scientific reports

Check for updates

OPEN

Bayesian optimized multimodal deep hybrid learning approach for tomato leaf disease classification

Bodruzzaman Khan[1]✉, Subhabrata Das[2], Nafis Shahid Fahim[1], Santanu Banerjee[3], Salma Khan[4], Mohammad Khalid Al-Sadoon[5], Hamad S. Al-Otaibi[5] & Abu Reza Md. Towfiqul Islam[6,7]✉

Manual identification of tomato leaf diseases is a time-consuming and laborious process that may lead to inaccurate results without professional assistance. Therefore, an automated, early, and precise leaf disease recognition system is essential for farmers to ensure the quality and quantity of tomato production by providing timely interventions to mitigate disease spread. In this study, we have proposed seven robust Bayesian optimized deep hybrid learning models leveraging the synergy between deep learning and machine learning for the automated classification of ten types of tomato leaves (nine diseased and one healthy). We customized the popular Convolutional Neural Network (CNN) algorithm for automatic feature extraction due to its ability to capture spatial hierarchies of features directly from raw data and classical machine learning techniques [Random Forest (RF), XGBoost, GaussianNB (GNB), Support Vector Machines (SVM), Multinomial Logistic Regression (MLR), K-Nearest Neighbor (KNN)], and stacking for classifications. Additionally, the study incorporated a Boruta feature filtering layer to capture the statistically significant features. The standard, research-oriented PlantVillage dataset was used for the performance testing, which facilitates benchmarking against prior research and enables meaningful comparisons of classification performance across different approaches. We utilized a variety of statistical classification metrics to demonstrate the robustness of our models. Using the CNN-Stacking model, this study achieved the highest classification performance among the seven hybrid models. On an unseen dataset, this model achieved average precision, recall, f1-score, mcc, and accuracy values of 98.527%, 98.533%, 98.527%, 98.525%, and 98.268%, respectively. Our study requires only 0.174 s of testing time to correctly identify noisy, blurry, and transformed images. This indicates our approach's time efficiency and generalizability in images captured under challenging lighting conditions and with complex backgrounds. Based on the comparative analysis, our approach is superior and computationally inexpensive compared to the existing studies. This work will aid in developing a smartphone app to offer farmers a real-time disease diagnosis tool and management strategies.

Keywords Tomato leaf disease, Bayesian optimization, Hybrid learning, Machine learning, CNN, Deep learning, Boruta, Tree-structured Parzen Estimator

Tomato (*Solanum lycopersicum*) is a commercial and nutritious food crop that plays a significant role in the agricultural economy[1]. It accounted for 16% of total vegetable production in 2019[2]. The latest statistics reveal that the world's tomato production is approximately 180 million metric tons, exporting USD 8.81 billion[3]. However,

[1]Department of Agricultural Construction and Environmental Engineering, Sylhet Agricultural University, Sylhet 3100, Bangladesh. [2]Langmuir Center of Colloids and Interfaces, Columbia University in the City of New York, New York, USA. [3]Department of Agriculture, Chhatrapati Shahu Ji Maharaj University, Kanpur, Uttar Pradesh 208012, India. [4]Institute of Leather Engineering and Technology, University of Dhaka, Dhaka 1209, Bangladesh. [5]Department of Zoology, College of Science, King Saud University, PO Box 2455, Riyadh 11451, Saudi Arabia. [6]Department of Disaster Management, Begum Rokeya University, Rangpur 5400, Bangladesh. [7]Department of Development Studies, Daffodil International University, Dhaka 1216, Bangladesh. ✉email: bodruzzamankhan.sau@gmail.com; towfiq_dm@brur.ac.bd

due to vulnerability to some infectious and non-infectious factors, tomato production is declining each year by 8–10%[4]. The effect of these factors may range from a slight impact to a much more significant impact—to a greater extent, wiping out the entire crop. Moreover, plant diseases are responsible for both the quantity and quality of production, which ultimately affect human health and cause economic damage and food insecurity[5,6]. Yellow leaf curl virus, bacterial spot, mosaic virus, septoria leaf spot, leaf mold, late blight, powdery mildew, and early blight are the most common tomato leaf diseases[7]. These leaf diseases are highly damaging to tomato crop yields, leading to significant investment in pest control measures by contributing to a large portion of the total economic impact[8]. Therefore, tomato leaf disease inspection and management are crucial components in agricultural practice aimed at mitigating crop yield losses from diseases.

The indiscriminate and excessive use of pesticides in agriculture, as a means to mitigate the effects of diseases and pests, poses significant economic, environmental, and health risks. While the intention may be to protect crop yields, this practice often leads to several detrimental outcomes. So, precise and early identification of diseases is necessary to minimize the risk of extensive use of pesticides, reduce crop loss, and helping farmers efficiently identify diseased plants and take timely corrective actions to ensure both the quality and quantity of crops. These diseases could become more severe and impede crop production if not identified and monitored on time[9]. Thus, early and accurate identification and classification of tomato leaf disease can aid in preventing disease spread among other plants, reduce yield losses, and ensure optimal production.

