PhytoNet: Mish-Optimized Deep Learning Architecture for Enhanced Tomato Leaf Disease Detection

Deependra Rastogi
IILM University
Prashant Johri
Galgotias University
Varun Tiwari
varun.tiwari@jaipur.manipal.edu
Manipal University Jaipur
Anand Singh
IILM University
Sumit Singh Dhanda
IILM University
Atul Kumar Singh
IILM University
Gaurav Kumar
IILM University

Research Article

Keywords: Plant Leaf Disease Identification, Deep Learning, SqueezeNet, Mish-Optimized Function

Posted Date: March 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8970511/v1

Additional Declarations: No competing interests reported.

PhytoNet: Mish-Optimized Deep Learning Architecture for Enhanced Tomato Leaf Disease Detection

Deependra Rastogi[1], Prashant Johri[2], Varun Tiwari[3], Anand Singh[4], Sumit Singh Dhanda[5], Atul Kumar Singh[6], Gaurav Kumar[7]

[1,4,5,6,7]*School of Computer Science and Engineering, IILM University, Greater Noida, UP, 201306, India.*
[3]*School of Computer Science and Engineering, Manipal University Jaipur, Jaipur, Rajasthan, India*
[2] *School of Computer Science and Engineering, Galgotias University, Greater Noida, Noida, UP, 203201, India.*

[1]*deependra.libra@gmail.com,* [2]*johri.prashant@gmail.com,* [3]*varun.tiwari@jaipur.manipal.edu,*
[4]*anandsingh7777@gmail.com,* [5]*dhandasumit@gmail.com,* [6]*er.atul.kumar.singh@gmail.com,*
[7]*midhagaurav2004@gmail.com*

Corresponding Author: Varun Tiwari

Abstract

Tomato leaf diseases significantly impact agricultural productivity, necessitating accurate and efficient diagnostic methods. Deep learning has emerged as a robust approach for plant disease detection, but challenges such as inefficient feature extraction, classification, model complexity and limited computational resources hinder its widespread adoption. This study introduces a PhytoNet (Mish-Optimized SqueezeNet) Framework to enhance tomato leaf disease prediction. The SqueezeNet architecture, known for its lightweight design, is optimized with the Mish activation function to improve feature extraction and classification capabilities while maintaining computational efficiency. The methodology involves training the SqueezeNet model on 10 classes of mendaley dataset of tomato leaf images, encompassing multiple disease classes and healthy samples. Data preprocessing techniques, including image augmentation and normalization, are employed to ensure model robustness. The integration of the Mish activation function in critical layers enhances non-linearity, aiding in better gradient flow and improved performance during training. Model evaluation is conducted using metrics such as accuracy, precision, recall and F1-score. Experimental results demonstrate that the PhytoNet outperforms traditional SqueezeNet and other lightweight architectures in terms of classification accuracy, achieving over 0.9957 accuracy on the dataset. Additionally, the model maintains low computational overhead, making it suitable for deployment on resource-constrained devices. Hence, the proposed framework effectively balances accuracy and efficiency, addressing critical limitations in existing plant disease detection models. This work underscores the potential of lightweight and activation-optimized deep learning frameworks for real-time agricultural applications, paving the way for scalable and sustainable solutions in precision farming.

Keyword: Plant Leaf Disease Identification, Deep Learning, SqueezeNet, Mish-Optimized Function.

1. Introduction

Rapid farming practices, to reduce costs, increase production, and minimize environmental impact [Cardellicchio et al., 2023, Zarboubi et al., 2025]. This study focuses on the tomato plant (Solanum lycopersicum), a vital staple crop globally recognized for its versatility, nutritional value, and economic significance in various agricultural sectors. Native to South

America and a member of the nightshade family, Solanaceae, the tomato is extensively cultivated for its edible fruit. Tomatoes are rich in vitamins, minerals, and antioxidants, making them a crucial component of diets worldwide. The plant thrives in warm, sunny climates and requires well-drained, fertile soils for optimal growth. Tomato cultivation faces threats from various pests and diseases, such as fungal infections and bacterial blights, which can severely diminish both yield and fruit quality [Agarwal et al, 2023, Das et al., 2025].

Two of the most devastating diseases affecting tomatoes are late blight (Phytophthora infestans) and early blight (Alternaria solani), both of which can lead to catastrophic crop losses if not managed effectively. Late blight, which thrives in cool, wet conditions, can rapidly destroy entire crops if not managed promptly, while early blight is particularly problematic in warm, humid environments, leading to reduced fruit production and defoliation [Cheng et al., 2022]. Other significant threats include soil-borne diseases like fusarium wilt and verticillium wilt, bacterial diseases such as bacterial spot and bacterial wilt, and fungal infections like powdery mildew. These diseases not only reduce the quantity of tomatoes produced but also compromise their quality, making them less marketable, which in turn impacts farmer income and overall food security.

Traditional methods for managing these diseases—such as using resistant varieties, crop rotation, and integrated pest management—are essential but often prove inadequate against rapidly evolving pathogens and changing environmental conditions. Timely identification of plant diseases is critical to preventing outbreaks and mitigating losses, as delays can result in widespread crop failure and significant economic repercussions for farmers.

Convolutional neural networks (CNNs) have emerged as a leading approach for addressing complex image processing challenges, particularly in agricultural settings with intricate backgrounds and inconsistent lighting; however, they also face challenges such as variability in image quality and the need for extensive training datasets [Vini and Rathika, 2024, Ghosh et al., 2025]. CNNs are especially well-suited for tasks like detecting and categorising plant diseases because of their exceptional efficacy in automatically learning and extracting information from images. Agarwal et al. (2020), for example, created a customised CNN model that successfully identified nine different tomato illnesses, proving its efficacy even when photos were taken in a variety of environmental settings. A branch of artificial intelligence called deep learning has quickly become well-known in the field of agriculture, transforming how academics and farmers tackle problems like management of resources, yield prediction, and disease diagnosis. More effective, accurate, and sustainable agricultural methods are becoming more and more necessary as the world's food need rises [Abbas et al., 2021, Sun et al., 2025]. Deep learning offers powerful tools to address these needs, enabling the analysis of vast amounts of data and the automation of complex tasks that were once labour-intensive and time-consuming.

This study introduces the innovative PhytoNet framework, designed to enhance the capabilities of CNNs specifically for tomato leaf disease prediction, providing a more efficient and accurate solution in agricultural applications. SqueezeNet, recognized as a lightweight deep learning model, offers efficiency that makes it particularly attractive for agricultural applications, allowing farmers to implement advanced disease detection without the need for extensive computing resources. While SqueezeNet is efficient, there remains room for performance improvements, especially in tasks requiring detailed and nuanced image analysis, potentially

achieved through the integration of advanced techniques such as specialized activation functions. The Mish activation function, a relatively recent development in deep learning, offers potential improvements over traditional activation functions like ReLU (Rectified Linear Unit). Mish is characterized by its smooth and non-monotonic properties, which help capture more complex patterns in data, leading to improved learning and generalization capabilities of neural networks. By integrating Mish activation into the SqueezeNet architecture, this study seeks to enhance the model's ability to detect and classify various tomato leaf diseases with greater accuracy and efficiency. The Mish activation function, a recent advancement in deep learning, provides advantages over traditional activation functions like ReLU (Rectified Linear Unit) by capturing more complex patterns in data due to its smooth, non-monotonic properties. This hybrid approach optimizes the model for real-world agricultural applications, where a balance between model efficiency and predictive performance is crucial. The goal is to provide a practical tool for farmers and agricultural experts, enabling more effective crop management and ultimately contributing to global food security.

The main contribution of this study is as follow:

- **Combines Lightweight Architecture with Advanced Activation:** Integrates the lightweight SqueezeNet architecture with the Mish activation function, resulting in an efficient model tailored for environments with limited computational resources.
- **Enhances Training Efficiency:** Improves model training efficiency by utilizing the Mish activation function, which accelerates convergence during the learning process, leading to faster and more effective training.
- **Increases Reliability and Generalizability:** Enhances the model's performance under diverse environmental conditions and lighting scenarios, thereby increasing the framework's reliability and generalizability in real-world agricultural applications.

This paper's remaining sections are arranged as follows: Prior research is reviewed in Section 2, and the materials and redesigned approach are described in Section 3. Results, assessment, comparative analysis, and discussion of the findings are presented in Section 4. Concluding thoughts and recommendations for more study are finally included in Section 5.

2. Related Work

In order to train a DenseNet121 model by transfer learning, Abbas et al. 2021 created synthetic tomato leaf pictures using C-GAN and mixed them with real ones. The model's accuracy for five, seven, and ten illness categories was 99.51%, 98.65%, and 97.11%, respectively, when tested on the PlantVillage dataset. Al-Shamasneh and Ibrahim (2024) suggested a feature extraction method that uses conformable polynomials to increase detection accuracy and speed. When tested on the same dataset using an SVM classifier, it obtained 98.80% accuracy, which might help increase agricultural yields and minimise financial losses. The authors of this study, Badiger and Mathew, 2023, use deep batch-normalized eLu AlexNet (DbneAlexNet), a deep learning technique for identifying and categorising tomato illnesses. After removing distortions from the leaf pictures through preprocessing, U-Net segmentation optimised by the Gradient-Golden Search Optimisation (GSO) technique is performed. Prior to classification, the segmented pictures are enhanced using DbneAlexNet, which was trained using a novel Gradient Jaya-Golden Search Optimisation (GJ-GSO) technique. With a low false positive rate of 0.078, a rate of true positives of 91.9%, a true negative rate of 92.2%, and an accuracy of 92.4%, the model demonstrated its efficacy in identifying plant diseases. Baser et al. 2023

introduced the TomConv model, achieving 98.19% accuracy in classifying tomato leaves into ten categories using an enhanced CNN on the PlantVillage dataset. Cardellicchio et al. 2023 evaluated YOLOv5-based detectors, both standalone and ensemble, for identifying tomato plant traits like nodes, fruit, and flowers, with effective performance on a complex dataset from a stress experiment involving various genotypes. Cheemaladinne and K 2024 discussed the "VARMAx-CNN-GAN Integration" framework, combining CNNs for feature extraction with GANs for generating synthetic images. This approach improves accuracy, precision, recall, and response time in tomato disease classification, offering valuable insights for better disease management. Cheng et al. 2022 developed an automated system using CNNs and a chatbot to identify eight tomato diseases and pests. Tested on 9,000 images, the system achieved 97.40% accuracy for anomaly detection, 93.63% for disease identification, and 98.70% for leaf mold/powdery mildew detection. Integrated with LINE, it helps farmers manage diseases efficiently. A technique for categorising tomato leaf diseases using several parallel CNNs with different configurations was highlighted by Islam et al. in 2022. With the use of normalisation approaches and activation layers such as Swish and Elu-Swish, the method achieves above 99.0% accuracy in training, 97.5% in validation, and 98.0% in testing. The technique improves deep learning for identifying plant diseases and overcomes prediction difficulties by combining outputs from many networks, potentially having worldwide agricultural advantages. A modified Mask Region Convolutional Neural Network (Mask R-CNN) for tomato leaf disease detection was presented by Kaur et al. in 2022. To save memory use and computational expenses, the model makes use of a simplified "Region Convolutional Neural Network (R-CNN)" head. In comparison to current techniques, it achieves a Mean Average Precision (mAP) of 0.88, an F1-score of 0.912, and 0.98 accuracy while decreasing the lesion detection time through feature extraction and anchor proportion optimisation. Kaushik et al., 2023 covered the TomFusioNet framework for mobile tomato crop analysis. It uses late-fusion and transfer learning techniques, with Multi-layer Perceptron models as meta-learners. The framework includes DeepRec for initial disease recognition and DeepPred for detailed illness identification, optimized with NSGA-II. Featuring HSV-based background elimination, TomFusioNet achieved accuracies of 99.93% and 98.32%, respectively. This literature Li et al., 2023 discussed LMBRNet, a model designed for identifying tomato leaf diseases. LMBRNet employs a grouped differentiated residual (CGDR) and deep separable convolution to enhance both accuracy and efficiency. It achieves a high accuracy of 99.7% with only 4.1M parameters, surpassing traditional models like ResNet50 and GoogleNet in performance, and offers better parameter efficiency compared to InceptionResNetV2 and MobileNetV2. The model exhibits robust performance across multiple datasets.

Nag et al., 2023 developed a mobile app for tomato leaf disease detection using CNNs. The study fine-tunes AlexNet, ResNet-50, SqueezeNet-1.1, VGG19, and DenseNet-121, achieving over 95% accuracy across models, with DenseNet-121 reaching 99.85%. DenseNet-121 is recommended for the app, which features a user-friendly interface in English and two major Indian languages, improving accessibility. Multi-scale residual modules are used in the Perveen et al., 2023 model to accommodate different lesion sizes and shapes. Lesion detection is improved by the Classifier and Bridge (CB) module, which links segmentation and feature extraction. Up-sampling and convolution are employed to deconvolutate the activation map during segmentation, and a skip connection is subsequently utilised to fuse it with low-level data. The network is trained to concentrate on lesion areas using a binary cross-entropy loss function and a restricted set of pixel-level labels.

Table 1 Related Work

S No.	Ref No	Work Done	Model Used	Dataset	Plants/Classes	Accuracy	Limitation	Research Gap
1	Abbas et al. 2021	Deep Learning-Based methodology for Tomato Disease Detection	Conditional Generative Adversarial Network (C-GAN) + DenseNet121 with Transfer Learning	PlantVillage	5 Classes, 7 Classes and 10 Classes for Tomato Leaf Image	5 Classes: .9951 7 Classes: .9865 10 Classes: .9711	Reliance on the quality and diversity of synthetic images	Extend disease detection to other plant parts and different disease phases
2	Al-Shamasneh and Ibrahim, 2024	Classification of tomato leaf images for detection of plant disease	Support Vector Machine	PlantVillage	14 Different Classes	0.988	Impacted by artifacts on images of tomato leaf, which might lead misclassification because certain diseases exhibit symptoms that are identical on the leaves.	Multi-infection Type
3	Badiger and Mathew, 2023	Deep Learning-assisted approach for detect and classify the tomato disease and utilized deep batch-normalized eLu Alex Net (DbneAlexnet) model for the classification the tomato plant leaves.	GJ-GSO-based DbneAlexNet	PlantVillage	10 Classes	0.91	Reliance on augmentation techniques such as position and color augmentation	Synthetic data generation

S No.	Ref No	Work Done	Model Used	Dataset	Plants/Classes	Accuracy	Limitation	Research Gap
4	Baser et al. 2023	Classification among 10 different categories of tomato plant leaves using the TomConv model which deploys an improved Convolutional Neural Network (CNN).	TomConv	PlantVillage	10 Different Classes	0.9819	Does not address the potential of leveraging advanced data augmentation techniques	Detect disease from other parts of tomato plant like root, stem and flower
5	Cheemaladinne and K 2024	Tomato Leaf Disease Detection	VARMAx-CNN-GAN Integration	350,000 images dataset		0.92	Inadequate feature representation, Class Imbalance	To Augment Training Data, Improve Faturee Representation
6	Cheng et al. 2022	Convolutional neural networks for tomato leaf disease detection in actual field circumstances	LPDM	PlantVillage	8 Classes	0.987	Model's generalizability in real-world scenarios, Pesticide Recommendation Concerns	Extend datasets to encompass varied stages, circumstances, and disease stages.
7	Islam et al. in 2022	Disease identification of Tomato Leaf	Improved AlexNet	2018 AI Challenger	9 Classes	0.9872	Concentrate disease identification on single leaves This research does not focus on the the progression of diseases illnesses over time or how they different across growth stages.	Handling of multiple diseases, and early-stage detection capabilities.

