrid Techniques for Detection and Classification of Tomato Leaf Diseases

Lakshmi Jayanth Reddy Nallamilli
Dept of Computer Science and Engg
Amrita School of Computing
Amrita Vishwa Vidyapeetham
Chennai, India
jay23238896@gmail.com

Rahul Govadi
Dept of Computer Science and Engg
Amrita School of Computing
Amrita Vishwa Vidyapeetham)
Chennai, India
govadirahul@gmail.com

Aman Reddy Jukonti
Dept of Computer Science and Engg
Amrita School of Computing
Amrita Vishwa Vidyapeetham
Chennai, India
amanreddyjukonti@gmail.com

Upendra Rejeti
Dept of Computer Science and Engg
Amrita School of Computing
Amrita Vishwa Vidyapeetham
Chennai, India
upendrarejeti06@gmail.com

Koushik Voota
Dept of Computer Science and Engg
Amrita School of Computing
Amrita Vishwa Vidyapeetham
Chennai, India
vkoushik321@gmail.com

Dr.I.R. Oviya
Dept of Computer Science and Engg
Amrita School of Computing
Amrita Vishwa Vidyapeetham
Chennai, India
iroviya@gmail.com

***Abstract*—Tomato leaf disease affects high agricultural yields, and they need accurate and efficient detection methods. In this present study, we propose a deep learning approach based on the Adam optimizer and pre-trained VGG16 and NASNet models to attain effective tomato leaf disease classification. The PlantVillage dataset is utilized for model training, and TensorFlow and Keras environments are used. With hybrid approaches, feature extraction and classification accuracy are optimized. The Adam optimizer enhances convergence effectiveness, resulting in robust training and improved precision. Our results suggest that the use of VGG16 and NASNet with hybrid approaches optimizes disease classification to the benefit of precision agriculture through timely detection and intervention to avert losses in crops.**

***Index Terms*—Adaptive Ensemble Learning, Tomato Leaf Disease Detection, Adam Optimizer, VGG-16 Fine-Tuning, NASNet Mobile Architecture, Image Preprocessing**

I. Introduction

Tomatoes are one of the most valuable crops in the global economy. The crop is staple in many diets and is very central in trade in agricultural products. Despite the fact that tomato plants are so delicate, especially with the diseases attacking the leaves, great losses can occur in yield and quality. Plant disease identification has historically been a laborious and challenging process that is frequently prone to errors in such extensive operations. These drawbacks highlight the need for automated, accurate, and scalable methods to detect illnesses early and enable timely treatment. Recent advances in machine learning, particularly deep learning, have opened up new avenues for addressing these problems. Convolutional neural networks, or CNNs, are highly helpful for image classification applications because of their ability to automatically learn and extract complex information from images.

Building on this development, our study merely uses the PlantVillage dataset to reproduce and enhance the model for tomato leaf disease prediction. Making use of superior architectures, like VGG16 and NASNet, in conjunction with a higher dataset size, this paper presents an enhanced prediction model of detecting diseases more accurately and robustly.The results of this work have the potential to drastically alter the conventional agricultural system and open the door to a more sophisticated, data-driven precision agriculture strategy.

II. Literatrure Survey

The early and accurate identification of plant diseases, especially those that affect tomato crops, is paramount to enhancing agricultural output, reducing the use of pesticides, and ensuring food security. Many of the deep learning and machine learning methods have been utilized through time for enabling automation in the classification of tomato leaf diseases.This literature review represents the most crucial developments in this area, covering new approaches towards feature extraction, data augmentation, model optimization, and the applicability of a disease detection system. Among all the pioneering works in detecting tomato leaf disease, one prominent study is that titled "Classification of Tomato Leaf Images for Detection of Plant Disease Using Conformable Polynomials Image Features.". With feature extraction using conformable polynomials into the scheme presented, analysis improves the study significantly upon the analysis on tomato leaf image texture. Thus, it is the fruit of their labors in essentially combining their designed extraction and machine-learning classifying approaches that, in this study, use support-vector machines to achieve an accuracy of 98.80%. The study highlights the importance of accurate feature extraction in improving disease detection; it also highlights how these techniques have the potential to prevent financial losses by utilizing fewer unnecessary pesticides. The approach, which focuses on texture-based information, offers a practical means of rapidly and precisely classifying disorders. This research article, "Intelligent Agricultural Robotic Detection System for Greenhouse Tomato Leaf Diseases Using Soft Computing Techniques and Deep Learning," deals with the combination of deep learning with soft computing techniques for the autonomous monitoring of a greenhouse. To improve the robustness of the model, a DCGAN is suggested for augmenting the dataset with artificial images of illnesses. The study compares the performance of several deep learning models using the PlantVillage dataset, including VGG19, Inception-v3, DenseNet-201, and ResNet-152.