Identification of plant diseases is a crucial aspect of plant pathology, and it is often accomplished through a meticulous examination of the morphological symptoms of plant leaf diseases with the naked eye, relying on the observer's ability to discern visual cues and anomalies in the plant's foliage. Morphological symptoms include visible changes in color, texture, shape, and overall appearance of leaves, as well as the presence of lesions, spots, or other distinctive patterns. Molecular, serological, and microbiological diagnostic techniques are the other standard methods in the realm of plant pathology for diagnosing plant diseases[10]. Implementing the visual inspection method demands incessant expert monitoring, rendering it primarily expensive and time-consuming. This is particularly evident in the case of large agricultural farms and remote locations where accessibility poses a significant challenge[11]. On large farms, the scale of operations necessitates a substantial investment in human resources, amplifying both the financial and temporal aspects[12]. Therefore, researchers have increasingly turned to automated systems for crop disease recognition as a proactive measure to overcome challenges associated with manual inspection[13,14]. Recently, machine learning (ML) has drawn widespread attention for crop disease identification and classification, similar to its remarkable success in other diverse domains[14–16]. One notable application of ML in agriculture involves the accurate categorization of different fruits and vegetables within large datasets[17], as well as to predict large-scale crop yields[18]. With the rapid advancement of artificial intelligence (AI), image processing and databases with different images help in the easy diagnosis and classification of plant diseases[19,20]. Researchers and farmers utilize digital images of plant leaves, stems, or fruits, captured under various conditions, to train machine learning models. Support Vector Machines (SVM)[21], Logistic Regression[22], Naïve Bayes Bayes[23], K-nearest Neighbors (KNN), adaptive boosting[24], Random Forest (RF)[25], Artificial Neural Networks (ANN)[26,27] are the commonly practiced conventional computer vision-based techniques used for automated plant disease identification and categorization. Besides ML models, deep learning (DL) methods, i.e., Convolutional Neural Network (CNN)[28,29], Recurrent neural networks (RNNs)[30], long short-term memory (LSTM), etc., are widely employed for plant disease categorization and identification[31] due to their remarkable performance in handling sequential data and capturing long-range dependencies, which are crucial aspects in the analysis of plant health. Nowadays, transfer learning approaches like AlexNet[32], ResNet[33], and VGG[34] are also popular for fruit counting, crop disease recognition, and classification, which drastically decrease the need for massive computational resources and the model construction period[35]. DL methods are also applied in multiscale agricultural sensing to provide a resource for the global agricultural community and advance precision agriculture[36]. Among the wide range of DL methods, the deep Convolutional Neural Network (CNN) is most widely used for crop disease detection, classification, and segmentation[37–39]. The inherent ability of CNNs to automatically learn hierarchical representations from input data makes them particularly effective in handling complex visual information, such as images of crops affected by diseases. CNNs excel at capturing intricate patterns and features within images, enabling them to discern subtle differences indicative of various crop ailments. This capability is crucial for accurate disease detection and classification, contributing to the development of more resilient agricultural practices. Additionally, CNNs facilitate image segmentation, allowing for the precise delineation of affected regions within a crop, aiding in targeted interventions for disease management.

Despite diseases leaving some distinct marks on tomato plant leaves, distinguishing between them proves challenging due to the subtle variations among the affected foliage. Various diseases can manifest with distinct symptoms, such as discoloration, lesions, or wilting, yet the similarities in appearance often confound even seasoned horticulturists. The intricacies lie in the nuanced patterns and shades that diseases present, making it a formidable task to pinpoint the exact ailment affecting the tomato plants. This complexity underscores the importance of precise diagnosis and vigilant monitoring to implement targeted interventions and safeguard the overall health of the tomato crop. Existing literature reveals the presence of well-established models designed for the early detection and classification of tomato leaf diseases. These models represent significant strides in agricultural technology, offering valuable tools for farmers and researchers. However, despite their contributions, certain limitations hinder their effectiveness. One notable concern is the restricted number of disease classes that these models can accurately identify, limiting their applicability to a broader range of potential issues. Additionally, there are challenges in achieving optimal generalizations and optimizations, which are crucial for enhancing the models' adaptability to diverse environmental conditions and disease variations. Another critical aspect is the impact of computational complexity on the overall robustness of these models. As agricultural systems evolve, addressing these limitations becomes imperative to ensure that these technological solutions remain effective and contribute to sustainable and resilient farming practices. There is a potential research gap in the literature to