S No.	Ref No	Work Done	Model Used	Dataset	Plants/Classes	Accuracy	Limitation	Research Gap
8	Kaur et al. in 2022	Characterization of contaminated area in tomato leaf disease and object identification methodology	Modified Mask Region Convolutional Neural Network (Mask R-CNN)	PlantVillage	1610 Images	0.98	Detection is reliant on precise ground-truth data	Identification of multiple stresses is not done.
9	Kaushik et al., 2023	End-to-end tomato crop analysis framework	TomFusionNet (DeepRec and DeepPred)	PlantVillage	14 Classes	DeepRec: .9993 DeepPred:.9832	Predictive analysis for multiple crops	End-to-end multi-objective optimization
10	Nag et al., 2023	Tomato disease identification with fine-tuned convolutional neural networks	Fine-tunes 5 CNNs, with DenseNet-121	PlantVillage	10 Classes	Over .99 for all CNN technique	Limited diversity of the dataset Poor generalizability across regions and inability to handle unseen diseases	Bandwidth limitations, and challenges in ensuring compatibility across diverse mobile platforms
11	Perveen et al., 2023	Tomato leaf disease diagnosis and segmentation of lesions	CNN-based U network model	PlantVillage	9 Classes	0.992	Cross-species Applicability, Integration of Biological and Technological Control Methods	Model's versatility, Data augmentation techniques to improve the model's robustness
12	Russel and Selvaraj, 2022	Classification using Multiscale parallel Deep CNN	CNN	PlantVillage	39 Classes	0.9917	Require a large amount of data and processing power.	Architecture optimisation and transfer learning potential.

S No.	Ref No	Work Done	Model Used	Dataset	Plants/Classes	Accuracy	Limitation	Research Gap
13	Zhang and Chen, 2022	IoT-enabled energy-efficient approach	ANN	PlantVillage	9 Classes	0.92	Struggles with interpretability and unbalanced datasets.	Adjusting parameters and resolving class disparity.
14	Bhagat and Kumar, 2023	Feature Selection for Leaf Disease Identification	BoWs and SURF Method	PlantVillage	14 Classes	0.97	Prone to overfitting and instability.	Ensemble techniques for generalisation and stability.
15	Zhou et al., 2021	Tomato Leaf Disease Identification	Restructural Residual Dense network	AI Challenger	9 Classes	0.95	Prone to overfitting without regularization.	Regularization techniques and hybrid models.
16	Alzahrani and Alsaade, 2023	Early Detection and Recognition of Tomato Leaf Disease	DenseNet169, ResNet50V2 and ViT	tomatoleaf	10 Classes	DenseNet169: .9900 ResNet50V2: .9560 ViT: .9800	Sensitive to initialization.	Robust initialization strategies and alternative models.
17	Nagaraju et al., 2021	Leaf disease detection using augmentation techniques	IPTA and Image Masking and REC-based hybrid segmentation algorithm	Maize Leaf Disease	4 Classes/ 4188 images	.74	Minimize misclassification errors effectively	Lack of robust strategies
18	Mishra et al., 2022	Weed density estimation in Soya bean crop	Inception V4	CFWID	4384 Images	.982	Environment Variability	Quantification of weed density
19	Kaur et al., 2022	Leaf Disease Using Hybrid Convolutional Neural Network	EfficientNet B7 deep architecture	PlantVillage	4 Classes	.987	Limited data availability mitigated	Fine Tune in DL Model

The review emphasises the significant developments in deep learning-based method for tomato leaf disease identification, highlighting the innovative framework, optimization strategies, and variety datasets employed. Despite the outstanding achievements, problem such as poor generalizability to actual conditions, dependence on controlled environments, and failure of pay attention on multiple plant sections and disease phases persist across exist. Common research gaps include the need for diversity of dataset, better augmentation approach, and deals to handle multi-stress situations and early-stage detection.

3. Materials and Methods

3.1 Dataset Acquisition

The study utilized a dataset available on Mendeley [J and Gopal, 2019], which comprises 39 distinct classes of plant leaf and background images, providing a wide variety of visual information. The dataset contains 61,486 images in total, demonstrating its richness and versatility for deep learning experiments. For this specific research, 20,070 images of tomato leaves were selected, all resized to a standardized dimension of 256x256 pixels for uniformity and efficiency in training deep learning models.

Table 1 Image Count Class wise

S No	Class Category	Number of Images
1	bacterial_spot	2123
2	early_blight	2621
3	healthy	2327
4	late_blight	2537
5	leaf_mold	1904
6	mosaic_virus	1641
7	septoria_leaf_spot	1771
8	target_spot	1554
9	twospotted_spider_mite	1676
10	yellow_leaf_curl_virus	4039

i. The subset focuses on 10 classes, including 9 disease categories and 1 healthy leaf category. Each disease represents unique characteristics like spots, yellowing, molds, or deformation, essential for accurate identification.
ii. The images were captured in both laboratory settings and real-world scenarios. This dual-sourced data enhances the model's ability to generalize and perform well across different conditions, such as varying lighting, backgrounds, and leaf orientations.
iii. The dataset is already data-augmented, meaning it includes transformations like rotations, flips, and scalings. This augmentation ensures robust model performance by exposing it to diverse versions of the same disease patterns.

3.2 Dataset Characteristic

For tomato leaf disease classification, the dataset with 20,070 images has been developed to support lightweight, mobile-ready machine learning models that can classify various diseases offline. Here are the primary characteristics:

Table 2 Primary Characteristics

Class	Symptoms
Bacterial Spot	**Spots:** Small, dark, greasy-looking lesions on leaves and fruits. **Progression:** Spots enlarge and coalesce, leading to yellow halos around lesions. **Texture:** Water-soaked and raised, especially under high humidity. **Severity:** Causes significant defoliation and reduced fruit yield.
Early Blight	**Spots:** Brown lesions with characteristic concentric rings, resembling a target. **Spread:** Begins on older leaves near the soil line and spreads upwards. **Other Symptoms:** Leaf yellowing and premature shedding. Impact: Leads to stunted growth and reduced fruit size.
Healthy	**Appearance:** Uniform green coloration with no visible spots, discoloration, or curling. **Structure:** Well-formed and evenly shaped leaves.
Late Blight	**Lesions:** Large, irregular, water-soaked spots that turn brown or black. **Additional Signs:** White fungal growth on the underside of leaves during humid conditions. **Effect:** Rapid defoliation and collapse of plants.
Leaf Mold	**Upper Side:** Yellow patches. **Underside:** Velvety olive-green mold growth. **Impact:** Leads to curled and shriveled leaves. **Spread:** Common in greenhouses with poor ventilation.
Mosaic Virus	**Pattern:** Distinct mottled appearance with alternating yellow and green patches. **Deformation:** Leaves may become wrinkled or twisted. **Impact:** Stunts growth, reduces leaf surface area, and weakens the plant.
Septoria Leaf Spot	**Spots:** Tiny, circular spots with dark brown margins and lighter centers. **Progression:** Spots enlarge, turn gray, and may have small black specks (pycnidia). **Effect:** Causes significant defoliation, exposing fruits to sunscald.
Target Spot	**Lesions:** Large, circular spots with concentric zones of light and dark brown. **Spread:** Rapid under warm, moist conditions. **Impact:** Leads to necrosis, reducing photosynthetic capacity.
Twospotted Spider Mite	**Damage:** Tiny specks (stippling) appear on leaves as chlorophyll is removed. **Webbing:** Fine webbing on the underside of leaves in severe infestations. **Effect:** Yellowing and browning of leaves, leading to defoliation.
Yellow Leaf Curl Virus	**Curling:** Leaves curl upward and show interveinal yellowing. **Growth:** Plants become stunted with smaller, misshapen leaves. **Impact:** Severely limits fruit production due to reduced plant vigor.

3.3 Dataset Preparation

This step focuses on carefully loading, visualizing, and preparing the dataset for training a model to classify tomato leaf diseases. Each aspect is addressed systematically, making the process more engaging and visually clear.