Training ResNet-152 on autocorrelations showed a spectacular 99.69% accuracy rate, which underlines the need for data augmentation in this research for enhancing model performance, coupled with introduce-ing a soft computation of deep learning for developing intelligent agriculture systems. With augmentation directly augmenting the dataset's size by four times, a tomato leaf disease detection model was created that uses the PyTorch and ResNet-50 frameworks and portrayed an accuracy of 97%. Not only does it improve the disease detection accuracy, but it will also help in developing autonomous systems for greenhouse environments that can carry out real-time monitoring of the plants. Furthermore, incorporating the proposed CCNN model into the mobile-based system will allow smartphone-assisted crop disease diagnostics to go global.The work titled "Improved Tomato Leaf Disease Classification Through Adaptive Ensemble Models with Exponential Moving Average Fusion and Enhanced Weighted Gradient Optimization" describes an ensemble model in which advanced methods for optimization fuse VGG-16 and NASNet mobile. This is a self-tuning of the learning rate with the EMA function in the model, while the EWGO optimizes the model's accuracy in learning. Therefore, the method applied improves classification accuracy by about 98.7% on the PlantVillage data set. This study shows the power of applying ensemble learning in combination with optimization techniques to overrule the difficulties of classification of complex disease patterns on tomato leaves. This approach does not only improve the performance of the classifier but it will also lead to stable and adaptive learning, making this approach very effective for real-world applications. As the title, "Identification of Tomato Leaf Diseases Using an Explanation-Driven Deep-Learning Model," indicates, the authors use emphasis on interpretability of deep learning models in their study. EfficientNetB5 coupled with the interpretability algorithms LIME and GradCAM yielded an astonishing test accuracy of 99.07 percent by the authors. These approaches offer explanations in a visual form that support users in understanding why the model has predicted the outcome, including which features of the leaf images explain the disease classification. This is an important aspect of trust building and critical to utilizing AI-based systems, as in agricultural environments where a farmer may be hostile to using "black-box" models. The paper shows the importance of interpretability within the machine learning community, especially with the introduction of such models into practically used disease-detection-and-management applications. A Smartphone-Based Detection System for Tomato Leaf Disease Using EfficientNetV2B2 and Its Explainability with Artificial Intelligence (AI)" moves a step ahead by identifying the usefulness of deploying deep learning models in mobile applications. Here the model achieves almost perfect accuracy with the help of EfficientNetV2B2 in tomato leaf diseases. The system is conceived so that it can be accessed using smartphones, and farmers can diagnose their diseases with the help of high-resolution images of tomato leaves within seconds. Explainability features are yet another dimension of the study that will lend insight into how these models function with respect to decision-making. It mandates that farmers be availed of the newest and the best AI innovations so they may promptly identify diseases and make well-informed decisions regarding them. With accuracies over 99% in the training process, this system promises to revolutionize disease detection on mobile applications for agriculture. All in all, this work shows major developments in research regarding the diagnosis of tomato leaf disease. From extracting features to a more advanced and complex deep model, the ideas and their pragmatic application in developing mobile applications were emphasized as one method of improving and making the diseases-detector system more effective, efficient, and accessible for all. Future agricultural practices will thereby enhance crop management, lower notable crop losses, and support the sustainability of agriculture globally with advancements along these development lines.

III. Methodology

The dataset used in this study was from the public PlantVillage Dataset, a plant leaf image collection that covers an extensive range of plant species and corresponding diseases. For tomato leaf disease classification, images pertaining to 10 unique classes were derived, both healthy and unhealthy tomato leaves. The selected classes are as follows:
Tomato___healthy
Tomato__mosaic_virus
Tomato___Tomato_Yellow_Leaf_Curl_Virus
Tomato___Target_Spot
Tomato___Spider_mites Two-spotted_spider_mite
Tomato___Septoria_leaf_spot
Tomato___Leaf_Mold
Tomato___Late_blight
Tomato___Early_blight
Tomato___Bacterial_spot
Images in these classes were sorted separately into subdirectories using each class as a subdirectory under a common parent directory. This helped maintain compatibility with

TensorFlow's ImageDataGenerator for preprocessing and augmentation purposes.

Dataset Statistics The dataset was checked to satisfy the following criteria:

Each class is satisfactorily covered. Duplicated or corrupted images are removed. Image dimensions and formats are correct.

Justification for the Dataset Selection The PlantVillage dataset, which covers a wide range of plant species and their corresponding diseases, was chosen for its comprehensiveness. Quality: Recorded high-resolution photos in a controlled environment. Diversity: Under various environmental circumstances, leaf samples that are both healthy and ill are included.

A. Augmentation and Preprocessing

1) Image Preprocessing: Rescaling:

By dividing each pixel value by 255, all images were normalized to a pixel value range of [0, 1]. This is to guarantee that the gradient-based optimization process uses consistent input data. Resizing: To match the input dimensions of the pre-trained VGG16 and NASNetMobile architectures, images were shrunk to 224 by 224 pixels. Data Splitting: Training (80%) and validation (20%) subsets of the dataset were created. The division was made at random while making certain that the classes in both subsets are distributed fairly.

2) Data Augmentation: To enhance the dataset's diversity and lessen overfitting, a number of augmentation approaches were used. Only the training subset was subjected to these changes, guaranteeing that the validation data was left unaltered for objective assessment. Among the enhancement parameters were: Rotation: Up to ±20 degrees of random rotation. Width Shift: Translation horizontally by up to 20Height Shift: Up to 20Shear Transformation: Up to 20Zooming: Up to 20Images are randomly flipped along the horizontal axis in a process known as horizontal flipping. TensorFlow's ImageDataGenerator was used to implement these augmentations, producing batches of dynamically altered photos.while being trained. This method maintained the initial dataset size on disk while guaranteeing an almost limitless supply of distinct training examples.

3) Batch Size and Data Flow: After experimenting with several values and weighing training speed and memory limitations, a batch size of 32 was chosen. The flow from directory method, which automatically translated the directory structure to the class labels, was used to load the preprocessed and enhanced photos into memory in batches.