explore novel feature extraction techniques, multimodal data fusion, handling disease variability, etc., in tomato leaf disease classification. In such a case, enhancing ML models' transparency, interpretability, and generalization capabilities across different tomato leaf diseases is essential. Additionally, DL models are frequently used in all the tasks related to image processing and disease classification, which overlooks the potential of conventional ML algorithms. A large amount of data and a considerable amount of time are required for the DL model to perform effectively, which makes it more expensive and time-consuming. In contrast, ML can perform very well with small training data, making machine learning classification faster[40]. In light of the aforementioned challenges, this study presents Bayesian optimized deep hybrid learning models that are designed to be robust, lightweight, and computationally efficient for the classification of tomato leaf diseases. The proposed novel approach involves optimizing seven hybrid models, wherein a CNN serves as a feature extractor and machine learning techniques such as RF, XGBoost, Gaussian Naïve Bayes (GaussianNB), SVM, Multinomial Logistic Regression, KNN, and a Stacking ensemble method are employed for the classification. Our hybrid models are designed to make the best use of the distinct features of both DL and ML models so that the models become well generalized and have high classification capabilities. The performance of these hybrid models has been extensively validated through comprehensive comparisons with state-of-the-art approaches. In addition, the study integrated a Boruta feature selection layer to discern and filter out the most relevant features essential for disease classification.

The utilization of Bayesian optimization in automated leaf disease detection, particularly in the context of tomato leaves, has been extensively explored in recent literature. This approach has been applied across various crops, including rice, maize, and plants in general. For instance, Wang et al.[41] employed an attention-based neural network combined with Bayesian optimization for rice disease detection and classification. Similarly, da Rocha et al.[42] utilized convolutional neural networks (CNNs) alongside hyperparameter optimization techniques for maize leaf disease classification. Additionally, Restrepo-Arias et al.[43] proposed a plant disease detection strategy integrating image texture analysis with Bayesian optimization using small neural networks. Building upon these studies, Seno et al.[44] introduced a novel Bayesian convolutional neural network approach for diagnosing tomato leaf diseases, contributing to the expanding repertoire of techniques in automated agricultural disease detection. These references collectively underscore the established presence of Bayesian optimization methods within the field, indicating its integration as a standard practice. The novelty of the study is that applying Bayesian optimization contributes to a more data-efficient and cost-effective agricultural process. While Bayesian optimization may have been utilized in automated leaf disease detection previously, our study distinguishes itself by providing comprehensive insights into its implementation, including parameter tuning, optimization strategies, and performance evaluation specific to tomato leaf disease detection. Thus, our research contributes significantly to the advancement of automated tomato leaf disease detection by innovatively harnessing Bayesian optimization techniques tailored to this domain's distinct requirements. Furthermore, by combining Bayesian optimization and DL-ML for classifying tomato leaf diseases, we aim to enhance various facets of agricultural operations, ranging from crop management to disease detection and yield prediction. It shows how the integration of cutting-edge technology can be used in traditional farming practices, presenting a paradigm shift those benefits both farmers and consumers alike. Bayesian optimization is particularly valuable in the context of hyperparameter tuning, as it leverages probabilistic models to explore the hyperparameter space efficiently. This improves the performance of deep neural networks and traditional ML algorithms for effectively classifying tomato leaf diseases, thus reducing the need for extensive manual parameter tuning and facilitating sustainable farming practices.

The rest of the paper is arranged in the following way.

In "Literature review" section, we discussed relevant studies and advancements in plant leaf disease detection through a comprehensive review of existing literature and cutting-edge research. Building upon the insights gained, we presented the suggested hybrid methodology in "Methodology" section. Recognizing the complexity of the issue, we crafted a multifaceted approach that combines the strengths of different techniques to enhance the overall efficacy of disease detection in plant leaves. This section articulates the conceptual framework, methodology, and potential advantages of our suggested hybrid model. "Results and discussion" section contains the results of the hybrid models and their comparison with existing studies in the field. The concluding remarks, summarizing the key findings and implications of this study, are presented in "Conclusions" section, thus aligning with the broader trend of incorporating automated systems for crop disease recognition to address challenges in food security and sustainable farming practices.

Literature review

After realizing the potential of AI in studying the plant science domain, ML and DL-based techniques became the two categories of well-researched methods for categorizing infected plants or plant leaf regions in the current literature. This section provides a comprehensive investigation of some of the most recent and prominent studies on automated disease recognition systems. For instance, Batool et al.[1] used the transfer learning (AlexNet) technique to extract features from 450 images and traditional ML (KNN) for the classification of 9 tomato leaf diseases. The authors[1] achieved 76.1% classification accuracy using 60% of the data for training and 40% for testing. A machine learning method using different feature extractor techniques, i.e., Hu moments (shape feature), haralick (texture feature), color histograms (color feature), and local binary pattern (LBP), was proposed by Basavaiah & Arlene Anthony[6] to classify four types of tomato leaf diseases with an accuracy of 94% and 90% by using RF classifier and DT classifier, respectively. In another work by Harakannanavar et al.[9], the Discrete Wavelet Transform (DWT), Principal Component Analysis (PCA), and Grey Level Co-occurrence Matrix (GLCM) statistical models were used to extract features, and SVM, KNN, and CNN were used for the classification of tomato leaf disorders. The model[9] achieved an accuracy of 99.09%, precision of 0.995, recall of 0.995, and f1-score of 0.988 using the CNN classifier. Agarwal et al.[26] developed a custom CNN of six feature extraction layers and two classification layers for the classification of 9 tomato leaf diseases and achieved an