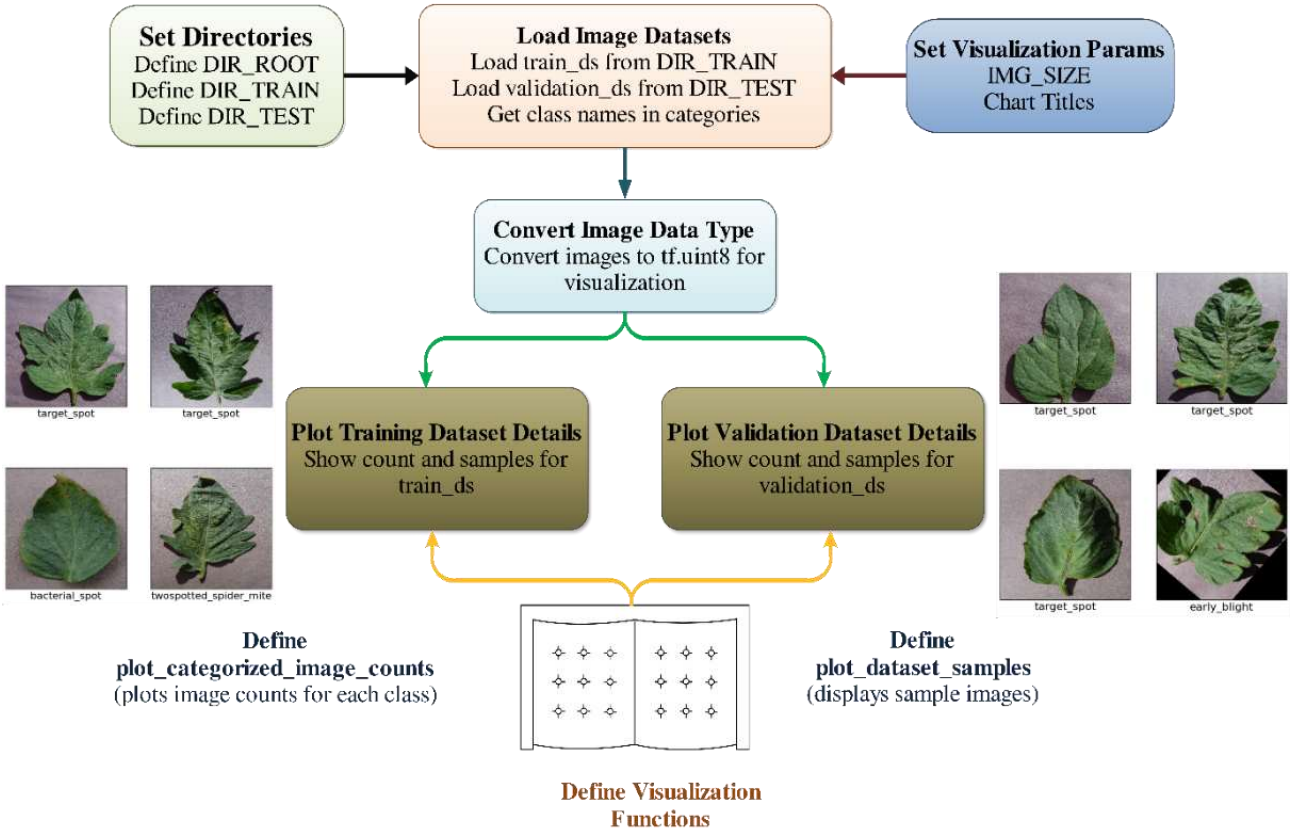

Fig 1 Data preparation flow

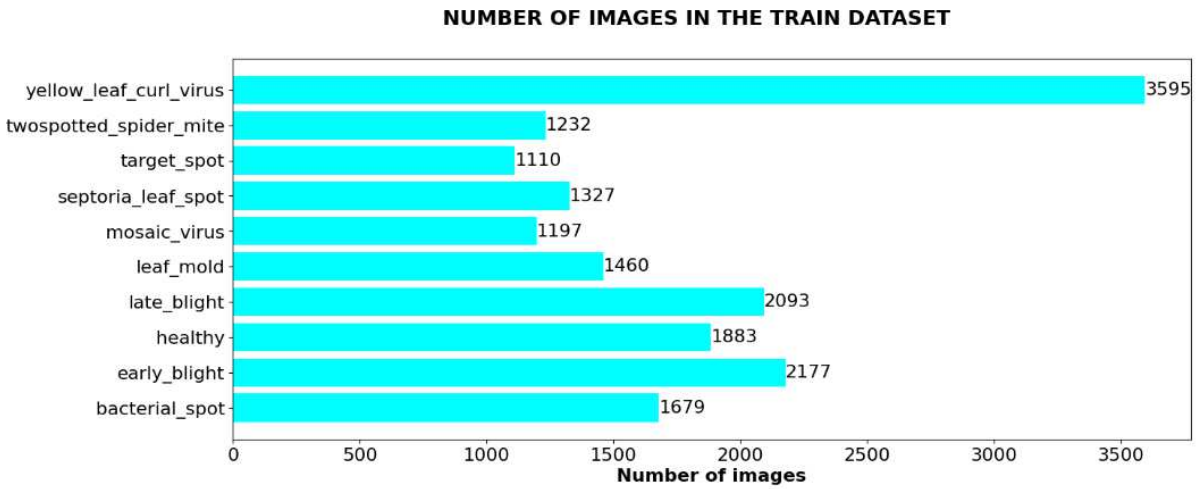

Fig 2 Number of Images in the Train Dataset

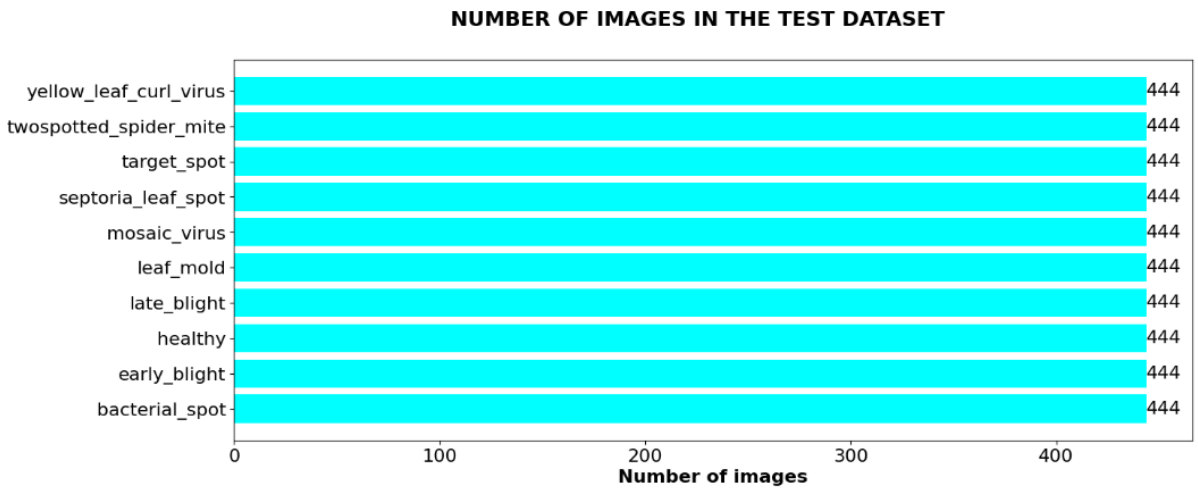

Fig 3 Number of Images in the Test Dataset

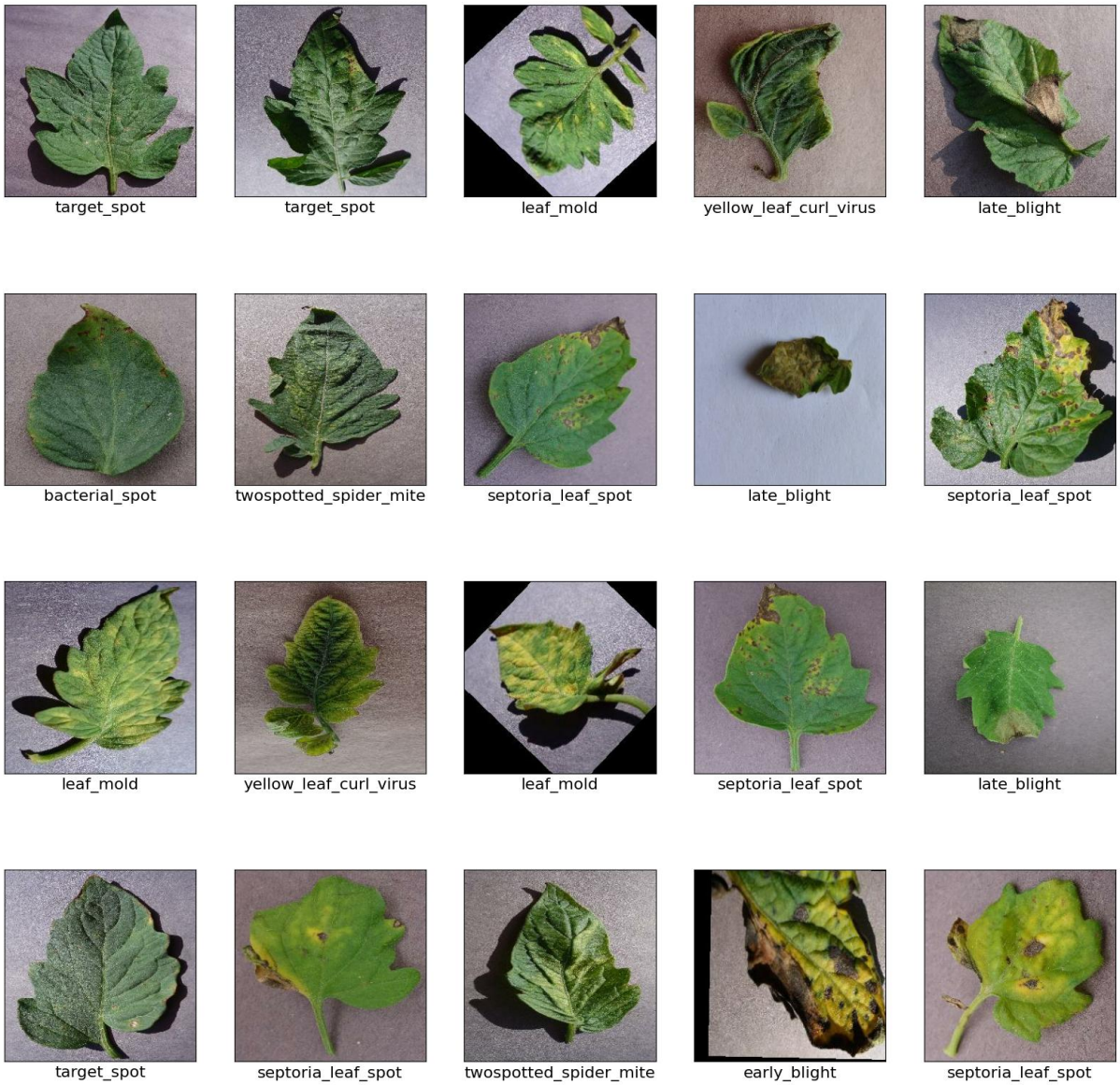

Fig 4 Images for the Train Dataset

This process begins by organizing the dataset paths through structured definitions. The root directory (DIR_ROOT) points to the main dataset folder, with subdirectories for training (DIR_TRAIN) and testing (DIR_TEST) dynamically constructed using Python's f-strings. This logical organization ensures smooth access to data for subsequent tasks. To facilitate clear data analysis, visualization parameters are set. Images are resized to a uniform dimension of 256x256 for visual consistency during exploration. Meaningful titles, like TRAINING_IMAGES_CHART_TITLE, ensure that charts are user-friendly. Although visualization relies on 256x256 images, resizing to 224x224 may later be needed for model training, aligning with standard pre-trained model requirements. Next, the dataset is loaded efficiently using TensorFlow's keras.utils.image_dataset_from_directory, which automates image loading from specified directories, assigns labels based on folder names, and resizes images to the defined size. The batch_size parameter is set to None to keep the dataset flexible. Categories (class names) are extracted into the categories variable, streamlining access to disease labels.

To prepare for visualization, the script maps image-label pairs to tf.uint8 format using TensorFlow's map function, ensuring compatibility with plotting tools. This step processes both training (train_ds) and validation (validation_ds) datasets, enabling seamless data exploration. The plot_categorized_image_counts function generates a visually appealing horizontal bar chart to display image counts per category. It uses a helper function, count_categorized_images, to calculate counts and annotates bars with precise values for better insights. The chart is further enhanced with bold titles and readable fonts.

For sample visualization, the plot_dataset_samples function extracts and displays the first 20 images in a 4x5 grid. Each image is paired with its label, providing a quick validation of dataset quality and consistency, helping identify potential anomalies. Finally, the script runs the defined functions to analyze the dataset. Charts showcasing image counts across categories in training and testing datasets offer an overview of distribution. Sample visualizations display representative images from each dataset, making the dataset more accessible and understandable for analysis and model preparation.

Table 3 Train and Test Image Count Class wise [J and Gopal, 2019]

S No	Class Category	Train	Test
1	bacterial_spot	1679	444
2	early_blight	2177	444
3	healthy	1883	444
4	late_blight	2093	444
5	leaf_mold	1460	444
6	mosaic_virus	1197	444
7	septoria_leaf_spot	1327	444
8	target_spot	1110	444
9	twospotted_spider_mite	1232	444
10	yellow_leaf_curl_virus	3595	444

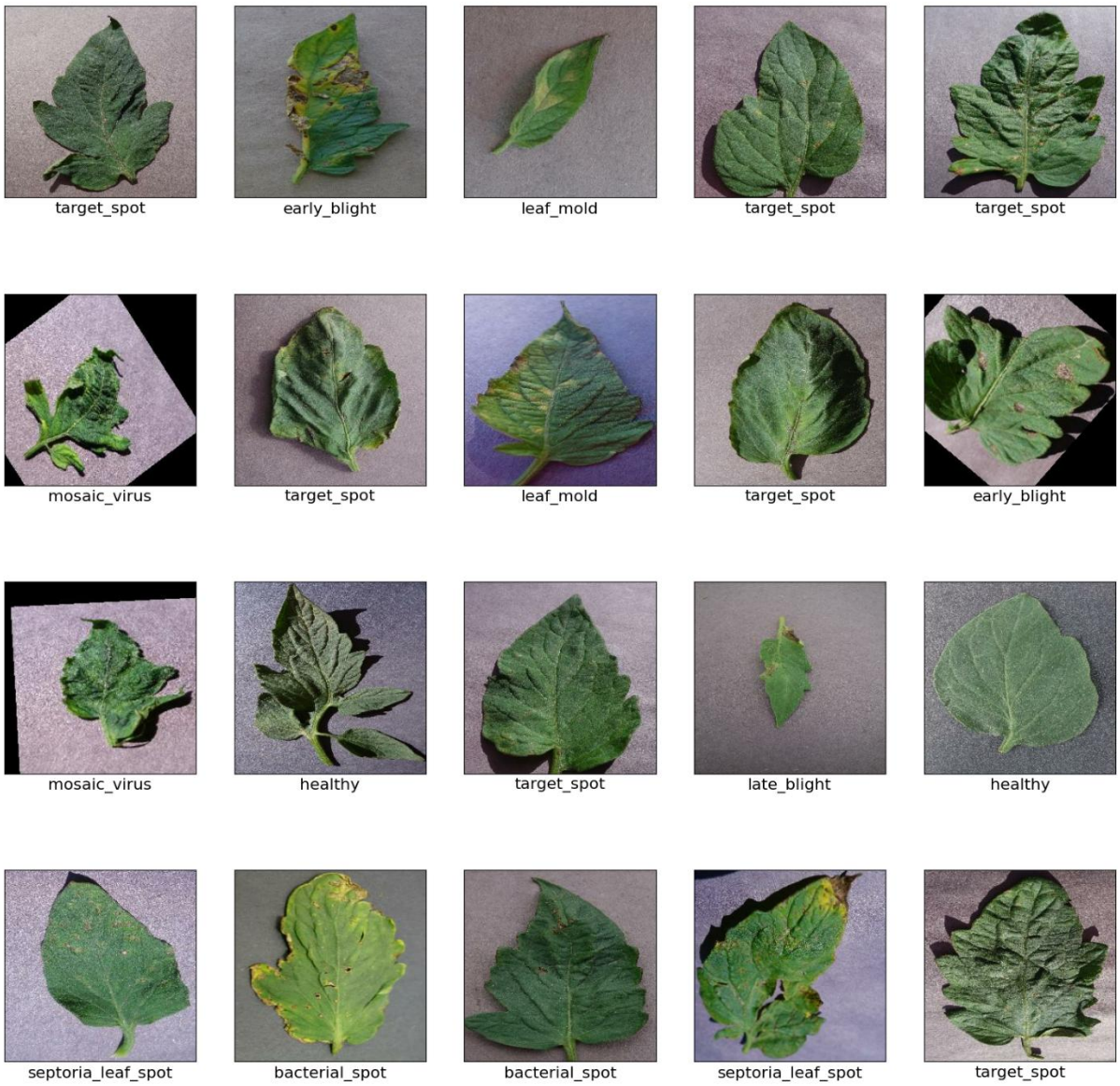

Fig 5 Images for the Test Dataset

3.3 Data Pre-processing Strategy

This study exemplifies a well-defined computational pipeline focusing on data preprocessing, a critical step for building machine learning models, especially for tasks like classifying tomato leaf diseases. Data Preprocessing in Deep Learning is an essential step to transform raw data into a format suitable for model training [Polly and Devi, 2024]. This process ensures data consistency, quality, and relevance, which significantly impacts the performance of deep learning models.

The definition of dataset paths employs a modular design, ensuring ease of data access and scalability. This approach ensures data modularity and reduces the risk of errors during directory referencing [Ullah et al., 2023b]. By using Python's f-strings, the paths dynamically adapt to variations in root directory structure, simplifying deployment across environments.

Such automation is crucial in large-scale datasets where logical organization improves data access efficiency and reliability.

This study defines parameters for consistent visualization:

$$\text{IMG\_SIZE} = (256, 256)$$

Resizing all images to uniform dimensions ensures that each sample contributes equally during training, avoiding bias from dimensional inconsistencies. For visualization, this uniformity highlights the dataset's overall structure. However, resizing to 256×256 for visualization contrasts with 224×224, commonly used for models like ResNet and EfficientNet, which expect inputs adhering to this size. This flexibility showcases adaptability to specific architectural requirements.

TensorFlow's automates dataset loading with the following steps:

Step 1. **Image-Label Association**
Each directory's name is mapped to a unique label C_i. Given N directories, the mapping can be expressed as:

$$C_i = Label\ Map: \{0,1,2, \dots, N-1\}$$

Step 2. **Dynamic Image Resizing**
All images I(x,y) are resized to a fixed size:

$$I'(x', y') = Resize\ (I(x, y), IMG\_SIZE)$$

Step 3. **Data Structure Optimization**
By setting batch_size=None, images are left ungrouped, enabling flexible batching strategies during training. This unbatched design aligns with computational paradigms where on-the-fly transformations enhance pipeline flexibility.

Images are prepared for visualization by converting them into an unsigned integer format:

$$I' = tf.cast\ (I, tf.uint8)$$

This step ensures compatibility with plotting libraries, preventing computational errors during rendering. The function operates on pairs (I, L), maintaining label integrity during transformations. The component plot_categorized_image_counts applies a structured approach to analyze and visualize the distribution of images across dataset categories. For a dataset D with n categories, it determines the count of images $N(C_i)$ in each category C_i using the formula:

$$N(C_i) = \sum_{j=1}^{|D|} 1(I_j \in C_i)$$

Where:

- $N(C_i)$: Count of images in category C_i,

- $1(I_j \in C_i)$: Indicator function returning 1 if image I_j belongs to category C_i, otherwise 0.

This approach establishes a highly efficient and reproducible system for dataset preprocessing and visualization. By employing mathematical precision to maintain uniformity and leveraging algorithmic enhancements for optimal data management, it lays a solid foundation for effective model training and comprehensive evaluation.

3.4 Adapted Methodology

3.4.1 SqueezeNet

In contrast to more conventional architectures like AlexNet and VGGNet, SqueezeNet is a lightweight convolutional neural network (CNN) built for effective performance with a substantially fewer number of parameters. It is appropriate for resource-constrained situations like mobile devices and the Internet of Things since it uses 50 times fewer parameters to achieve AlexNet-level accuracy on the ImageNet dataset. The architecture uses cutting-edge design techniques to minimise the model's size without sacrificing functionality. It accomplishes this by utilising squeeze layers to reduce the input channels to 3×3 filters, removing fully linked layers in favour of global average pooling, and substituting computationally costly 3×3 convolutions with 1×1 filters [Tsang, 2021].

The model constitutes of **convolution layers, fire modules, and pooling layers**, highlights the fundamental building blocks of the SqueezeNet architecture, presenting an overview of its structural design. Deeper analysis is made possible by the convolution layers, which are in charge of removing low-level information from the input image, such as edges and textures [Iandola et al., 2016]. The fire modules, as the core innovation of SqueezeNet, combine squeeze and expand operations to reduce computational complexity while maintaining rich feature representation. Pooling layers complement this structure by downsampling feature maps, reducing spatial dimensions, and conserving computational resources [Tsang, 2021].

In order to extract significant spatial and hierarchical characteristics from the input data, a convolutional layer—a basic part of convolutional neural networks, or CNNs—applies convolution operations.