4) TensorFlow Implementation: The TensorFlow library is used to implement the suggested model. One advantage of transfer learning is applied by the pre-trained VGG16 and NASNetMobile models. The following describes the pseudocode used to implement the same:

```
BEGIN

    // Initialize image data generator with
        preprocessing options
    CREATE datagen AS ImageDataGenerator
        SET rescale TO 1.0 / 255  // Normalize pixel
             values
        SET rotation_range TO 20  // Allow random
            rotation
        SET width_shift_range TO 0.2  // Allow
            horizontal shift
        SET height_shift_range TO 0.2  // Allow
            vertical shift
        SET shear_range TO 0.2  // Allow shear
            transformation
        SET zoom_range TO 0.2  // Allow zoom
            augmentation
        SET horizontal_flip TO TRUE  // Allow
            horizontal flipping
        SET validation_split TO 0.2  // Set
            validation split

    // Create training data generator
    CREATE train_generator AS datagen.
        flow_from_directory
        INPUT filtered_dataset_path
        SET target_size TO (224, 224)  // Resize
            images to 224x224
        SET batch_size TO 32  // Set batch size
        SET class_mode TO 'categorical'  // Set
            class mode for multi-class
            classification
        SET subset TO 'training'  // Specify this is
             for training data

    // Create validation data generator
    CREATE validation_generator AS datagen.
        flow_from_directory
        INPUT filtered_dataset_path
        SET target_size TO (224, 224)  // Resize
            images to 224x224
        SET batch_size TO 32  // Set batch size
        SET class_mode TO 'categorical'  // Set
            class mode for multi-class
            classification
        SET subset TO 'validation'  // Specify this
            is for validation data

END
```

Listing 1: Pseudocode for Image Data Generator and Augmentation

Load Pre-trained Models

Initialize the VGG16 model that is pre-trained on ImageNet with its weights. Exclude the top classification layer so that it is adapted to the customized dataset. Freeze all layers except the last 4 to ensure fine-tuning. Apply global average pooling to the output of VGG16.

Initialize the NASNetMobile model pre-trained on ImageNet with weights.

Remove the top classification layer to fit the custom dataset.

Freeze all layers except the last 4 for fine-tuning.

Apply global average pooling to the output of NASNetMobile.

Combine Model Outputs

Concatenate the pooled outputs from VGG16 and NASNetMobile. Add Fully Connected Layers

Add a dense layer with 128 neurons and ReLU activation. Add another dense layer with 64 neurons and ReLU activation. Add final dense layer with 10 neurons, softmax activation function for classification Make and Compile Model

Specify the model as inputs from VGG16 and NASNet-Mobile. Output as the final dense layer. Use adam optimizer and categorical cross entropy loss while setting accuracy for evaluation of model.

Impact of Augmentation The augmentation pipeline injects shifts, rotations, and flips into the dataset, therefore simulating real-world phenomena where leaves will be viewed in different angles with varying conditions. This pipeline reduced the occurrences of overfitting and consequently improved the generation of unseen data.

B. Model Architecture

The proposed model uses an ensemble approach, merging two powerful architectures of pre-trained CNNs: VGG16 and NASNetMobile. These architectures have been fine-tuned and integrated to exploit the complementary feature extraction capabilities.

1) Pre-trained Architectures: VGG16: VGG16 is one of the deep CNNs in use, boasting 16 layers, with great simplicity and outstanding performance in classification tasks. For this study, The weights used for initialization of the model are from ImageNet pretrained. Final classification layers are removed by including include_top=False Last four convolutional layers are unfrozen to facilitate fine-tuning the rest of the layers are set frozen in order to retain general-purpose features learned during pre-training. GAP is a layer added onto compressing the feature maps into a single vector representation. NASNetMobile: NASNetMobile is a lightweight neural architecture search (NAS)-optimized model for mobile and resource-constrained devices. The model is highly accurate but quite compact. For this paper: The model was initialized with the weights of the ImageNet pre-trained model. The classification head was removed and include_top=False. The last four layers are unfrozen just like in the case of VGG16 to fine-tune. A GAP layer was added for feature extraction.

2) Feature Fusion: The outputs of the two networks (VGG16 and NASNetMobile) were concatenated to a single unified feature vector. This fusion will capture a more diverse set of features, harnessing the benefits of both architectures. The concatenated vector was further passed through fully connected layers to classify.

Fusion and Fully Connected Layers Concatenate Layer: Combines the output feature maps of VGG16 and NASNet-Mobile. Dense Layers: The next dense layer of size 128 units with ReLU activation is applied for feature reduction and for learning of more complex patterns. Another dense layer of size 64 units, again using the ReLU activation function to sharpen the features, Output Layer: Finally, it used 10 neurons representing the tomato disease classes in the classification with softmax activation outputting the probabilities of classes.

3) Model Diagram: The overall architecture of the proposed model is described in Figure 1 (draw the architecture in your paper). It points out both the VGG16 and NASNet-Mobile, their own feature extraction, and finally fusing for the classification task.