average accuracy of 91.2%. A hybrid CNN-RNN classifier developed by David et al.[27] obtained an accuracy of 81.75% and 98.25% as the best categorical accuracy for the classification of tomato leaf diseases. Islam et al.[40] suggested a model for tomato leaf disease detection using attention-dilated CNN (ADC) and logistic regression (LR) classifiers. The authors[40] extracted features by ADC, smoothed them by bilateral filtering, removed noise by the Otsu method, and used an LR classifier for classification. This study[40] found training and testing accuracy of 100% and 96.6%, respectively, which indicates the overfitting of the model. Currently, researchers are emphasizing the application of Bayesian optimization in the detection of plant leaf diseases. For example, Wang et al.[41], da Rocha et al.[42], Restrepo-Arias et al.[43], and Seno et al.[44] utilized various deep learning-based systems, incorporating Bayesian tuning to enhance their performance. Despite being commendable, these studies advocate for the broader adoption of Bayesian optimization in plant leaf disease classification. Tm et al.[45] proposed a simple CNN-based LeNet model to detect and identify tomato leaf diseases and achieved an average accuracy of 94–95%. However, the model suffers from challenges related to optimization in terms of learning rates, optimizers, and hidden layers which can significantly impact convergence and the accuracy of the model. Rangarajan et al.[46] used pre-trained DL (VGG16 net, AlexNet) models to classify tomato leaf diseases and found that AlexNet (97.49%) showed higher classification accuracy than VGG16net (97.23%). Using a fine-tuned MobileNet V2 and Adagrad optimizer, Zaki et al.[47] obtained an accuracy of 90% on tomato leaf disease classification. Liu and Wang[48] found a detection accuracy of 92.39% using an improved Yolo V3 model to detect six types of tomato pests and diseases. Zhang et al.[49] developed a multi-feature fusion faster (MF^3) R-CNN based soybean leaf disease detection algorithm and found a mAP score of 83.34% on the test dataset. The presence of multiple diseases on a single soybean leaf may limit the model[49] performance. Another study was conducted on CNN (4 convolution layer) and transfer learning-based approaches, i.e., VGG19, DenseNet-201, and ResNet152v2, by KP and Anitha[50] to classify grape plant leaf diseases and found that DenseNet-201 obtained precision of 98.31%, recall of 98.27%, f1-score of 98.28%, and overall accuracy of 98.27% over other algorithms. Repeatedly replicated data and higher model parameters in DenseNet[50] lead to expensive memory consumption and computation. Kibriya et al.[51] deployed two CNN-based models named VGG16 and GoogLeNet, which exhibited great performance for tomato leaf disease classification, with an accuracy of 99.23% for GoogLeNet and 98% for VGG16. Another model by Chen et al.[52], which showed quite good performance, obtained average accuracy, precision, recall, and F-measure of 96%, 98%, 95%, and 97%, respectively, by using a modified AlexNet-based CNN model. Alruwaili et al.[53] presented a real-time-based DL model named RTF-RCNN for detecting five tomato leaf diseases with an accuracy of 97.42%. RCNN is a time-consuming method and may classify incorrectly due to the progressive search of the image. Real-time object detection models also have issues regarding scaling and latency. Recently, Soeb et al.[54] conducted a study on five types of tea leaf disease detection using the YOLOv7 algorithm. The authors[54] found 97.3%, 96.4%, 96.7%, 96.5%, and 98.2% accuracy, recall, precision F1-score, and mAP, respectively. Ahmad et al.[55] proposed machine learning-based plant disease phenotyping for three potato, six tomato, and three apple leaf diseases. The highest detection efficiencies of this[55] study were 97.8% for apples, 96.2% for potatoes, and 95.6% for tomatoes, using Directional Local Quinary Patterns (DLQP) and medium gaussian kernel-based SVM. In another study, Turkoglu et al.[56] used real-time apple pest and disease images to develop a hybrid model, Multi-model LSTM-based CNNs (MLP-CNNs). GoogleNet, AlexNet, and DenseNet201 were used for feature extraction, and LSTM and SVM for classification. The study[56] found that concatenated models performed better than standalone models, and the best accuracy (99.2%) was obtained using MLP-CNNs and LSTM classifiers. Al-gaashani et al.[57] concatenated features from MobileNetV2 and NASNetMobile and then used SVM, RF, and MLR for tomato leaf disease classification. The study[57] found MLR provides a maximum average accuracy of 97%. Additionally, various other studies have proposed different models for the detection of plant leaf diseases, such as ResNet50-SeNet[58], C-GAN-DenseNet121[59], CNN-Autoencoders[60], Inception-v3[61], CAE-CNN[62], Pre-trained models and LDDTA[63], ResNet-34 based Faster-RCNN[64], $DSGAN^2$-IAPO[65], VGG-19-LR[66], ResNet-152 and Inception-v3[67], ResNet-50-SVM[68], and Custom CNN[69], all of which demonstrate considerable accuracy and performance metrics.