The key components of the convolutional layer are like that

- Input Data: The data fed into the layer, such as an image or feature map, typically represented as a 3D matrix (height, width, and channels).
- Filters (Kernels): Small matrices (e.g., 3×3, 5×5) that slide over the input, performing element-wise multiplications followed by summation to compute feature maps.
- Stride: The number of steps the filter moves during convolution. Larger strides reduce the spatial size of the output.
- Padding: Extra rows/columns added around the input to control the output size. Common types include:
    - Valid Padding: No padding, reducing the spatial dimensions.
    - Same Padding: Adds padding to maintain the same dimensions as the input.

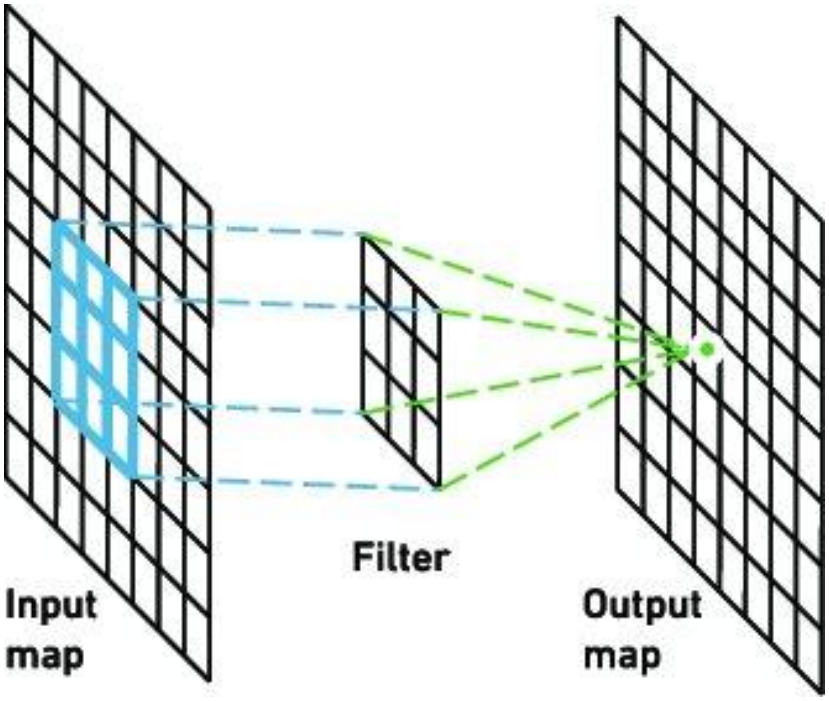

Fig 6 Outline of the convolutional layer [Yakura et al., 2018]

Convolutional layers are essential to **SqueezeNet's** ability to extract features from incoming data. These layers discover important patterns like edges, textures, and forms by performing convolution operations on the source image using filters, often known as kernels. A 7x7 convolutional layer with 96 filters and a stride of 2 is the first component of the framework. This layer processes the input image (224 × 224 × 3) to extract low-level features while downsampling the spatial dimensions, setting the stage for subsequent operations. Additionally, SqueezeNet employs 1×1 convolutions extensively within its Fire Modules to reduce parameters and computational complexity [Iandola et al., 2016]. These smaller filters are particularly efficient, capturing features without unnecessary redundancy. The architecture also integrates 3×3 convolutions in the expand layers of the Fire Modules to ensure spatial relationships are preserved. By combining these convolutional operations strategically, SqueezeNet achieves a balance between computational efficiency and feature richness, contributing to its compact yet powerful design.

So The output feature map dimensions (H_{out}, W_{out}) are computed as:

$$H_{out} = \left\lfloor \frac{H_{in} + 2P - K}{S} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2P - K}{S} + 1 \right\rfloor$$

Where

- H_{in} , W_{in} : Input height and width
- K: Kernel Size
- S : Stride
- P : Padding

The **Fire Module** is the key building block of the SqueezeNet architecture, designed to reduce the model size while maintaining feature extraction efficiency. Each Fire Module consists of two main components: the squeeze layer and the expand layer.

i. Squeeze Layer: This layer applies 1×1 convolutions to the input feature maps, significantly reducing the number of input channels and computational requirements. It acts as a bottleneck, compressing the input data to a smaller representation before expansion [Atam, 2021].
ii. Expand Layer: Following the squeeze layer, the expand layer operates on the compressed feature map using a combination of 1×1 and 3×3 convolutions. The 1×1 convolutions refine the fine-grained details, while the 3×3 convolutions capture broader spatial relationships. The outputs of these convolutions are concatenated depth-wise to form the final output of the Fire Module [Atam, 2021].

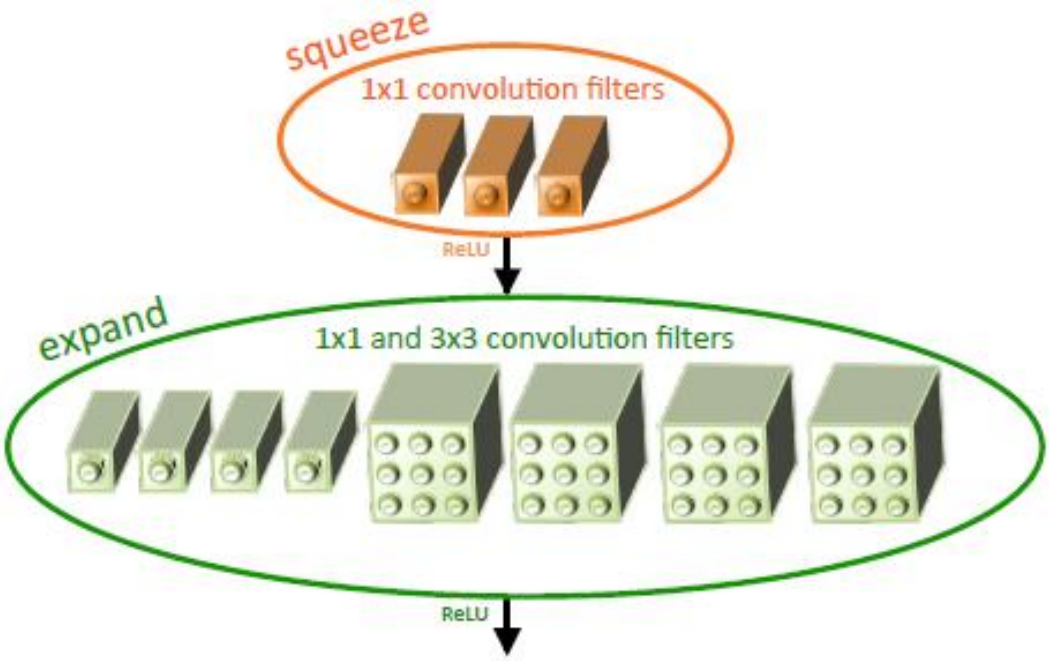

Fig 7 Fire Module with hyperparameter

S 1x1 = 3, E 1x1 =4 and E 3x3 = 4

The squeeze layer applies 1 x 1 convolutions to reduce input channels:

$$C_{squeeze} = F_{squeeze} \times C_{input}$$

Where $F_{squeeze}$ is the ratio of squeezed channels.

The expand layer generates output channels using both 1×1 and 3×3 convolutions:

$$C_{expand} = F_{1\times1} \times C_{3\times3}$$

So the total parameter count for the fire module is:

$$Parameters = \left(C_{input} \times C_{squeeze} \times 1^2\right) + \left(C_{squeeze} \times (C_{1\times1} + C_{3\times3}) \times 3^2\right)$$

Global Average Pooling (GAP) is a pooling technique that preserves feature maps' depth-wise information while reducing their spatial dimensions. In contrast to conventional pooling

methods (such as average or max pooling), which work on fixed-size windows, GAP calculates the average of every feature map over its whole spatial area. It takes the values of each feature map and averages them to create a 1 × 1 × C vector from a H × W × C feature map (height, width, and channels) [Atam, 2021].

$$GAP\ (x) = \frac{1}{H\ \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} x(i,j)$$

Where $x(i,j)$ are the special values of the feature map.

Advantages of this layer is;

- GAP does not introduce additional parameters, helping prevent overfitting.
- Captures the overall presence of features rather than local details.
- Each channel in the output vector corresponds to a class, facilitating simple and effective classification.

The **softmax layer** is the final layer in many classification models, including SqueezeNet. It converts raw scores (logits) from the preceding layer into probabilities for each class, ensuring the sum of probabilities equals 1. This normalization helps interpret the output in terms of confidence for each class. Mathematical formulation of this layer is like that;

Given logic z_1, z_2, …., z_n the softmax function calculates probabilities as;

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_i}}$$

Where $P(y_i)$ is the probability of the i-th class.

Key characteristic for softmax layer is belongs to;

- Converts scores into a probability distribution across all classes.
- Assigns higher probabilities to dominant classes while suppressing less likely ones.
- Allows predictions to be interpreted directly as confidence levels.

In SqueezeNet, the GAP layer feeds directly into the softmax layer, enabling the model to output probabilities for each class without requiring fully connected layers. This design enhances efficiency and reduces overfitting, contributing to SqueezeNet's compact and high-performance architecture.

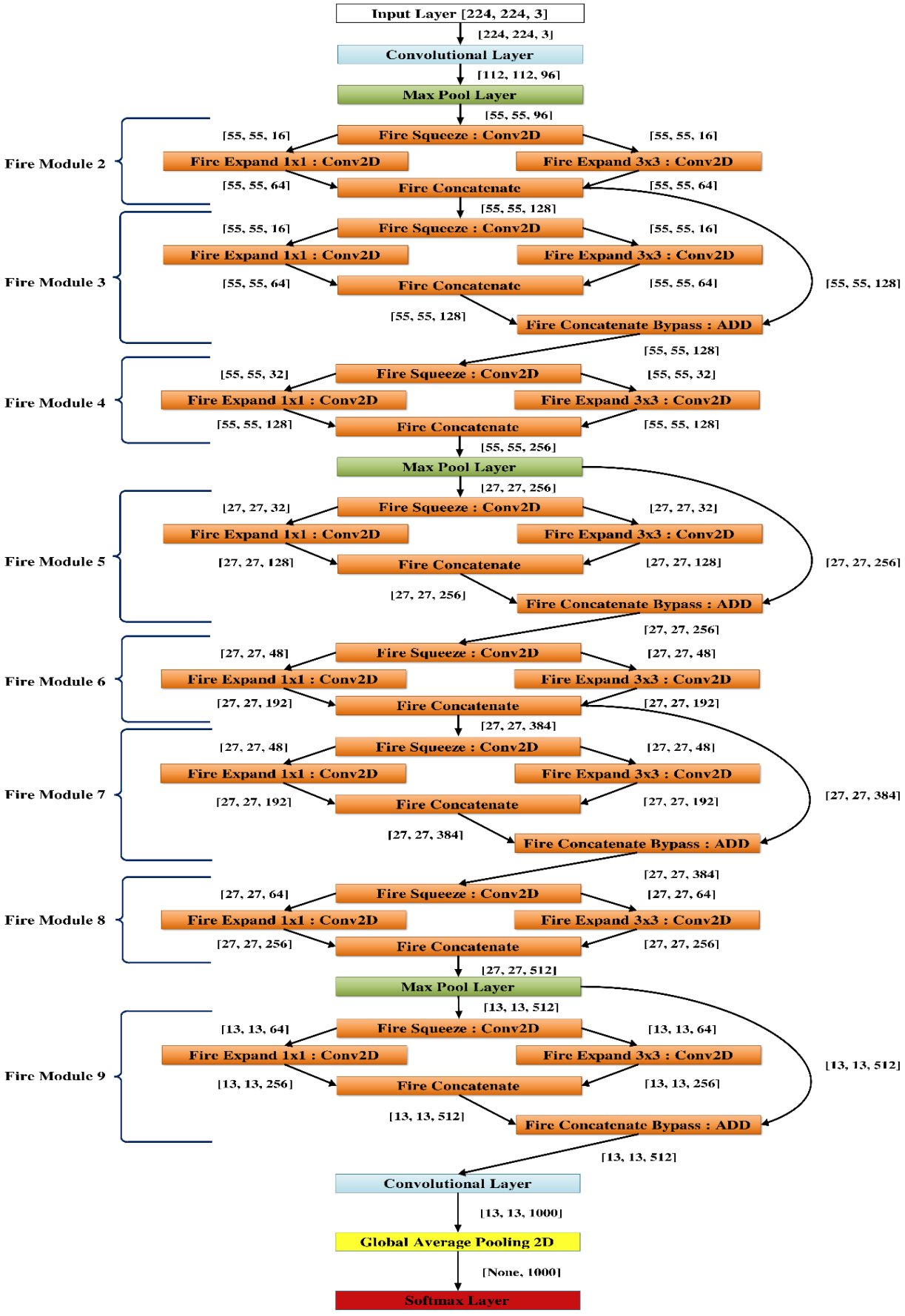

Fig 8 SqueezeNet Architecture [Iandola et al., 2016]

Here is a table summarizing the hyper parameters of the SqueezeNet architecture:

Table 4 Hyperparameter for SqueezeNet Framework

Convolutional	Hyperparameter	Value/Details
Convolutional Layer	Kernel Size	7 × 7 (initial), 1 × 1 (squeeze), 3 × 3 (expand layers)
	Stride	2 (initial layer), 1 (other layers)
	Number of Filters	96 (initial layers), Various in fire Modules
Fire Module	Squeeze Kernel Size	1 × 1
	Squeeze Filters	Proportional to expand filters, reduces input channels
	Expand Kernel Sizes	1 × 1 and 3 × 3
	Expand Filters	Balanced between 1 × 1 and 3 × 3 convolutional
Pooling Layer	Pooling Type	Max pooling (intermediate), Global Average Pooling (final layer)
	Pooling Kernel Size	3 × 3 for max polling
	Pooling Stride	2 (max pooling)
General	Input Image Size	224 × 224 × 3
	Number of Classes	Varies based on dataset (e.g., 1000 for ImageNet)
	Activation Function	ReLU
Optimization	Learning Rate	Typically 10^{-3} or adaptive
	Batch Size	Varies depending on hardware

3.4.2 Mish Activation Function

The Mish activation function, introduced in 2019 by Diganta Misra, is a smooth and non-monotonic activation function designed to enhance the learning efficiency of neural networks. It is a blend of mathematical elegance and practical efficiency, providing better gradient flow and feature representation compared to traditional activation functions like ReLU and Swish [Misra, 2019]. Below is a detailed breakdown of Mish.

The Mish activation function is mathematically expressed as:

$$Mish(x) = x.\tanh\left(ln(1 + e^{x})\right)$$

Where

- *x:* Input to the function
- $ln(1 + e^{x})$: The softplus function, providing a smooth approximation of ReLU.
- tanh : The hyperbolic tangent function, introducing bounded non-linearity.