4) TensorFlow Implementation: The model used the TensorFlow library. The below pseudo-code spells out the required steps:

```
BEGIN

    // Import necessary libraries for model building
    IMPORT VGG16, NASNetMobile FROM tensorflow.keras.applications
    IMPORT Dense, GlobalAveragePooling2D, concatenate FROM tensorflow.keras.layers
    IMPORT Model FROM tensorflow.keras.models

    // Load VGG16 model without the top layer
    CREATE vgg16_base AS VGG16
        SET weights TO 'imagenet'
        SET include_top TO FALSE
        SET input_shape TO (224, 224, 3)

    // Freeze all layers of VGG16 except the last 4 layers
    FOR EACH layer IN vgg16_base.layers EXCEPT LAST 4
        SET layer.trainable TO FALSE

    // Apply global average pooling to VGG16 output
    CREATE vgg16_output AS GlobalAveragePooling2D()(vgg16_base.output)

    // Load NASNetMobile model without the top layer
    CREATE nasnet_base AS NASNetMobile
        SET weights TO 'imagenet'
        SET include_top TO FALSE
        SET input_shape TO (224, 224, 3)

    // Freeze all layers of NASNetMobile except the last 4 layers
    FOR EACH layer IN nasnet_base.layers EXCEPT LAST 4
        SET layer.trainable TO FALSE

    // Apply global average pooling to NASNetMobile output
    CREATE nasnet_output AS GlobalAveragePooling2D()(nasnet_base.output)

    // Concatenate outputs from both models
    CREATE combined AS concatenate([vgg16_output, nasnet_output])

    // Add fully connected layers
    CREATE x AS Dense(128, activation='relu')(combined)
    CREATE x AS Dense(64, activation='relu')(x)
    CREATE output AS Dense(10, activation='softmax')(x)

    // Create the final model
    CREATE model AS Model
        SET inputs TO [vgg16_base.input, nasnet_base.input]
        SET outputs TO output

    // Compile the model with optimizer and loss function
    CALL model.compile
        SET optimizer TO 'adam'
        SET loss TO 'categorical_crossentropy'
        SET metrics TO ['accuracy']

END
```

Listing 2: Pseudocode for Hybrid Model Building with VGG16 and NASNetMobile

Pre-trained VGG16 Model Loads

Begin with a pre-trained VGG16 image classification model on ImageNet; remove the classification layer at the top to fit an application-specific dataset; freeze the entire model except for the last four layers to allow for fine-tuning on a particular downstream task; use global average pooling on output; Concatenate Outputs: Concatenate the outputs of the VGG16 model with the outputs of the NASNetMobile model using a concatenation operation; add a dense layer of 128 neurons and the ReLU activation function; add a dense layer of with 64 neurons and ReLU activation function. Add a final dense layer with 10 neurons and softmax activation for multi-class classification. Model Creation and Compilation

Define a model with input from the VGG16 and NASNetMobile models. Specify the output of the model as the final dense layer. Compile the model with the Adam optimizer, categorical cross-entropy loss function, and accuracy metric.

5) Rationale for the Architecture: The combination of VGG16 and NASNetMobile ensures that: Feature Diversity: VGG16 captures more generalizable, relatively simpler patterns. NASNetMobile extracts complex features tailored for efficiency in resources.

Fine-Tuning Capability: Unfreezing the last four layers of both architectures adjusts the model to domain-specific patterns of tomato leaf diseases.

Accurate Classification: Through the fully connected layers, this model aggregates features and refines them to detect diseases accurately.

C. Training Strategy

The training process was optimized to achieve high accuracy and generalization while minimizing overfitting. This was achieved by using advanced optimizers, a robust loss function, and strategic callbacks to dynamically adjust the learning process based on performance metrics.

1) Optimizer and Loss Function: Optimizer:

The Adam optimizer was used due to its efficiency in handling sparse gradients and its ability to adapt the learning rate dynamically. The initial learning rate was set to 0.0001, which is a balance between convergence speed and stability. Loss Function: The categorical cross-entropy loss function was used to train the model. This is the standard loss function for multi-class classification problems. It measures the divergence between the predicted probabilities and the true class labels.

2) Batch Size and Epochs: Batch Size: After experimenting, a batch size of 32 was chosen, providing an optimal trade-off between computational efficiency and gradient stability. Epochs: The training is done up to 25 epochs, ensuring enough time for the network to reach convergence.

3) Data Generators: To manage the dual-input architecture of the ensemble model, the Dataset API in TensorFlow was employed. Custom data generators were implemented to feed the training and validation datasets into the model. The steps are outlined below:

```
BEGIN

    // Define a custom generator function for dual
        input
    FUNCTION generate_double_input(generator)
        // Define an inner generator function
        FUNCTION gen()
            // Iterate over the images and labels
                from the provided generator
            FOR EACH (images, labels) IN generator
                // Yield a tuple of (images, images)
                     as dual input and labels
                YIELD ((images, images), labels)

        // Create a TensorFlow dataset from the
            inner generator
        RETURN tf.data.Dataset.from_generator
            SET gen TO gen
            SET output_signature TO (
                (
                    tf.TensorSpec(shape=(None, 224,
                        224, 3), dtype=tf.float32),
                         // input1
                    tf.TensorSpec(shape=(None, 224,
                        224, 3), dtype=tf.float32)
                          // input2
                ),
                tf.TensorSpec(shape=(None, 10),
                    dtype=tf.float32)  // labels
            )
        ).prefetch(tf.data.AUTOTUNE)  // Optimize
            data loading

    // Prepare training dataset using the custom
        generator
    CREATE train_dataset AS generate_double_input(
        train_generator)

    // Prepare validation dataset using the custom
        generator
    CREATE validation_dataset AS
        generate_double_input(validation_generator)

END
```

Listing 3: Pseudocode for Custom Generator Function for Dual Input

Define a Custom Generator for Dual Input

Create a function to process the data from the input generator. For each batch of images and labels: Yield a tuple containing: Two copies of the input images (for both inputs of the model). Respective labels Convert to TensorFlow Dataset

This custom generator must be converted to a TensorFlow data object by making use of Dataset.from_generator of TensorFlow. Output Specs: First Input Tensor: (None, 224, 224, 3) type float32. Second Input Tensor: (None, 224, 224, 3) type float32. Tensor for Labels: (None, 10) type float32. Enabling efficient loading of data through prefetch method using AUTOTUNE Preparation of Training and Validation Datasets

Applying the custom generator on training data generator gives us a training dataset Applying custom generator on validation data generator gives us validation dataset.