However, based on the discussion of the earlier literature, it was noticed that most of the models are computationally expensive, time-consuming, improperly tuned, exhibit low performances, have small datasets, and cover only a limited number of disease classes.

An organized review of the aforementioned scientific advancements in plant leaf disease detection is presented in Table 1. Though the studies have demonstrated substantial advances in automated plant leaf disease detection and classification, there are still some gaps that need to be addressed in order to increase the models' accuracy, generalization, computational time, cost, and efficacy.

Methodology

In this section, we elaborately explained data acquisition, preparation, and preprocessing and proposed Bayesian optimized hybrid methodology. A comprehensive illustration of the proposed methodology is shown in Fig. 1.

Dataset acquisition

A large amount of representative and authentic data is required to construct effective and optimized crop disease classification and detection models. Machine learning and deep learning systems perform better when fed a large volume of data. In this proposed methodology, we used a diverse, open-access, research-oriented 'plantvillage' dataset[70,71] for tomato leaf disease classification. It is a versatile dataset that contains more than 54,000 images of 14 types (i.e., fruits and vegetables) of crops. This expert's contributed dataset has images in color, segmented, and grayscaled variations. For our research, we used 18,159 colored and labeled images of tomato leaves divided into 10 different classes (16,568 images for 9 disease categories, i.e., Bacterial spot: 2127, Early blight: 1000, Late blight: 1908, Leaf mold: 952, Septoria leaf spot: 1771, Target spot: 1404, Tomato leaf mold (TMV): 373,

Author	Approach	Method/Model/Technique	Limitation
Batool et al.[1]	Hybrid	AlexNet + KNN	Small and perfectly balanced dataset
Basavaiah and Arlene Anthony[6]	ML	Different feature extraction techniques + DT/RF	Limited disease classes and feature extraction techniques
Harakannanavar et al.[9]	Hybrid	DWT + PCA + GLCM + CNN	Limited disease classes
Mohanty et al.[19]	DL	GoogLeNet, AlexNet	Computationally expensive
Agarwal et al.[26]	DL	CNN	Limited disease classes
David et al.[27]	Hybrid	CNN-RNN	The dataset was minimal
Islam et al.[40]	Hybrid	ADC + LR	Overfitting issue
Wang et al.[41]	BO Hybrid	ADSNN-BO	Insufficient hyperparameter tuning
Da Rocha et al.[42]	BO DL	AlexNet, ResNet50, SqueezeNet	Lack of extensive hyperparameter optimization
Restrepo-Arias et al.[43]	BO DL	MobileNet	Insufficient hyperparameter optimization
Seno et al.[44]	BO DL	AlexNet based BCNNs	Does not include extensive hyperparameter optimization
Tm et al.[45]	DL	LeNet	Suffers from optimization
Rangarajan et al.[46]	DL	VGG16 net, AlexNet	Time-consuming and computationally expensive
Zaki et al.[47]	DL	Fine-tuned MobileNet V2	Limited disease classes
Liu and Wang[48]	DL	Improved Yolo V3	Limited disease classes and low-quality images
Zhang et al.[49]	DL	MF^3 R-CNN	Small dataset and low classification performance
KP and Anitha[50]	DL	DenseNet-201	Small dataset, limited classes, computationally inefficient
Kibriya et al.[51]	DL	GoogLeNet and VGG16	Limited disease classes
Chen et al.[52]	DL	Modified AlexNet	Need for faster, reliable classification approaches
Alruwaili et al.[53]	DL	RTF-RCNN	Time-consuming method
Soeb et al.[54]	DL	YOLOv7	Computationally expensive
Ahmad et al.[55]	ML	DLQP + SVM	Needs more features, classes, and ensemble classifiers
Turkoglu et al.[56]	Hybrid	MLP-CNNs	Computationally expensive and time-consuming
Al-gaashani et al.[57]	Hybrid	MobileNetV2 + NASNetMobile + MLR	Limited disease classes
Zhao et al.[58]	Hybrid	ResNet50 + SeNet	Computationally expensive
Abbas et al.[59]	Hybrid	C-GAN-DenseNet121	High parameter count
Khamparia et al.[60]	Hybrid	CNN + Autoencoders	Overfitting issue
Haque et al.[61]	DL	Inception-v3	High parameter count
Bedi et al.[62]	Hybrid	CAE + CNN	Limited disease classes
Alam et al.[63]	DL	Pre-trained models, LDDTA	High parameter count
Nawaz et al.[64]	Hybrid	ResNet-34 based Faster-RCNN	Results indicating potential overfitting
Mahadevan et al.[65]	Hybrid	$DSGAN^2$-IAPO	Some classification metrics need improvement
Tiwari et al.[66]	Hybrid	VGG-19 + LR	Very high parameter count
Sanga et al.[67]	DL	ResNet-152, Inception-v3	Computationally expensive
Mohameth et al.[68]	Hybrid	ResNet-50 + SVM	Computationally expensive
Rahman et al.[69]	DL	Custom CNN	Need exploration of other architectures, data balancing techniques, and diverse datasets

Table 1. Review of existing leaf disease methodologies with limitations. N.B. BO for Bayesian optimized.