The Mish activation function is built upon the following essential components [Misra, 2019].

i. **Softplus Function:**
   - The term $ln(1 + e^x)$ ensures smoothness and differentiability.
   - For large x, $ln(1 + e^x) \rightarrow x$ making Mish behave linearly.
   - For small x, $ln(1 + e^x) \rightarrow e^x$ ensuring graceful handling of negative inputs.

ii. **Hyperbolic Tangnt (tanh):**
   - $tanh(y) = \frac{e^y - e^{-y}}{e^y - e^{-y}}$
   - It maps the softplus output to a range of $(-1,1)$, introducing non-linearity to Mish.

iii. **Linear Scaling (x):**
   - The input x is scaled linearly, maintaining the magnitude of the input while applying the non-linearity.

So the behaviour of the mish activation function:

i. **For Positive Inputs (x>0):**
   - The softplus function approximates x and tanh(x)≈1, making Mish output values close to x.
   - Mish behaves similarly to ReLU for positive inputs but with a smoother transition.

ii. **For Negative Inputs (x<0):**
   - Mish suppresses negative values but does not truncate them to zero, unlike ReLU.
   - This retention of negative information helps in better feature representation and gradient flow.

iii. **At Zero (x=0):**
   - $Mish(0) = 0.\tanh\left(ln(1 + e^0)\right)$ ensuring smooth behavior at the origin.

The Mish activation function demonstrates several significant mathematical properties, enhancing its utility in deep learning tasks. The first derivative of Mish is given by:

$$Mish'(x) = tanh\left(ln(1 + e^x)\right) + x.sech^2\left(ln(1 + e^x)\right) \times \frac{e^x}{1 + e^x}$$

Here:

- $sech(y) = \frac{1}{cosh(y)}$ is the hyperbolic secant function.
- This ensures a stable gradient flow, addressing issues like vanishing and exploding gradients.

The second derivative is also smooth, enabling stable learning across varying activation magnitudes, which further facilitates efficient optimization. The range of Mish is:

$$Output \in (-c, \infty)$$

where $c \approx 0.31$. This range allows Mish to handle negative inputs gracefully by compressing them without truncating to zero, unlike ReLU.

3.4.3 PhytoNet (Mish-Optimized SqueezeNet) Framework

The PhytoNet Framework combines the lightweight and efficient architecture of SqueezeNet with the powerful capabilities of the Mish activation function. SqueezeNet is renowned for its compact design and reduced computational complexity, making it an ideal choice for deployment in resource-constrained environments. By replacing the ReLU activation function with Mish, this framework enhances its ability to extract complex features and achieve improved classification performance.

In the first input preprocessing stage of the framework, image data is normalised to fall within the interval [1,1]. By ensuring that the input data is centred around zero, this normalisation aids in accelerating convergence during training. The model's first layer uses a convolution operation with a stride of two and a big 7x7 kernel. This layer extracts important low-level features like edges and textures while shrinking the input image's spatial dimensions. The output of this layer is passed through the Mish activation function, which is defined as

$$Mish(x) = x.\tanh\left(ln(1 + e^{x})\right)$$

Mish's smooth and non-monotonic properties allow it to retain small negative values, contributing to richer feature learning and better gradient flow compared to traditional activations like ReLU. At the heart of the architecture lies the Fire Module, a highly efficient building block designed to reduce computation while maintaining high accuracy. The squeeze phase and the expand phase are the two primary stages that make up each Fire Module. A 1×1 convolution minimises computing expenses in the squeeze step by reducing the number of input channels. The Mish activation function, which improves the module's capacity to recognise complex patterns, comes next. In the expand stage, two parallel branches process the squeezed feature maps. The first branch employs 1×1 convolutions to capture fine-grained details, while the second branch uses 3×3 convolutions to capture spatial context. The outputs of these branches are concatenated to form a rich, multi-scale feature representation. This design allows the Fire Module to balance efficiency and representational power effectively.

The Fire Modules are arranged sequentially, with intermediate max-pooling layers to progressively reduce spatial dimensions and focus on the most relevant features. As the network deepens, the feature maps become increasingly abstract and representative of the input data. After processing through multiple Fire Modules, the framework applies dropout to the final module's output to prevent overfitting. A 1x1 convolutional layer is then applied to the output, reducing the number of feature maps to correspond with the number of output classes. The spatial average of each feature map is then calculated via global average pooling, producing a condensed and comprehensible vector representation of the input. Lastly, class probabilities for the input image are produced via a softmax activation function.

The integration of the Mish activation function into SqueezeNet significantly enhances the model's performance by improving the flow of gradients during training. Unlike ReLU, which clips negative values to zero, Mish retains small negative values, enabling the network to learn more nuanced features. Its smooth and continuous nature also facilitates better optimization, making the framework highly effective for various machine learning tasks.

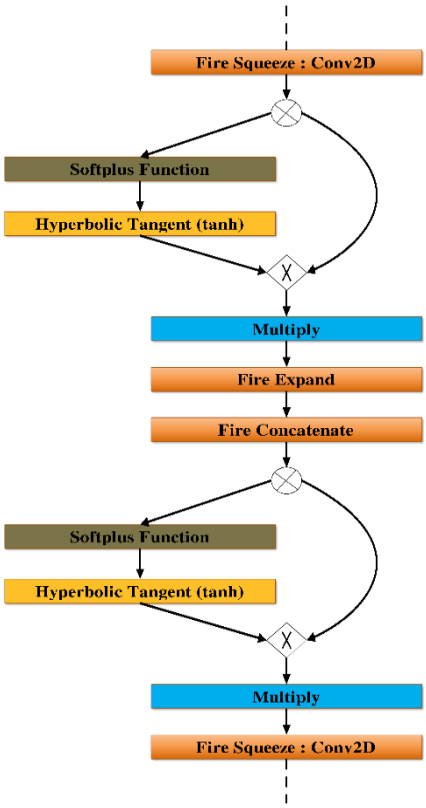

Fig 9 Mish Activation Function in Squeeznet Architecture

The PhytoNet Framework is particularly well-suited for scenarios where efficiency and accuracy are paramount. Its compact design and robust learning capabilities make it ideal for applications in edge computing, real-time systems, and mobile platforms. Additionally, its ability to achieve superior performance with fewer parameters makes it a valuable tool for tasks such as object recognition, medical imaging, and other resource-intensive domains. By merging the efficiency of SqueezeNet with the advanced feature extraction capabilities of Mish, this framework represents a significant advancement in the field of deep learning.

Algorithm Steps : PhytoNet

Step 1: Initialization

Input Parameters:

- Input shape: $\boldsymbol{I_{shape}} = (\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{C})$, where H, W and C are the height, width, and channels of the input image.
- Number of classes: $\boldsymbol{n_{classes}}$

Define Mish Activation Function:

$$\boldsymbol{Mish(x)} = \boldsymbol{x}.\tanh\left(\ln\left(1 + e^{x}\right)\right)$$

Step 2: Pre-processing

Normalize the input data I:

$$\boldsymbol{I_{rescaled}} = \frac{\boldsymbol{I}}{\boldsymbol{127.5}} - \boldsymbol{1}$$

Step 3: Initial Convolution and Pooling

- Apply a convolution layer:

$$\boldsymbol{Conv1} = \boldsymbol{W_1} \times \boldsymbol{I_{rescaled}} + \boldsymbol{b_1}$$

$$W_1 \in \mathbf{R}^{7\times7\times C\times96}$$

$$Stride = 2$$

- Activate using Mish:

$$A_{conv1} = Mish(Conv1)$$

- Apply max pooling:

$$P_{conv1} = \max_{3\times3}(A_{conv1}),$$

$$Stride = 2$$

Step 4: Fire Module Operations

Each Fire Module contains:

Squeeze Step:

- Perform 1 × 1 convolutional:

$$Squeeze = W_{sq} \times P_{input} \times b_{sq}$$

$$W_{sq} \in R^{1\times1\times F_{input}\times S_1}$$

- Apply Mish activation

$$A_{squeeze} = Mish(Squeeze)$$

Expand Step:

- Perform 1×1 convolution

$$Expand_{1\times1} = W_{e1} \times A_{squeeze} + b_{e1}$$

$$W_{e1} \in \mathbf{R}^{1\times1\times S_1\times E_1}$$

- Perform 3×3 convolution

$$Expand_{3\times3} = W_{e3} \times A_{squeeze} + b_{e3}$$

$$W_{e1} \in \mathbf{R}^{3\times3\times S_1\times E_3}$$

$$Padding = Same$$

Concatenate:

- Combine the feature maps

$$Output_{fire} = Concatenate([Expand_{1\times1}, Expand_{3\times3}])$$

- Apply Mish Activation:

$$A_{fire} = Mish(Output_{fire})$$

Step 5: Fire Module Sequence

Apply a series of Fire Modules with pooling as follows

- Fire2→Fire3→Fire4→MaxPooling
- Fire5→Fire6→Fire7→Fire8→Maxpooling
- Fire9

Step 6: Regularization

Apply dropout

$$D_{fire9} = Dropout(A_{fire9}, rate = 0.5)$$

Step 7: Final Convolutional and Global Pooling

- Perform 1x1 convolution for classification

$$Conv_{10} = W_{10} \times D_{fire9} + b_{10}$$

$$W_{10} \in \mathbf{R}^{1\times1\times F_{input}\times n_{classes}}$$

- Apply global average pooling

$$GAP = \frac{1}{H \times W}\sum_{i=1}^{H}\sum_{j=1}^{W} Conv10_{ij}$$

Step 8: Output Activation

- Apply softmax for multi-class classification

$$Output = Softmax(GAP)$$

$$Softmax(z_i) = \frac{e^{z_j}}{\sum_{j=1}^{n_{classes}} e^{z_j}}$$

Note:

- W and b: Weights and biases of the convolutional layers.
- ×: Convolution operation.
- Mish activation and Softmax are applied at appropriate steps to ensure smooth gradients and proper classification.
- Fire Modules significantly reduce parameters while maintaining accuracy, leveraging
- $Squeeze$ and $Expand$ pathways.

4. Result and Discussion

4.1 Evaluation Matrices

Evaluation metrics are numerical measurements that are used to evaluate how well deep learning and machine learning models perform. Following evaluation matrices is used for the quantitative measure of the presented model.

A performance indicator called **accuracy** assesses how successfully a model forecasts the desired results for a certain dataset. It can be described as the percentage of correctly classified occurrences compared to all instances.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ Predictions} \times 100$$

The degree to which a model's predictions match the actual labels is measured by its **loss**. In this paper, the loss function, SparseCategoricalCrossentropy(), assesses the distinction between the actual class labels and the expected probability distribution for multi-class classification situations when the labels are given as integers.

For a given sample i, the sparse categorical cross-entropy loss is computed as:

$$L_i = -log(p_{i,y_i})$$

Where

- y_i: The true class index for sample i.
- p_{i,y_i}: The predicted probability for the true class y_i, as output by the model's softmax layer.

An assessment metric called **precision** is used to gauge how well a model predicts the future. It measures the proportion of expected positive cases that turn out to be true positives.

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positivs\ (TP) + False\ Positives(FP)}$$

Where

- True Positives (TP): The number of correctly predicted positive instances.
- False Positives (FP): The number of instances incorrectly predicted as positive but are actually negative.

An evaluation parameter called **recall**, sometimes referred to as sensitivity or true positive rate, gauges how successfully a model detects each and every real positive case in a dataset. The percentage of real positives that the model accurately predicts as positive is its main focus.

$$Recall = \frac{True\ Positives\ (TP)}{True\ Positivs\ (TP) + False\ Nigatives(FN)}$$

Where

- False Negatives (FN): The number of actual positive instances that the model failed to predict as positive.

In classification problems, the **F1 Score** is a statistic that strikes a balance between recall and precision. It is a single figure that represents the disparity across recall and precision, calculated as the harmonic mean of the two. When there are imbalanced classes in the dataset or when both false positives and false negatives are significant, the F1 score is especially helpful.

$$F1\ Score = 2\ \times \frac{Precision\ \times Recall}{Precision + Recall}$$

4.2 Experiment Results

The experimental results are showed in Figures 9 and 10, showcasing accuracy and loss graphs for the SqueezeNet and PhytoNet (Mish-Optimized SqueezeNet) architectures using Adam optimizers. The training accuracy is shown by the blue line in these graphs, while the reliability of validation during training is shown by the orange line. The effectiveness of the model during the validation and training phases is visually represented by the accuracy and loss graph, which highlights how well the Adam optimiser improves learning. Initially, the training process starts with relatively high loss and low accuracy. Over successive epochs, the model demonstrates significant improvement, with training loss steadily decreasing and accuracy increasing The Adam optimiser, which is well-known for its effective gradient descent and adjustable learning rate, is essential for speeding up convergence because it dynamically modifies the step size. Validation metrics generally follow a similar trend, reflecting the model's ability to generalize. However, as training progresses, validation loss may begin to fluctuate, and validation accuracy

could exhibit slight dips, signalling the onset of overfitting. This is a common challenge when the model becomes overly specialized to the training data. To address this, techniques like early stopping are employed, preserving the model's best performance and maintaining a balance between learning and generalization.