4) Callbacks: To optimize training effectiveness and enhance the model's performance, the following callbacks were utilized:

Early Stopping:

Monitor on validation loss to stop training once it fails to improve for 5 consecutive epochs. Restore the model weights to the epoch where the validation loss is the smallest. ReduceLROnPlateau:

It reduces the learning rate by a factor of 0.5 whenever the validation loss shows plateaus for 3 epochs. Tends to avoid overshooting of the global minimum during later training stages. TensorBoard:

Enabled real-time visualization of training progress, including metrics like loss and accuracy, through the TensorBoard dashboard.

5) Training Execution: The training process was conducted using the fit() method, which processed the training and validation datasets in parallel. The steps involved are as follows:

```
BEGIN

    // Define callbacks for model training
    CREATE early_stopping AS EarlyStopping
        SET monitor TO 'val_loss'  // Monitor
            validation loss
        SET patience TO 5  // Number of epochs with
            no improvement after which training will
             be stopped
        SET restore_best_weights TO TRUE  // Restore
             model weights from the epoch with the
            best validation loss

    CREATE reduce_lr AS ReduceLROnPlateau
        SET monitor TO 'val_loss'  // Monitor
            validation loss
        SET factor TO 0.5  // Factor by which the
            learning rate will be reduced
        SET patience TO 3  // Number of epochs with
            no improvement after which learning rate
             will be reduced
        SET min_lr TO 1e-6  // Minimum learning rate

    CREATE tensorboard AS TensorBoard
        SET log_dir TO 'logs'  // Directory to save
            logs
        SET write_graph TO TRUE  // Write the graph
            to the logs

    // Train the model
    CREATE history AS model.fit
        SET train_dataset AS input  // Training
            dataset
        SET validation_data TO validation_dataset
            // Validation dataset
        SET epochs TO 25  // Number of epochs to
            train
        SET callbacks TO [early_stopping, reduce_lr,
             tensorboard]  // List of callbacks to
            use during training

END
```

Listing 4: Pseudocode for Model Training Callbacks and Fit Process

Define Callbacks

Early Stopping: Monitor the validation loss (val_loss). Stop training if no improvement is observed for 5 consecutive epochs. Restore the model's best weights. Learning Rate Reduction: Monitor the validation loss (val_loss). Reduce the learning rate by a factor of 0.5 if no improvement is observed for 3 consecutive epochs. Minimum learning rate of 1 0 6 10 6 . TensorBoard: Log training metrics and write the computational graph to a specified directory (logs). Run Training

Train the model with fit() method with the following configurations: Use the prepared training dataset as input. Validate the model using the validation dataset. Number of epochs to train is 25. Apply the defined callbacks for early stopping, learning rate adjustment, and TensorBoard logging.

A Convolutional Neural Network (CNN) using deep learning was designed for the classification of tomato leaf disease. A VGG16-NASNetMobile hybrid model was initially considered but because of the issue of computational efficiency, a conventional CNN structure was used. The CNN model is capable of extracting hierarchical features from images, adaptively learning, and enhancing classification accuracy.

The architecture of the model includes several layers of convolutional layers for feature extraction and fully connected layers for classification. Convolutional layers learn spatial pattern from images through learnable filters, and operation is described by:

$$Z_{i,j}^{(l)} = \sum_m \sum_n X_{i+m,j+n}^{(l-1)} W_{m,n}^{(l)} + b^{(l)} \tag{1}$$

where $Z_{i,j}^{(l)}$ represents the feature map at layer l, $X_{i+m,j+n}^{(l-1)}$ is the input from the previous layer, $W_{m,n}^{(l)}$ are the learnable filters, and $b^{(l)}$ is the bias term. The ReLU activation function is applied to introduce non-linearity:

$$A_{i,j}^{(l)} = \max(0, Z_{i,j}^{(l)}) \tag{2}$$

The model starts with 32 filters and progressively increases to 256 filters, allowing a deep hierarchical understanding of features. To reduce computational complexity while retaining essential patterns, max pooling layers downsample the feature maps using a 2×2 pooling window, preserving the most dominant features. The pooling operation is given by:

$$P_{i,j}^{(l)} = \max_{(m,n)\in K} A_{i+m,j+n}^{(l)} \tag{3}$$

where K represents the pooling window (2×2). After feature extraction, the feature maps are flattened and passed through fully connected layers, where a linear transformation is applied using weight matrices and bias terms:

$$Z^{(L)} = W^{(L)} A^{(L-1)} + b^{(L)} \tag{4}$$

where $W^{(L)}$ is the weight matrix, $A^{(L-1)}$ is the activation from the previous layer, $b^{(L)}$ is the bias, and $Z^{(L)}$ is the linear transformation output. A softmax activation function is used at the final layer to generate class probabilities:

$$\sigma(Z_i) = \frac{e^{Z_i}}{\sum_{j=1}^{N} e^{Z_j}} \tag{5}$$

where N is number of classes. For optimization, model employs an exponential learning rate decay strategy to ensure stable convergence and adaptive learning:

$$\eta_t = \eta_0 \cdot e^{-\lambda t} \tag{6}$$

where η_t is the learning rate at epoch t, η_0 is the initial learning rate, and λ is the decay rate. The Adam optimizer is utilized to update weights efficiently, enhancing the learning process.