Two-spotted spider mites (TSSM): 1676, Tomato yellow leaf curl virus (TYLCV): 5357, and 1591 images for the Healthy class). Some of the images were blurry, noisy, and had uneven lighting and low contrast. The sample images of the tomato leaf diseases are depicted in Fig. 2.

The following are the signs and symptoms of the nine tomato leaf diseases:

Target spot is a result of the fungal pathogen *Corynespora cassiicola*. Initial symptoms include small, irregularly shaped lesions, which later develop into larger, ring-shaped lesions. This leads to the yellowing and necrosis of the leaves.

Tomato yellow leaf curl virus (TYLCV) is transmitted by silverleaf whitefly (*Bemisia tabaci*). TYLCV infection induces severe symptoms in tomato plants, including yellowing of leaves, curling of shoots, and shortened shoots. This type of infection can result in considerable yield reductions of up to 100%.

Tomato late blight is attributed to the oomycete pathogen *Phytophthora infestans* (P. infestans). Symptoms of late blight on tomato leaves and fruit are characterized by irregularly shaped, water-soaked lesions and fruit rot, distinguishing it from early blight.

Early blight of tomatoes, mainly caused by the fungus *Alternaria linariae* (= *A. tomatophila*, previously known as *A. solani*), affects the foliage, fruit, and stems of tomato plants. It initially shows up as lesions on the lower leaves and can eventually spread to the entire fruit.

Bacterial spot, caused by *Xanthomonas vesicatoria*, *Xanthomonas euvesicatoria*, *Xanthomonas gardneri*, and *Xanthomonas perforans*, manifests as brown, circular, water-soaked spots on the leaves. Under favorable conditions, these spots can merge to form long, dark streaks.