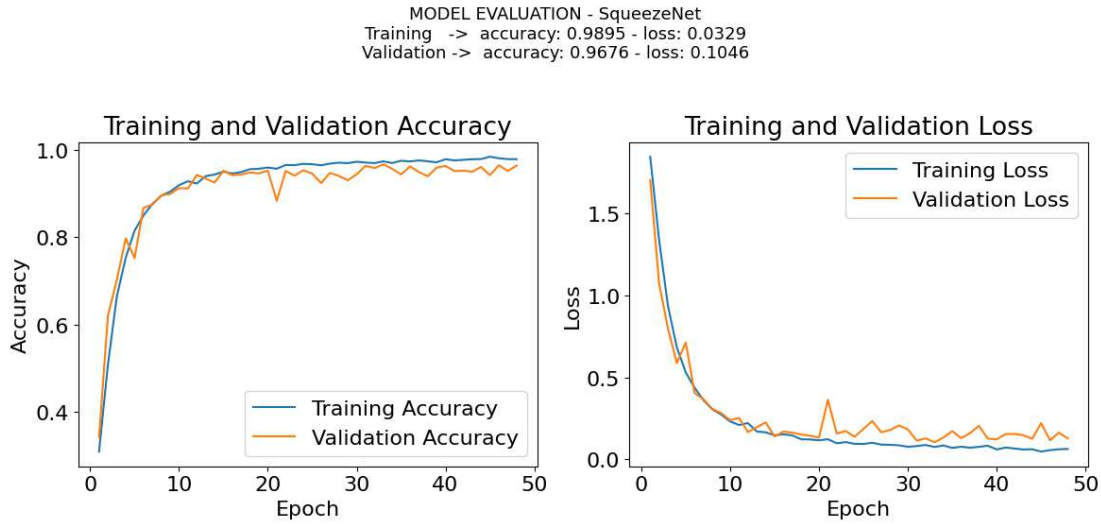

Fig 10 SqueezeNet Model Training and Validation Graph for Accuracy

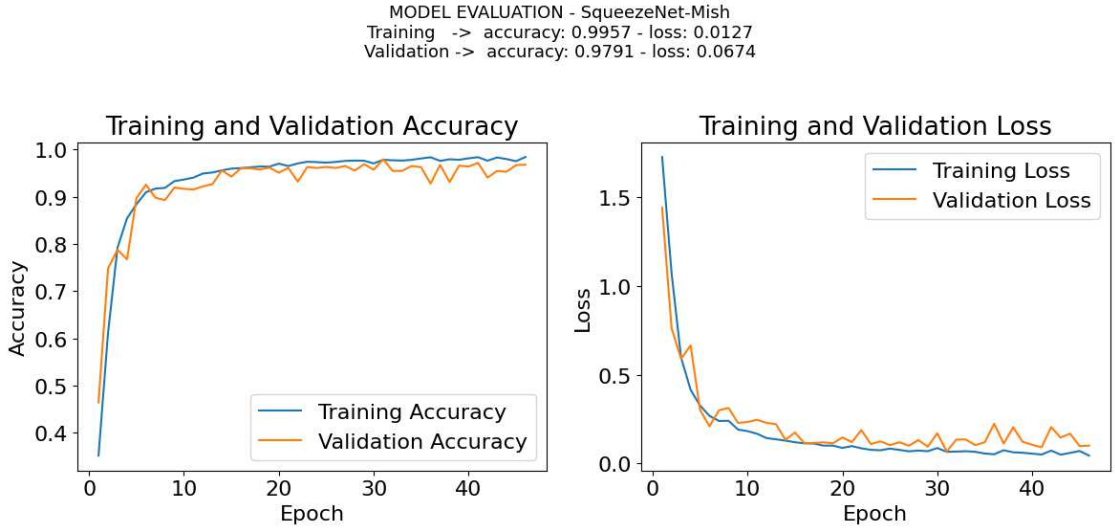

Fig 11 SqueezeNet Model Training and Validation Graph for Loss

Table 5 Model Accuracy and Graph

	Accuracy		Loss	
	Training	**Validation**	**Training**	**Validation**
SqueezeNet	0.9895	0.9676	0.0329	0.1046
PhytoNet (Mish-Optimized SqueezeNet)	0.9957	0.9791	0.0127	0.0674

The table compares the performance of two models—SqueezeNet and Mish Optimized SqueezeNet—across training and validation datasets, focusing on accuracy and loss. SqueezeNet demonstrated high performance but a slight decline in generalisation on unknown data, achieving accuracy of 0.9895 on the data used for training set and 0.9676 on the validation set. However, after using the Mish activation function, Mish Optimized SqueezeNet demonstrated an enhancement with an accuracy in training of 0.9957 and an accuracy for validation of 0.9791, indicating improved learning and generalisation. In terms of loss, SqueezeNet showed a validation loss of 0.1046, indicating considerable error on validation data but little mistake on training data, with a training loss of 0.0329. With a validation loss of 0.0674 and a training loss of 0.0127, Mish Optimized SqueezeNet demonstrated even greater performance on unseen data and superior fitting of models on the training set. Overall, Mish Optimized SqueezeNet outperforms SqueezeNet in both accuracy and loss, particularly in terms of generalization, owing to the advantages of the Mish activation function.

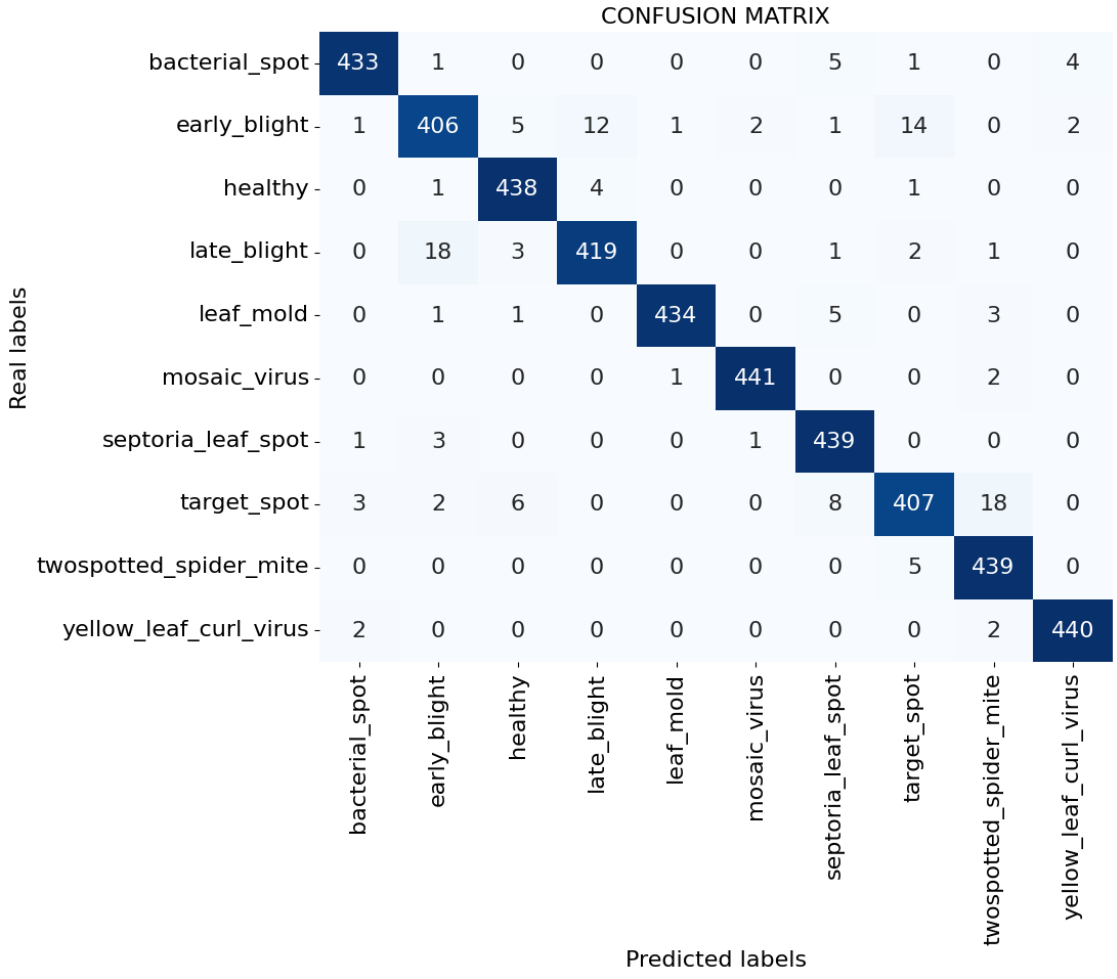

Fig 12 Confusion Matrix for SqueezeNet Framework

A confusion matrix evaluates the classification performance of SqueezeNet and PhytoNet (Mish-Optimized SqueezeNet) architectures across ten classes, including bacterial_spot, early_blight, healthy, and others. It compares true labels with predicted labels to visualize correct predictions (diagonal values) and misclassifications (off-diagonal values). The dataset is prepared using TensorFlow, with images and labels extracted as NumPy arrays. Predictions are generated using the trained model, with the highest probability class selected for each

image. The matrix highlights classification accuracy per class, with diagonal values indicating correct predictions and off-diagonal values showing errors. For instance, confusion between early_blight and late_blight suggests visual similarity. Misclassification patterns inform strategies like data augmentation or model fine-tuning to improve performance.

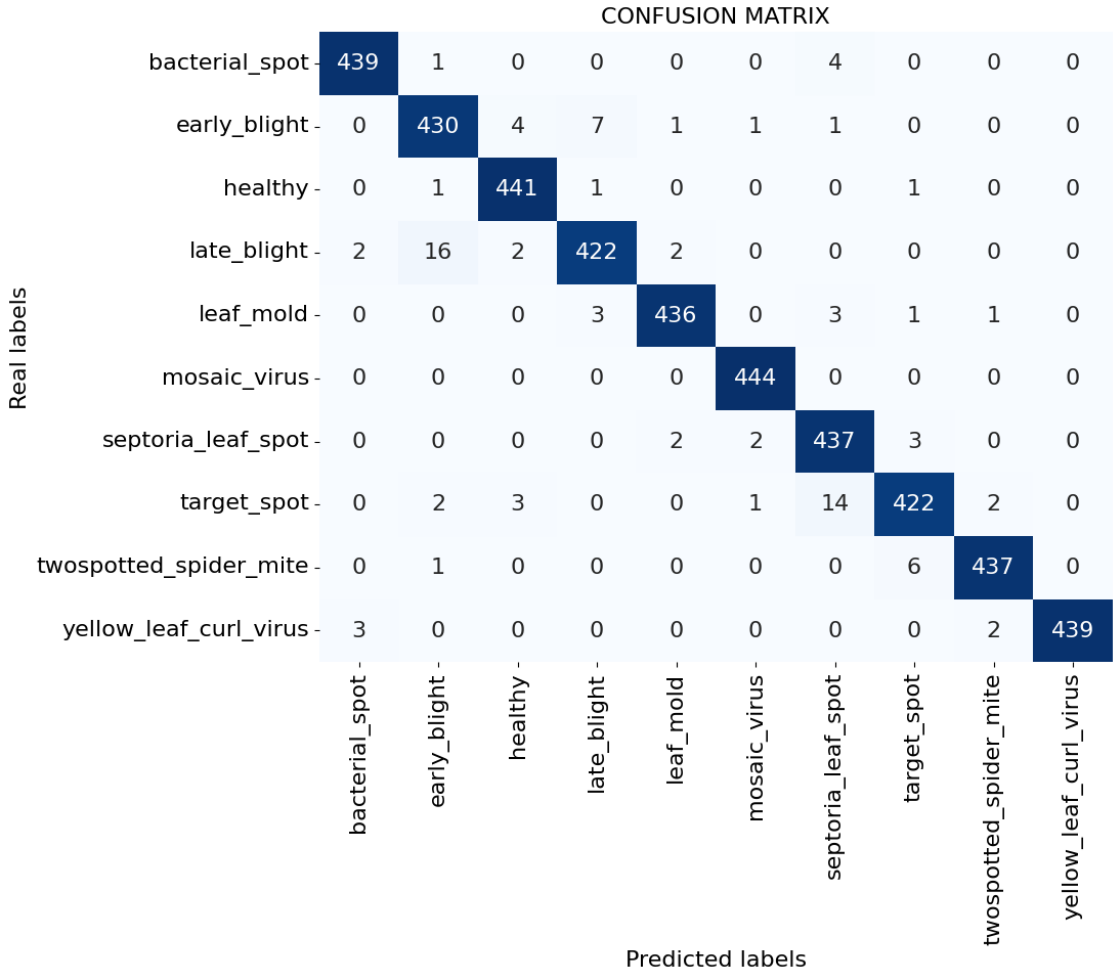

Fig 13 Confusion Matrix for PhytoNet Framework

Table 6 Class wise Evaluation Result

	Squeezenet Architecture			Mish Optimized Squeezenet Architecture		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
bacterial_spot	0.9841	0.9752	0.9796	0.9887	0.9887	0.9887
early_blight	0.9398	0.9144	0.9269	0.9534	0.9685	0.9609
healthy	0.9669	0.9865	0.9766	0.9800	0.9932	0.9866
late_blight	0.9632	0.9437	0.9534	0.9746	0.9505	0.9624
leaf_mold	0.9954	0.9775	0.9864	0.9887	0.9820	0.9853
mosaic_virus	0.9932	0.9932	0.9932	0.9911	1.0000	0.9955
septoria_leaf_spot	0.9564	0.9887	0.9723	0.9521	0.9842	0.9679
target_spot	0.9465	0.9167	0.9314	0.9746	0.9505	0.9624
twospotted_spider_mite	0.9441	0.9887	0.9659	0.9887	0.9842	0.9865
yellow_leaf_curl_virus	0.9865	0.9910	0.9888	1.0000	0.9887	0.9943

The table compares the performance of two models, SqueezeNet and Mish Optimized SqueezeNet, using metrics such as precision, recall, and F1-score across tomato plant disease classes. For most classes, the Mish Optimized SqueezeNet outperforms the standard SqueezeNet in all metrics, indicating improved performance. For instance, in the "bacterial_spot" class, Mish Optimized SqueezeNet shows higher precision, recall, and F1-score, reflecting better identification and classification accuracy. Similarly, the "early_blight" class demonstrates an increase in recall and F1-score, suggesting that Mish Optimized SqueezeNet is better at identifying instances of this disease. In the "healthy" class, both models perform well, but Mish Optimized SqueezeNet still has a slight edge in precision and recall, contributing to a higher F1-score. For "late_blight" and "leaf_mold," Mish Optimized SqueezeNet also shows improved metrics, signifying more accurate detection and fewer false positives. In "mosaic_virus," the Mish Optimized SqueezeNet achieves perfect recall, indicating its capability to identify every instance of this disease without missing any, while maintaining a strong F1-score. Other classes, such as "septoria_leaf_spot," "target_spot," and "twospotted_spider_mite," also benefit from Mish Optimized SqueezeNet's enhanced performance, with the model showing better precision, recall, and F1-score in most cases. Finally, in the "yellow_leaf_curl_virus" class, Mish Optimized SqueezeNet achieves perfect precision and recall, leading to an optimal F1-score. Overall, Mish Optimized SqueezeNet demonstrates superior performance across all plant disease classes, showcasing the effectiveness of the Mish activation function in improving classification accuracy.

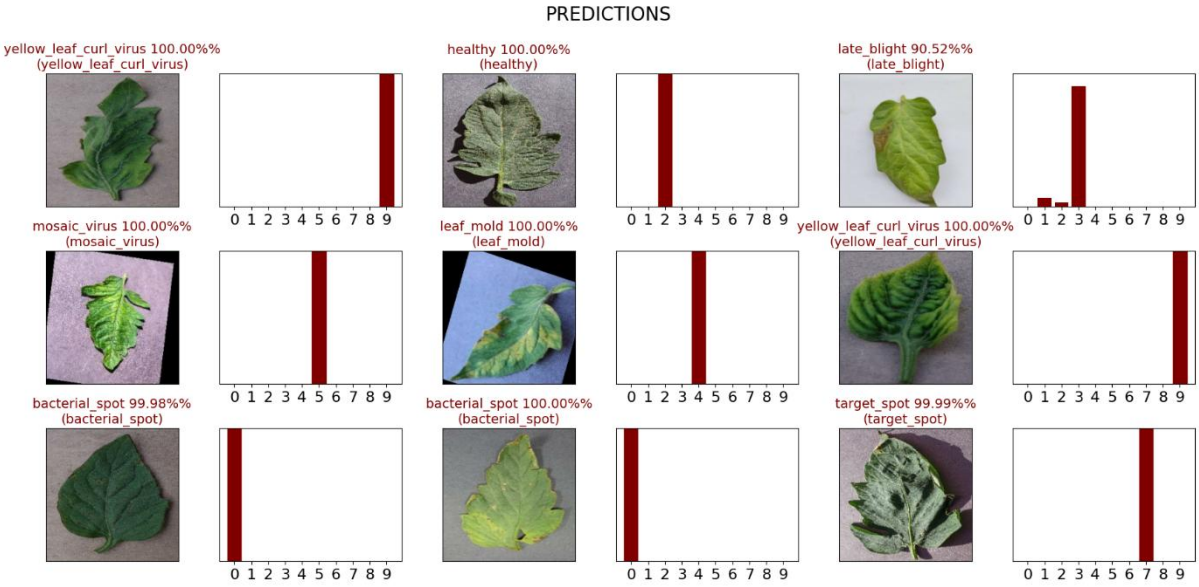

Fig 14 Prediction Visualization for SqueezeNet Framework

In the context of prediction using the PhytoNet Framework for enhanced tomato leaf disease classification via deep learning, a subset of nine test images is selected for performance evaluation. These images are cast into the tf.uint8 data type to ensure compatibility with the model's prediction mechanism. Corresponding ground truth labels and prediction results are extracted for these images. Both the SqueezeNet and PhytoNet (Mish-Optimized SqueezeNet) models, which have been trained on a dataset of tomato leaf diseases, generate predictions for each image. A direct comparison is made between the true labels and the predicted classes. To facilitate the analysis, the plot_predictions function is employed to visualize the images along with their predicted and true labels. This visualization enables a precise assessment of the

model's prediction accuracy by comparing the predicted categories with the ground truth for each selected image. The step is vital for evaluating the model's generalization capability, ensuring its robustness and reliability in real-world disease detection scenarios.