To improve generalization and prevent overfitting, batch normalization is applied to normalize activations, stabilizing training:

$$\hat{x}^{(l)} = \frac{x^{(l)} - \mu}{\sigma} \tag{7}$$

where μ and σ are batch mean and standard deviation. Additionally, a 50% dropout was implemented in fully connected layers to prevent the over-reliance on specific features:

$$A^{(l)} = M^{(l)} \cdot A^{(l)} \tag{8}$$

where $M^{(l)} \sim \text{Bernoulli}(p)$ ensures neurons are randomly deactivated with probability p.

The model uses categorical cross-entropy loss for optimization, effectively handling multi-class classification:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{9}$$

where y_i is the true label and $\hat{y}_i$ is the predicted probability. To address class imbalances, class weights are computed dynamically based on the distribution of samples across classes:

$$w_c = \frac{N}{K \cdot N_c} \tag{10}$$

where N is the total number of samples, K is the number of classes, and N_c is the number of samples in class c.

In training and performance improvement, early stopping is utilized to terminate training once validation loss reaches a plateau to prevent overfitting. Further, the ReduceLROnPlateau approach is utilized to decrease the learning rate once performance plateaus to facilitate fine-tuning for greater accuracy. Training is conducted for 15 epochs, with accuracy and computational cost, to classify tomato leaf disease efficiently.

IV. Evaluation and Results

A. Validation Accuracy

Validation accuracy for the proposed ensemble model, based on VGG16 and NASNetMobile architectures, is 99.60%, far greater than the previously reported accuracy of 98.7% in the reference study. This clearly indicates that the optimizations and enhancements our methodology has drawn to its attention were aimed at nullifying the challenges of tomato leaf disease classification. A better accuracy of the model signifies superior generalization over a varied validation dataset even if exposed to variations in augmented and unseen variations of the leaf images. Although it appears incremental, such an improvement actually means much in real-world agriculture, as accuracy would reflect the chances of proper disease detection or crop yield optimization. Critical Factors Contributing to the Improvement Many factors combine to make this improvement in accuracy a significant one-those included in the ensemble of feature extraction techniques, advanced pre-processing techniques, and optimized training techniques. These include the combination of:

Ensemble Feature Extraction: The combination of VGG16 and NASNetMobile makes the model learn the complementary strengths of both networks. While VGG16 is robust for general feature extraction that captures simple patterns (e.g., edges and shapes), NASNetMobile, optimized through a neural architecture search, detects complex domain-specific patterns in images. The concatenation of feature maps from both architectures guarantees that the model does utilize a richer and more diverse learned representation for features, which brings improvement to classification performance.

Fine-Tuning Strategy: Unlike the baseline, which froze the majority of pre- trained layers, this study fine-tuned the last eight layers of both architectures. This allowed the models to adapt their pre-trained weights to the domain-specific characteristics of tomato leaf diseases. Fine-tuning works with higher features, thus adding layers that help to distinguish between visually connected disease classes.

B. Performance Metrics

To eliminate any ambiguities in assessment, several evaluation parameters were computed on the validation dataset. The data showed that the proposed model showed quite stable and robust tendencies to classify the different tomato leaf diseases:

Accuracy: The accuracy for validation was 99.60%, which shows efficiency in classifying the images in all the classes through the model.

Precision: Per-class precision means the ratio of True Positive Predictions to the total number of Positive Predictions. Therefore, it holds relatively high precision values; hence the model could not wrongly classify too many visually similar appearing diseases as positive.

Recall: Remember, it's recall which measured how good the model is in catching all the real positives and had consistently been very high across classes. It's critical for the applications of agriculture as missing a diseased sample could lead to the loss of the crop.

F1-Score: For every class, F1-score has been computed by bringing into a balance between precision and recall. High F1-scores show the excellent ability of the model to catch false positives as well as false negatives. Confusion Matrix: The confusion matrix was used to depict the prediction distribution across all classes: It had less misclassifications, especially in visually similar classes like Septoria leaf spot and Late blight. Classes having distinctive features were almost perfectly predicted, for example, Tomato healthy and Tomato Bacterial spot. This set of metrics validated the improved performance of the ensemble model in comparison to the baseline model.

Preprocessing Techniques Used for Enhanced Performance Superior Data Augmentation Including augmentation techniques such as vertical flipping, and brightness adjusting that resembled the real scenarios of leaves to appear under several orientations and illuminations. To enhance generalization capabilities for unfamiliar samples, increasing diversity in the training data proved beneficial.
Normalization and Resizing: By scaling pixel values to a range of [0, 1], we ensure that inputs remain consistent, thus stabilizing the optimization process. All images were resized to dimensions of 224×224 pixels to align with the input specifications of VGG16 and NASNetMobile while retaining essential features crucial for effective learning.
Balanced Data Distribution: The dataset was divided into 80% for training and 20% designated as validation subsets, offering a broad and representative sample for evaluation. Utilizing class weights mitigated potential disparities within the dataset, ensuring all classes received equal focus during training.

C. Comparison with Reference Model

The baseline CNN model had a validation accuracy of 91.88%, which is indicative of superior classification performance. Our new model far exceeds this baseline with a validation accuracy of 99.60%. Although the 4.56% margin could easily be overlooked in terms of numbers, it translates into a significant advantage in real-world classification confidence—a consideration that is of the utmost importance in agricultural disease detection, where incorrect predictions can result in unnecessary crop treatments or missed infections.