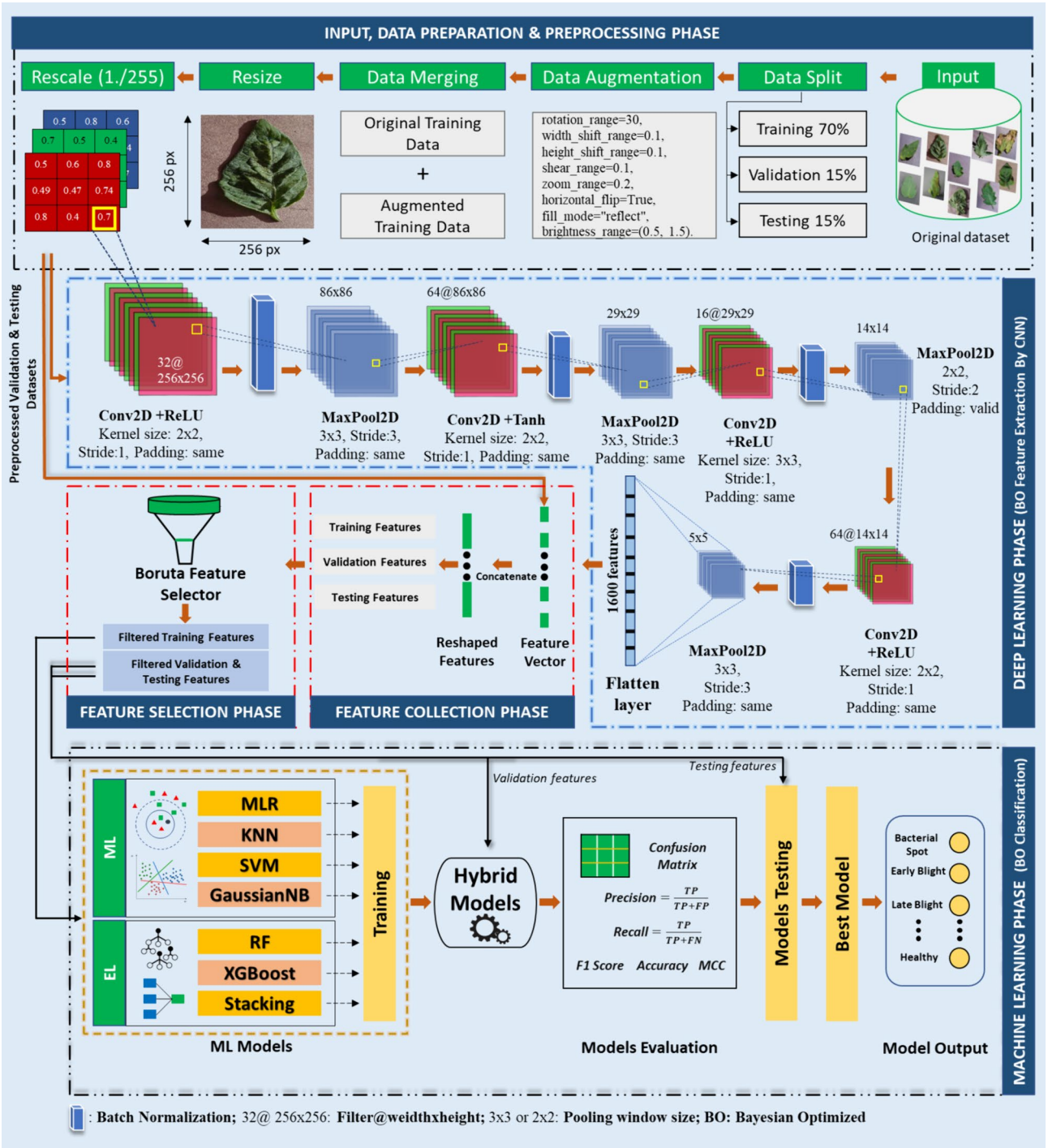


Fig. 1. Detailed illustration of the proposed Bayesian hybrid learning methodology.

Tomato leaf mold is initiated by a fungal pathogen called *Passalora fulva* (also known as *Cladosporium fulvum*). The symptoms of this disease appear as light green spots on the leaves, which later develop into olive-green conidia and necrotic spots.

Tomato mosaic virus is transmitted from one plant to another by several species of *aphids*. Infected plants exhibit a range of symptoms, such as irregular fruit shapes, fruit lesions, and reduced fruit size. Other symptoms include deformed growth points, unusual leaf coloration, shapes, patterns, twisted stems, and overall plant distortion and stunting.

Two-spotted spider mites cause tan or yellow, crusty-textured symptoms on the undersides of affected leaves.

Septoria leaf spot is caused by the fungus *Septoria lycopersici*. Initial symptoms include small, water-soaked, circular spots on the undersides of older leaves, which then develop gray to tan centers with a dark-brown margin.

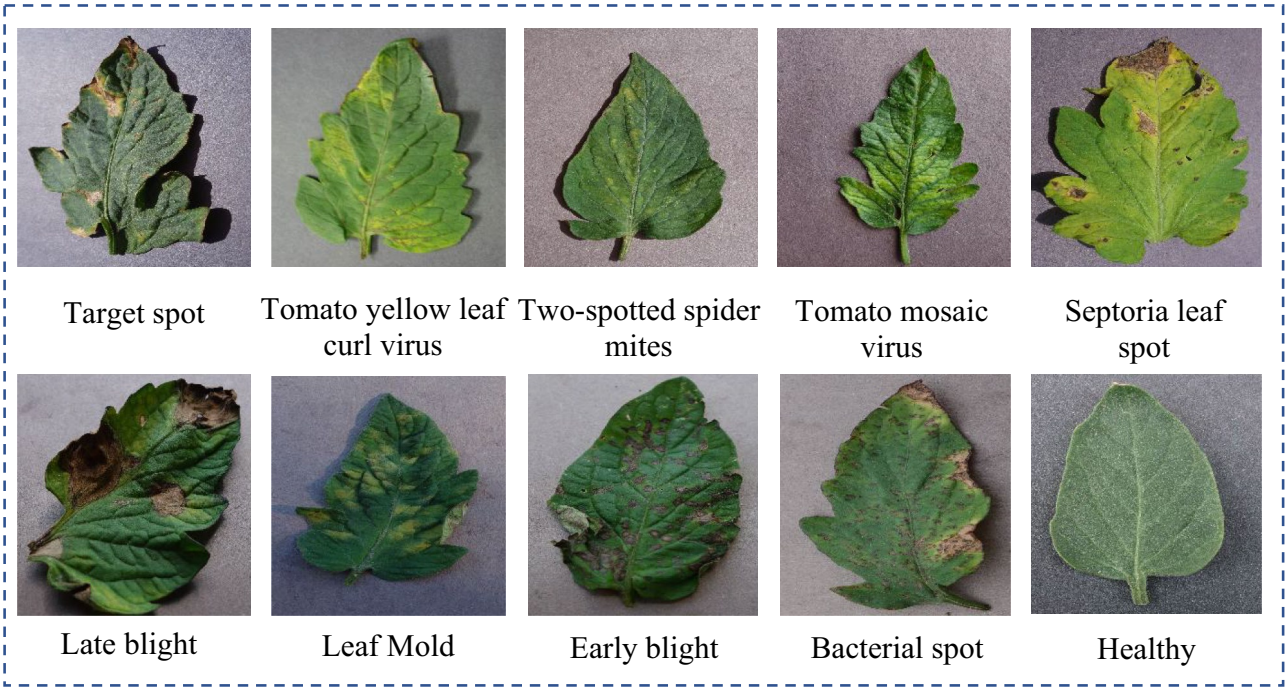


Fig. 2. Sample images of PlantVillage dataset (Source: PlantVillage Dataset (kaggle.com)).