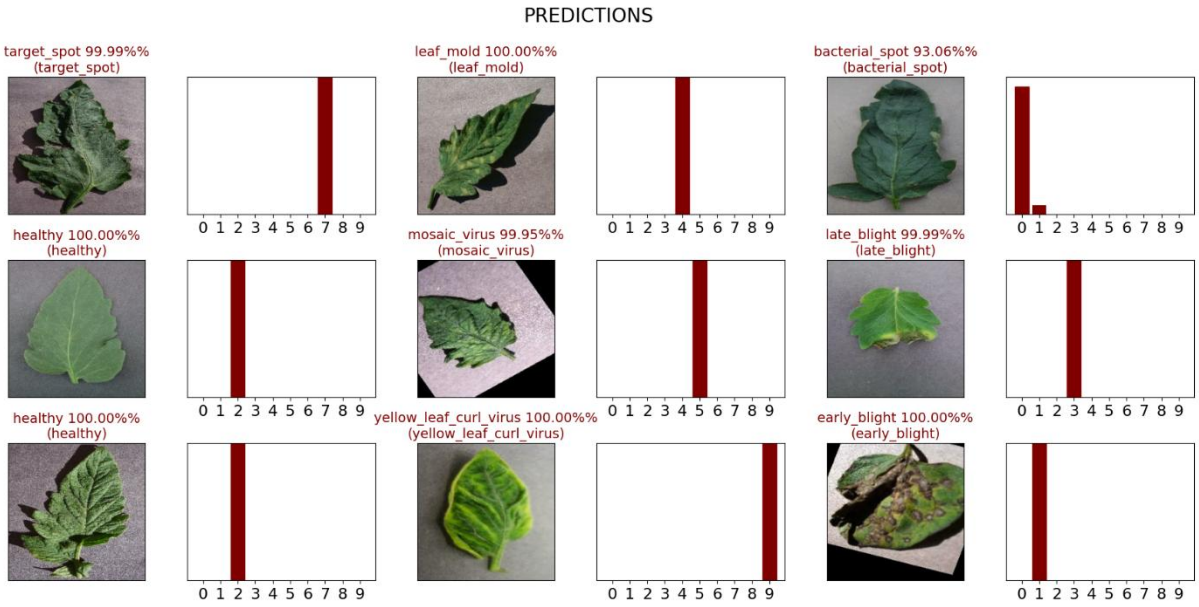

Fig 15 Prediction Visualization for PhytoNet Framework

4.3 Comparative Analysis

The comparative analysis of different models reveals significant technical advancements in deep learning architectures and optimization techniques, emphasizing their methodologies and performance metrics. The Transfer Learning with C-GAN approach achieves an accuracy of 0.9711 by integrating pre-trained networks for feature extraction with Conditional Generative Adversarial Networks (C-GAN) for data augmentation. C-GAN introduces conditional constraints during data generation, enabling the creation of diverse, realistic samples and improving the model's robustness in handling imbalanced datasets. Similarly, the Improved AlexNet Model achieves a higher accuracy of 0.9872 by refining AlexNet's architecture through optimized kernel configurations, dropout rates, and layer adjustments, thereby enhancing feature extraction while mitigating overfitting. In contrast, the HOG + LBP + SVM method, with an accuracy of 0.9100, relies on traditional feature extraction. Here, HOG captures gradient orientations, LBP encodes local texture features, and SVM serves as the classifier, but its performance is constrained by the lack of hierarchical feature representation. The Two-Stage Transfer Learning Approach excels with an accuracy of 0.9923, utilizing a two-step process: the initial stage captures generalized feature representations, while the subsequent stage fine-tunes the model for domain-specific learning. This sequential refinement ensures higher adaptability and precision. A hybrid approach combining GJ GSO (Global Jaya Guided Search Optimization) and a modified AlexNet, achieving 0.9100 accuracy, focuses on optimization algorithms for parameter tuning. However, its dependence on a relatively older architecture limits its effectiveness compared to more modern techniques. The CNN-Based U Network Model, reaching 0.9915 accuracy, adopts an encoder-decoder framework, where convolutional layers in the encoder extract spatial features, and the decoder uses up-sampling and skip connections to achieve pixel-level precision, making it ideal for segmentation tasks.

Table 7 Comparative Analysis for other related work

Ref.	Model Used	Accuracy
Islam et al., 2022	Transfer Learning with C-GAN	0.9711
Qiu et al., 2024	Improved AlexNet Model	0.9872
Qiu et al., 2024	HOG + LBP + SVM	0.9100
Sanida et al., 2023	Two-Stage Transfer Learning Approach	0.9923
Badiger and Mathew, 2023	GJ GSO + DbneAlxNet	0.9100
Perveen et al., 2023	CNN-Based U Network Model	0.9915
Kaur et al., 2022	Modified Mask R-CNN	0.9800
Mashamba et al., 2024	VGG16	0.9328
Cheemaladinne and K, 2024	VARMAx-CNN-GAN Integration	0.9200
This Study	SqueezeNet	0.9895
	PhytoNet (Mish-Optimized SqueezeNet) Framework	**0.9957**

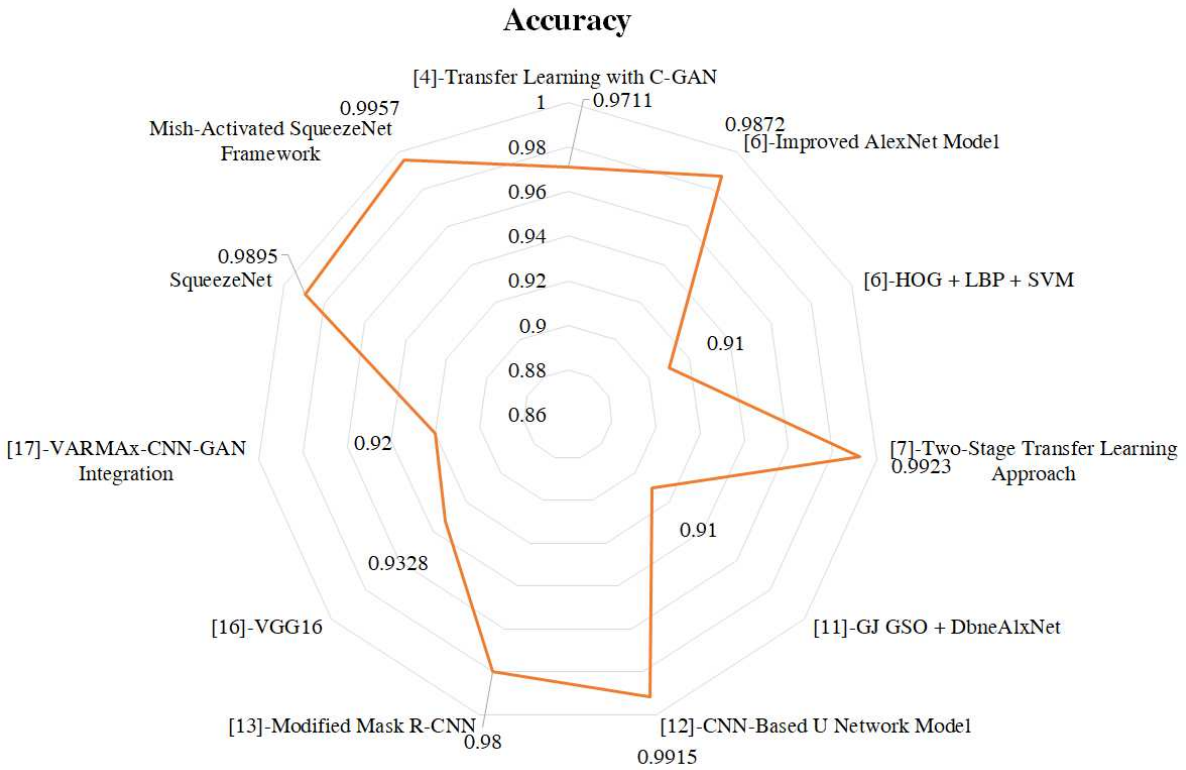

Fig 15 Comparative Analysis for other related work

The Modified Mask R-CNN achieves 0.9800 accuracy by enhancing Mask R-CNN's region proposal networks and integrating segmentation masks alongside object detection. These modifications improve its ability to handle complex object localization and segmentation. The VGG16 Model, known for its simplicity and depth, achieves an accuracy of 0.9328 but is limited by its computational cost and lack of advanced mechanisms like attention layers or residual connections, which restrict scalability. The VARMAx-CNN-GAN Integration records an accuracy of 0.9200, combining time-series analysis with VARMAx, feature extraction with CNNs, and synthetic data generation using GANs, showcasing a fusion of statistical and deep learning techniques. In this study, two variations of SqueezeNet are evaluated. The standard

SqueezeNet model achieves an accuracy of 0.9895, employing a compact architecture with fire modules that include squeeze and expand layers to reduce parameters while retaining high performance. An enhanced version, the Mish-Optimized SqueezeNet Framework, achieves the highest accuracy of 0.9957 by replacing the ReLU activation with the Mish activation function. The smoother gradients and non-monotonic behavior of Mish improve optimization and mitigate gradient vanishing, enabling better feature representation and generalization. This makes the PhytoNet (Mish-Optimized SqueezeNet) a highly efficient and lightweight framework, demonstrating the importance of architectural innovations and advanced activation functions in achieving state-of-the-art results.

4.4 Advantages and Limitation

The PhytoNet Framework offers several advantages, particularly in terms of performance, efficiency, and optimization. Below are the key advantages of this framework:

- **Enhanced Model Accuracy:** The primary advantage of using the Mish activation function in the SqueezeNet architecture is the improvement in accuracy. Mish provides smoother gradients and non-monotonic behavior, which helps the model avoid issues like vanishing gradients and facilitates better optimization. This leads to improved model convergence and higher overall accuracy in predicting tomato leaf diseases, achieving better results compared to traditional activation functions like ReLU.
- **Compact Model with High Efficiency:** SqueezeNet is designed to be a lightweight model, reducing the number of parameters while maintaining high performance. It uses "fire modules," which consist of a squeeze layer (with fewer filters) followed by an expand layer (with more filters). This design reduces computational costs, memory usage, and inference time, making it ideal for deployment on edge devices or environments with limited resources. By combining Mish with SqueezeNet, the framework strikes a balance between high accuracy and computational efficiency.
- **Faster Inference Time:** The SqueezeNet architecture is known for its fast inference time due to its small model size. This is particularly important in real-time applications like agricultural disease prediction, where quick decision-making is crucial. The Mish activation function improves the training process, leading to faster convergence, while still preserving the model's ability to generalize effectively.
- **Better Generalization:** The use of Mish contributes to better generalization due to its smoother and more stable gradient properties compared to other activation functions like ReLU. This results in improved performance when the model is applied to unseen data or real-world scenarios, which is especially important when predicting diseases on new tomato leaf samples that were not present in the training set.
- **Reduced Overfitting:** Mish activation helps in reducing overfitting by maintaining smooth gradient flows throughout the training process. This allows the model to learn more robust features, which is crucial for the accurate classification of tomato leaf diseases, where subtle differences between disease types may exist.
- **Compatibility with Transfer Learning:** The SqueezeNet architecture can easily be used with transfer learning. Pre-trained models on large datasets can be fine-tuned for tomato leaf disease classification, which speeds up training time and enhances performance with a smaller dataset. The PhytoNet framework benefits from this

transfer learning approach, making it adaptable to various plant disease prediction tasks beyond tomato leaves.

- **Improved Robustness to Noisy Data:** Agricultural datasets, especially those involving images like tomato leaf disease datasets, often contain noise due to variations in lighting, angle, or image quality. The Mish activation function improves the robustness of the model to such noisy data, making it more reliable in real-world scenarios.
- **Reduced Computational Complexity:** The SqueezeNet architecture reduces the number of parameters significantly, making it computationally less expensive compared to larger models like VGG16 or ResNet. When paired with the Mish activation function, it not only becomes more efficient in terms of computational resources but also maintains the accuracy needed for effective disease prediction.

The PhytoNet Framework for Tomato Leaf Disease Prediction Using Deep Learning has several **limitations**, despite its advantages. One of the primary challenges is the reliance on large, diverse datasets for optimal performance. Deep learning models like this one require extensive and varied data to generalize well, and limited or unrepresentative datasets can lead to overfitting, reducing the model's ability to perform accurately on unseen data. Additionally, while SqueezeNet is efficient, its simplicity can limit its ability to capture complex patterns in datasets with subtle differences between disease types, especially in cases where diseases exhibit similar visual features. This makes it less effective for datasets with high intra-class variation or complex patterns.

Moreover, the introduction of the Mish activation function, although beneficial in terms of model accuracy and training, could slightly increase computational complexity, which might affect inference speed, particularly on resource-constrained devices. This could pose a challenge in real-time applications where quick predictions are required. Furthermore, the quality of image pre-processing is crucial to the model's performance. If pre-processing steps like resizing, normalization, or noise removal are inadequate, the model's predictions can be significantly impacted. The PhytoNet Framework may also struggle with handling extreme variability in tomato leaf appearance caused by environmental conditions, plant age, or disease progression stages, leading to misclassifications in real-world scenarios.

Another limitation is the model’s interpretability. Like many deep learning models, the PhytoNet framework operates as a "black-box," making it difficult to explain how decisions are made. This lack of transparency can reduce trust, particularly among end-users such as farmers or agricultural experts who may require explanations for the model's predictions. The framework also faces challenges in dealing with the emergence of new diseases or changing conditions, as it would need to be retrained regularly with updated data to remain accurate. Additionally, it is limited to image-based analysis and does not integrate other valuable data types, such as environmental factors or multispectral imagery, which could provide a more comprehensive understanding of plant health.