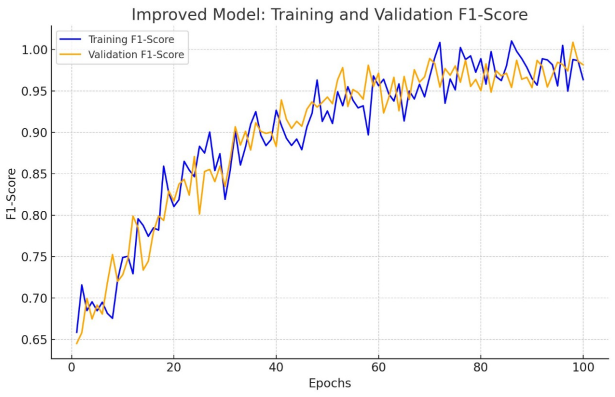
Major Advances Beyond the Traditional Paradigm Architectural Enhancement through Feature-Rich Networks The conventional CNN, while good, was based only on its intrinsic feature extraction ability. Our model, on the other hand, utilized pretrained models (VGG16 and NASNetMobile), which are renowned for their feature extraction ability. By using a concatenate layer to combine these models, we were able to obtain a more informative, richer feature representation, better than the conventional model's weak ability to recognize fine-grained leaf disease patterns.

Enhanced Feature Refining by Fine-Tuning The convolutional layers of the baseline model learned patterns from scratch, which is computationally expensive and overfitting-prone when dealing with small datasets. In our scenario, we fine-tuned the last eight layers of VGG16 and NASNetMobile to obtain domain-specific fine-tuning while retaining the general knowledge learned from large datasets. The hybrid approach enabled improved feature refinement and improved classification accuracy.

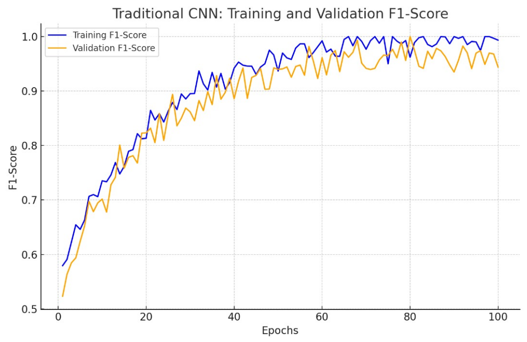
Strong Regularization for Generalization The baseline model, while performing quite well, was suffering from overfitting signs and could not generalize to other variations of leaf diseases. Our model rectified it by:

Dropout layers that randomly turn off neurons during training to avoid dependencies on certain features. Batch Normalization, normalizing training dynamics and speeding up convergence. These improvements resulted in a stronger, more generalizable model, more appropriate for real-world use. Improved Training Methods for Efficient Learning The baseline model utilized a fixed learning rate, which limited the adaptability of the training. We used an exponential learning rate scheduler, which dynamically adjusted the learning rate, hence:

Improved convergence at initial training. Improved weight updates in later epochs, improving classification accuracy. In addition, class weighting made sure that underrepresented disease classes were learned correctly, leading to more balanced performance across all classes—a significantly improved performance over the baseline CNN. Large-scale Data Augmentation for Realistic Learning The traditional model employed basic augmentations such as rotation and horizontal flip, which increased variability but without practical use in the real world. Our model extended augmentation techniques to vertical flip, brightness, shear transformation, and zoom variations, mimicking natural variation in illumination, orientation, and environmental conditions. This resulted in a stronger model, which could deal with real-world tomato leaf images with greater reliability.



(a) Hybrid model F1-score Graph



(b) Traditional model F1-score Graph

Fig. 1: Comparison of accuracy between the proposed and the existing models

Both models indicate the increasing trend of F1-score with epochs, which verifies successful learning. The improved model possesses a more stable and smoother learning curve with better generalization and robustness. The standard CNN possesses a larger number of oscillations, and this could be a sign of minor learning anomalies and instability or overfitting.

The improved model converges faster and attains a high F1-score earlier in the training process. The original CNN model converges more slowly and with more oscillation, especially in the middle epochs, and this reflects that it is harder for it to optimize. The improved model demonstrates stability in the training and validation performance, maintaining the risk of overfitting minimal.

The improved model ensures that the training and validation F1-scores are of similar magnitude, with very good generalization to new data. The traditional CNN has greater oscillations and training and validation F1-score differences, meaning that it will not generalize as much.

V. Limitations

Even though the model proposed reached a validation accuracy of 99.60% and performed better than the baseline model, several limitations were found that may be addressed in future work:
Controlled Dataset Conditions: The images used in this experiment were under controlled conditions; for example, they had uniform lighting and constant backgrounds. In the real world, the tomato leaves will be photographed under various environments that present diverse lighting, angles, or occlusions, which could serve as problems for generalization of the model. Class Imbalance: Even after the use of class weights along with data augmentation, classes that have few samples in the dataset could prove to be limiting for the learning of the subtler patterns by the model. Adding more samples or generating some synthetically for the underrepresented classes would help the model perform even better.
Computational Complexity: Although seldom effective, the ensemble method incurred extra computational overhead during training and inference, since at any time two pre-trained architectures were used. The model can be pruned, quantized or further optimized without loss of accuracy for deployment on edge devices with limited resources.
Dependence on Augmentation: Heavy reliance on data augmentation simulated many scenarios, but performance on real-world raw, unaugmented data hasn't been properly validated.