5. Conclusion

The proposed PhytoNet Framework provides a significant advancement in tomato leaf disease prediction, addressing the critical balance between accuracy and computational efficiency. By integrating the Mish activation function into the SqueezeNet architecture, the model capitalizes on enhanced gradient flow and improved non-linear feature learning, enabling superior classification performance compared to conventional lightweight architectures. This approach

demonstrates a classification accuracy .9957, showcasing its potential for practical applications in precision agriculture. One of the key strengths of the framework lies in its lightweight design, which ensures low computational overhead. This characteristic makes the model highly suitable for deployment in resource-constrained environments, such as edge devices and mobile platforms, where traditional deep learning models face limitations. The integration of robust preprocessing techniques, including data augmentation and normalization, further enhances the model's generalizability and robustness, enabling consistent performance across diverse datasets. The experimental results validate the efficacy of the PhytoNet, highlighting its ability to outperform standard SqueezeNet and other lightweight architectures in terms of accuracy and efficiency. Moreover, the framework's adaptability to real-world conditions, combined with its low inference time, establishes it as a practical solution for real-time monitoring of tomato crops. This research underscores the transformative potential of lightweight, activation-optimized deep learning frameworks in addressing critical challenges in agriculture. The successful application of the PhytoNet to tomato leaf disease prediction opens avenues for extending this approach to other crops and plant disease datasets. Future work could focus on incorporating multimodal data, such as environmental and soil conditions, to further improve prediction accuracy. Overall, this study contributes to the development of scalable, efficient, and accessible solutions for disease management in precision farming, promoting sustainable agricultural practices worldwide.

Author Contribution

Deependra Rastogi: Conceptualization, Methodology, Investigation, Data Curation, Formal Analysis, Writing – Original Draft Preparation.

Prashant Johri: Supervision, Validation, Formal Analysis, Writing – Review & Editing.

Varun Tiwari: Investigation, Resources, Data Curation, Visualization.

Anand Singh: Validation, Formal Analysis, Visualization.

Sumit Singh Dhanda: Investigation, Data Curation, Writing – Review & Editing.

Atul Kumar Singh: Formal Analysis, Resources.

Gaurav Kumar: Writing – Review & Editing.

Data Availability

The dataset used in this study, "Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network," is publicly available and can be accessed through the Mendeley Data repository at https://doi.org/10.17632/tywbtsjrjv.1.

Conflicts of Interest

No conflicts of interest are disclosed by the author or author(s).

Funding Statement

No funding involves in this study.

Consent to Publish Declaration: Not applicable

Ethics and Consent to Participate Declarations: Not applicable

References

[1]. Cardellicchio, A., Solimani, F., Dimauro, G., Petrozza, A., Summerer, S., Cellini, F., Renò, V., 2023. Detection of tomato plant phenotyping traits using YOLOv5-based single stage detectors. Computers and Electronics in Agriculture 207, 107757. https://doi.org/10.1016/j.compag.2023.107757

[2]. Cheng, H.-H., Dai, Y.-L., Lin, Y., Hsu, H.-C., Lin, C.-P., Huang, J.-H., Chen, S.-F., Kuo, Y.-F., 2022. Identifying tomato leaf diseases under real field conditions using convolutional neural networks and a chatbot. Computers and Electronics in Agriculture 202, 107365. https://doi.org/10.1016/j.compag.2022.107365

[3]. Agarwal, M., Singh, A., Arjaria, S., Sinha, A., Gupta, S., 2020. ToLeD: Tomato Leaf Disease Detection using Convolution Neural Network. Procedia Computer Science 167, 293–301. https://doi.org/10.1016/j.procs.2020.03.225

[4]. Abbas, A., Jain, S., Gour, M., Vankudothu, S., 2021. Tomato plant disease detection using transfer learning with C-GAN synthetic images. Computers and Electronics in Agriculture 187, 106279. https://doi.org/10.1016/j.compag.2021.106279

[5]. Islam, M.P., Hatou, K., Aihara, T., Seno, S., Kirino, S., Okamoto, S., 2022. Performance prediction of tomato leaf disease by a series of parallel convolutional neural networks. Smart Agricultural Technology 2, 100054. https://doi.org/10.1016/j.atech.2022.100054

[6]. Qiu, J., Lu, X., Wang, X., Chen, C., Chen, Y., Yang, Y., 2024. Research on image recognition of tomato leaf diseases based on improved AlexNet model. Heliyon 10, e33555. https://doi.org/10.1016/j.heliyon.2024.e33555

[7]. Sanida, T., Sideris, A., Sanida, M.V., Dasygenis, M., 2023. Tomato leaf disease identification via two–stage transfer learning approach. Smart Agricultural Technology 5, 100275. https://doi.org/10.1016/j.atech.2023.100275

[8]. Vini, S.L., Rathika, P., 2024. TrioConvTomatoNet: A robust CNN architecture for fast and accurate tomato leaf disease classification for real time application. Scientia Horticulturae 330, 113079. https://doi.org/10.1016/j.scienta.2024.113079

[9]. Ye, Y., Zhou, H., Yu, H., Hu, H., Zhang, G., Hu, J., He, T., 2024. Application of Tswin-F network based on multi-scale feature fusion in tomato leaf lesion recognition. Pattern Recognition 110775. https://doi.org/10.1016/j.patcog.2024.110775

[10]. Nag, A., Chanda, P.R., Nandi, S., 2023. Mobile app-based tomato disease identification with fine-tuned convolutional neural networks. Computers & Electrical Engineering 112, 108995. https://doi.org/10.1016/j.compeleceng.2023.108995

[11]. Badiger, M., Mathew, J.A., 2023. Tomato plant leaf disease segmentation and multiclass disease detection using hybrid optimization enabled deep learning. Journal of Biotechnology 374, 101–113. https://doi.org/10.1016/j.jbiotec.2023.07.011

[12]. Perveen, K., Debnath, S., Pandey, B., Chand, S.P., Bukhari, N.A., Bhowmick, P., Alshaikh, N.A., Arzoo, S., Batool, S., 2023. Deep learning-based multiscale CNN-based U network model for leaf disease diagnosis and segmentation of lesions in tomato. Physiological and Molecular Plant Pathology 128, 102148. https://doi.org/10.1016/j.pmpp.2023.102148

[13]. Kaur, P., Harnal, S., Gautam, V., Singh, M.P., Singh, S.P., 2022. An approach for characterization of infected area in tomato leaf disease based on deep learning and object detection technique. Engineering Applications of Artificial Intelligence 115, 105210. https://doi.org/10.1016/j.engappai.2022.105210

[14]. Li, M., Zhou, G., Chen, A., Li, L., Hu, Y., 2023. Identification of tomato leaf diseases based on LMBRNet. Engineering Applications of Artificial Intelligence 123, 106195. https://doi.org/10.1016/j.engappai.2023.106195

[15]. Zhang, D., Huang, Y., Wu, C., Ma, M., 2023. Detecting tomato disease types and degrees using multi-branch and destruction learning. Computers and Electronics in Agriculture 213, 108244. https://doi.org/10.1016/j.compag.2023.108244 (Zhang et al., 2023)

[16]. Mashamba, M.M., Telukdarie, A., Munien, I., Onkonkwo, U., Vermeulen, A., 2024. Detection of bacterial spot disease on tomato leaves using a Convolutional Neural Network (CNN). Procedia Computer Science 237, 602–609. https://doi.org/10.1016/j.procs.2024.05.145

[17]. Cheemaladinne, V., K, S.R., 2024. Tomato leaf disease detection and management using VARMAx-CNN-GAN integration. Journal of King Saud University. Science/Mağallaẗ Ǧāmiʻaẗ Al-malik Saʻūd. al-ʻUlūm 36, 103340. https://doi.org/10.1016/j.jksus.2024.103340

[18]. Kaushik, H., Khanna, A., Singh, D., Kaur, M., Lee, H.-N., 2023. TomFusioNet: A tomato crop analysis framework for mobile applications using the multi-objective optimization based late fusion of deep models and background elimination. Applied Soft Computing 133, 109898. https://doi.org/10.1016/j.asoc.2022.109898

[19]. Wang, X., Yang, W., Yang, Y., Huang, M., Zhu, Q., 2023. Identification of tomato bacterial wilt severity based on hyperspectral imaging technology and spectrum Transformer network. Ecological Informatics 78, 102353. https://doi.org/10.1016/j.ecoinf.2023.102353

[20]. Baser, P., Saini, J.R., Kotecha, K., 2023. TomConv: An Improved CNN Model for Diagnosis of Diseases in Tomato Plant Leaves. Procedia Computer Science 218, 1825–1833. https://doi.org/10.1016/j.procs.2023.01.160

[21]. Kodali, R.K., Gudala, P., 2021. Tomato Plant Leaf Disease Detection using CNN. https://doi.org/10.1109/r10-htc53172.2021.9641655 (Kodali and Gudala, 2021)

[22]. Kumar, N.S., Sony, J., Premkumar, A., R, M., Nair, J.J., 2024. Transfer Learning-based Object Detection Models for Improved Diagnosis of Tomato Leaf Disease. Procedia Computer Science 235, 3025–3034. https://doi.org/10.1016/j.procs.2024.04.286

[23]. Al-Shamasneh, A.R., Ibrahim, R.W., 2024. Classification of tomato leaf images for detection of plant disease using conformable polynomials image features. MethodsX 13, 102844. https://doi.org/10.1016/j.mex.2024.102844

[24]. Georgantopoulos, P.S., Papadimitriou, D., Constantinopoulos, C., Manios, T., Daliakopoulos, I.N., Kosmopoulos, D., 2023. A Multispectral Dataset for the Detection of Tuta Absoluta and Leveillula Taurica in Tomato Plants. Smart Agricultural Technology 4, 100146. https://doi.org/10.1016/j.atech.2022.100146

[25]. Astani, M., Hasheminejad, M., Vaghefi, M., 2022. A diverse ensemble classifier for tomato disease recognition. Computers and Electronics in Agriculture 198, 107054. https://doi.org/10.1016/j.compag.2022.107054

[26]. Hettiarachchi, D., Fernando, V., Kegalle, H., Halloluwa, T., 2022. UrbanAgro: Utilizing advanced deep learning to support Sri Lankan urban farmers to detect and control common diseases in tomato plants, in: Elsevier eBooks. pp. 263–282. https://doi.org/10.1016/b978-0-323-90550-3.00010-2

[27]. J, A.P., Gopal, G., 2019. Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network. NA. https://doi.org/10.17632/tywbtsjrjv.1 (J and Gopal, 2019)

[28]. Polly, R., Devi, E.A., 2024. "Semantic segmentation for plant leaf disease classification and damage detection: A deep learning approach." Smart Agricultural Technology 9, 100526. https://doi.org/10.1016/j.atech.2024.100526

[29]. Ullah, N., Khan, J.A., Almakdi, S., Alshehri, M.S., Qathrady, M.A., El-Rashidy, N., El-Sappagh, S., Ali, F., 2023b. An effective approach for plant leaf diseases classification based on a novel DeepPlantNet deep learning model. Frontiers in Plant Science 14. https://doi.org/10.3389/fpls.2023.1212747

[30]. Tsang, S.-H., 2021. Review: SqueezeNet (Image Classification) - Towards Data Science. Medium.

[31]. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1602.07360

[32]. Yakura, H., Shinozaki, S., Nishimura, R., Oyama, Y., Sakuma, J., 2018. Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism. NA 127–134. https://doi.org/10.1145/3176258.3176335

[33]. Atam, A., 2021. SqueezeNet Model [with Architecture] [WWW Document]. OpenGenus IQ: Learn Algorithms, DL, System Design. URL https://iq.opengenus.org/squeezenet-model/

[34]. Misra, D., 2019. Mish: A Self Regularized Non-Monotonic Neural Activation Function. arXiv (Cornell University).

[35]. Agarwal, M. (2023). Tomato Plant Disease Classification Using Local Patterns. Advances in Electrical and Electronic Engineering, 21(4), 360-368. doi:10.15598/aeee.v21i4.5036

[36]. Russel, N.S., Selvaraj, A., 2022. Leaf species and disease classification using multiscale parallel deep CNN architecture. Neural Computing and Applications 34, 19217–19237. https://doi.org/10.1007/s00521-022-07521-w

[37]. Zhang, Y., Chen, M., 2022. An IoT-enabled energy-efficient approach for the detection of leaf curl disease in tomato crops. Wireless Networks 29, 321–329. https://doi.org/10.1007/s11276-022-03071-0

[38]. Bhagat, M., Kumar, D., 2023. Efficient feature selection using BoWs and SURF method for leaf disease identification. Multimedia Tools and Applications 82, 28187–28211. https://doi.org/10.1007/s11042-023-14625-5

[39]. Zhou, C., Zhou, S., Xing, J., Song, J., 2021. Tomato leaf disease identification by restructured deep residual dense network. IEEE Access 9, 28822–28831. https://doi.org/10.1109/access.2021.3058947

[40]. Alzahrani, M.S., Alsaade, F.W., 2023. Transform and deep learning algorithms for the early detection and recognition of tomato leaf disease. Agronomy 13, 1184. https://doi.org/10.3390/agronomy13051184

[41]. Nagaraju, M., Chawla, P., Upadhyay, S., Tiwari, R., 2021. Convolution network model based leaf disease detection using augmentation techniques. Expert Systems 39. https://doi.org/10.1111/exsy.12885

[42]. Kaur, P., Harnal, S., Tiwari, R., Upadhyay, S., Bhatia, S., Mashat, A., Alabdali, A.M., 2022. Recognition of leaf disease using hybrid convolutional neural network by applying feature reduction. Sensors 22, 575. https://doi.org/10.3390/s22020575

[43]. Mishra, A.M., Harnal, S., Gautam, V., Tiwari, R., Upadhyay, S., 2022. Weed density estimation in soya bean crop using deep convolutional neural networks in smart agriculture. Journal of Plant Diseases and Protection 129, 593–604. https://doi.org/10.1007/s41348-022-00595-7

[44]. Sun, H., Fu, R., Wang, X. et al. Efficient deep learning-based tomato leaf disease detection through global and local feature fusion. BMC Plant Biol 25, 311 (2025). https://doi.org/10.1186/s12870-025-06247-w

[45]. Ghosh, H., Rahat, I.S., Emon, M.M.R. et al. Advanced neural network architectures for tomato leaf disease diagnosis in precision agriculture. Discov Sustain 6, 312 (2025). https://doi.org/10.1007/s43621-025-01149-1

[46]. Zarboubi, M., Bellout, A., Chabaa, S., & Dliou, A. (2025). CustomBottleneck-VGGNet: Advanced tomato leaf disease identification for sustainable agriculture. Computers and Electronics in Agriculture, 232, 110066. https://doi.org/10.1016/j.compag.2025.110066

[47]. Das, A., Pathan, F., Jim, J. R., Ouishy, M. R., Kabir, M. M., & Mridha, M. F. (2025). XLTLDisNet: A Novel and Lightweight Approach to Identify Tomato Leaf Diseases with Transparency. Heliyon, 11(4), e42575. https://doi.org/10.1016/j.heliyon.2025.e42575