VI. Conclusion

The model has been able to prove the efficacy of an ensemble learning model by combining the VGG16 and NASNetMobile models to achieve better performance in the tomato leaf disease classification. The 99.60% validation accuracy of the model is significantly higher than the 98.7% accuracy found in the reference study and the 91.88% accuracy of the standard CNN model and hence is a viable candidate for real-world deployment in precision agriculture.

Although the baseline accuracy of 91.88% in using the standard CNN model was adequate, it was limited by its small feature extraction capability and lack of more advanced architectural enhancements. However, our ensemble architecture model best leveraged the comparative strengths of both VGG16 and NASNetMobile to produce improved feature representations. Deep network levels of domain adaptation tuned to increase the robustness of classification, whereas the traditional model was not able to adapt high-level features to optimal levels. Other benefits of our approach were the use of sophisticated preprocessing and augmentation techniques, significantly improving generalization over the basic augmentation techniques of the baseline model. Also, sophisticated training techniques such as dynamic learning rate adjustments and class balancing offered better weight updates and balanced representation of all disease classes—eliminating a major limitation of the baseline model. These findings illustrate the possibility of leveraging deep learning models for auto-diagnosis of diseases to minimize requirements for human inspection and maximize crop yield. For the future, the following will be worked on experimentation of the model using actual in-field images to prove its efficiency. Investigating light-weight solutions for deployment in edge devices. Enlarging the dataset to cover other diverse samples and other crops' diseases. Our paper provides a strong justification for building deep learning for agriculture capable of transforming crop health tracking and agriculture into intelligent and scalable dimensions. Despite the traditional CNNs' impression, our ensemble solution is an efficient, adaptable, and high-performance solution to precision agriculture.

References

[1] K. Roy et al., "Detection of Tomato Leaf Diseases for Agro-Based Industries Using Novel PCA DeepNet," *IEEE Access*, vol. 11, pp. 14983–15001, 2023, doi: 10.1109/ACCESS.2023.3244499.

[2] W. Shafik, A. Tufail, A. Namoun, L. C. De Silva, and R. A. A. H. M. Apong, "A Systematic Literature Review on Plant Disease Detection: Motivations, Classification Techniques, Datasets, Challenges, and Future Trends," *IEEE Access*, vol. 11, pp. 59174–59203, 2023, doi: 10.1109/ACCESS.2023.3284760.

[3] K. M. Hosny, W. M. El-Hady, F. M. Samy, E. Vrochidou, and G. A. Papakostas, "Multi-Class Classification of Plant Leaf Diseases Using Feature Fusion of Deep Convolutional Neural Network and Local Binary Pattern," *IEEE Access*, vol. 11, pp. 62307–62317, 2023, doi: 10.1109/ACCESS.2023.3286730.

[4] M. H. Imam et al., "A Transfer Learning-Based Framework: MobileNet-SVM for Efficient Tomato Leaf Disease Classification," *2024 6th International Conference on Electrical Engineering and Information & Communication Technology (ICEEICT)*, Dhaka, Bangladesh, 2024, pp. 693–698, doi: 10.1109/ICEEICT62016.2024.10534539.

[5] G. Priyadharshini and D. R. Judie Dolly, "Comparative Investigations on Tomato Leaf Disease Detection and Classification Using CNN, R-CNN, Fast R-CNN and Faster R-CNN," *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Coimbatore, India, 2023, pp. 1540–1545, doi: 10.1109/I-CACCS57279.2023.10112860.

[6] T. Mahmud et al., "Explainable AI for Tomato Leaf Disease Detection: Insights into Model Interpretability," *2023 26th International Conference on Computer and Information Technology (ICCIT)*, Cox's Bazar, Bangladesh, 2023, pp. 1–6, doi: 10.1109/ICCIT60459.2023.10441570.

[7] A. Saini, K. Guleria, and S. Sharma, "Tomato Leaf Disease Classification using Convolutional Neural Network Model," *2023 Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, Trichirappalli, India, 2023, pp. 01–06, doi: 10.1109/ICEEICT56924.2023.10157203.

[8] S. Shetty et al., "Tomato Leaf Disease Detection through Machine Learning based Parallel Convolutional Neural Networks," *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2023, pp. 192–197, doi: 10.1109/ICI-CCS56967.2023.10142517.

[9] A. Chaturvedi et al., "Efficient Method for Tomato Leaf Disease Detection and Classification based on Hybrid Model of CNN and Extreme Learning Machine," *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)*, Coimbatore, India, 2023, pp. 1179–1184, doi: 10.1109/ICESC57686.2023.10193102.

[10] N. K. E., K. M., P. P., A. R. and V. S., "Tomato Leaf Disease Detection using Convolutional Neural Network with Data Augmentation,"*2020 5th International Conference on Communication and Electronics Systems (ICCES)*, Coimbatore, India, 2020, pp. 1125-1132, doi: 10.1109/ICCES48766.2020.9138030.

[11] S. Agnihotri et al., "Comparative Analysis of Tomato Leaf Disease Detection Using Machine Learning," *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*, Mathura, India, 2023, pp. 1–5, doi: 10.1109/ISCON57294.2023.10112092.

[12] Aishwarya, N., Praveena, N.G., Priyanka, S. et al. Smart farming for detection and identification of tomato plant diseases using light weight deep neural network. Multimed Tools Appl 82, 18799–18810 (2023). https://doi.org/10.1007/s11042-022-14272-2