Data preparation and preprocessing

An essential step in the pipeline of the disease classification model is image preparation and pre-processing. Since the raw images might vary in size, contain noise, or have uneven illumination, pre-processing is necessary before applying them to any deep learning models[72]. Additionally, proper preprocessing of data impacts the model's performance. First, we renamed the lengthy class labels into an easily readable format (i.e., 'tomato_yellow_leaf_curl_virus' into 'TYLCV', 'tomato_mosaic_virus' into 'TMV", and 'two-spotted_spider_mites' into 'TSSM').

In the second stage of data preparation and preprocessing, we distributed the size of the classes available in the dataset. For the class-specific data preparation, we applied two popular techniques. These include data augmentation and random downsampling. The detailed descriptions of the adopted approaches are described in "Data augmentation" and "Random downsampling" sections.

Dataset splitting

The original dataset was divided into three sub-datasets: training, validation, and testing datasets. The division was executed by randomly selecting 70% of the data for training purposes, allocating 15% for validation, and reserving another 15% for testing. The training dataset was used for training purposes, and the validation dataset was used to assess the models' performance after training. A validation dataset is also needed for evaluating the tuned hyperparameters in order to improve the models' performance. Once the models were verified by the validation dataset, the testing dataset was utilized for testing. The comprehensive description of these three datasets is presented in Tables S1 and S2. These three datasets were stored locally in three folders, and each folder was divided into ten subfolders with the names of the disease classes. The folder arrangement is graphically depicted in Fig. S1.

Data augmentation

As some of the classes possessed a higher number of images than others, an imbalance problem arose in the training dataset. Training a deep learning model using an imbalanced dataset can result in poor generalization. This can occur because the model may become too biased toward the majority classes and fail to perform well for the minority classes. In order to prevent the issue of an imbalanced dataset, a widely used method known as data augmentation was applied. Data augmentation serves as a regularization strategy to avoid overfitting and enhance the model's robustness by introducing variability in the dataset. This also ensures that the model performs more accurately when categorizing real-life plant leaf diseases[73]. However, the main aim of data augmentation is to increase the dataset's size artificially[74]. In this study, we utilized data augmentation for nine classes using the 'ImageDataGenerator' class from Keras to generate augmented images from the original images to balance the training dataset by increasing the number of images in those classes. This process correctly preserved the labels on augmented images. We maintained the images in the range of 1400 to 2130 for different classes to create a balanced dataset, although not perfectly balanced since perfectly balanced datasets are rare in real life. The number of augmented images was determined by setting the required number of image samples per class and then subtracting the image numbers already present in the training dataset. We increased image samples for the 'Bacterial spot', 'Early blight', 'Healthy', 'Late blight', 'Leaf mold', 'Septoria leaf spot', 'Target spot', 'TMV', and 'TSSM'

classes from 1489, 700, 1114, 1336, 667, 1240, 983, 262, 1174 to 2127, 1600, 1591, 1909, 1852, 1771, 1404, 1990, and 1676, respectively, by using different augmentation techniques. The implemented augmentation techniques are rotation_range = 30, width_shift_range = 0.1, height_shift_range = 0.1, shear_range = 0.1, zoom_range = 0.2, horizontal_flip = True, fill_mode = "reflect", brightness_range = (0.5, 1.5).

The augmented images were saved with the ".jpg" extension in the augmented folder, and the 'aug_' prefix was used to differentiate the original training images from the augmented images (Fig. S2). Then the original and augmented images were merged to generate the final training dataset. The distribution of the images is illustrated in Fig. S3.

Random downsampling

Random downsampling is the technique of removing data samples from the majority class at random in order to prevent them from dominating the learning algorithm. In machine learning, it is frequently used to balance the dataset so that the training data is more representative and consumes less memory. In the "TYLCV' class, the image samples were comparatively higher than in other classes in the original training dataset. So, for this class, we downsampled images from 3750 to 2000 by randomly selecting 2000 images using the 'random.sample' function of the Python 'random' module.

The following Algorithm 1 depicts the process of data augmentation and random downsampling.

Start

$N_{requisite}$ (c) = desired number of training samples per class (Bacterial spot: 2127, Early blight: 1600, Healthy: 1591, Late blight: 1909, Leaf mold: 1852, Septoria leaf spot: 1771, Target spot: 1404, TMV: 1990, TSSM: 1676, and TYLCV: 2000)

$N_{augment}$ (c) = number of augmented training samples

For each class c:
  $N_{original}$ (c) = number of original training images in class c
  If $N_{original}$ (c) < $N_{requisite}$ (c):
    $N_{augment}$ (c) = $N_{requisite}$ (c) - $N_{original}$ (c)
    Apply augmentation methods to generate $N_{augment}$ (c) new images
  If $N_{original}$ (c) > $N_{requisite}$ (c):
    Perform random downsampling to reduce to $N_{requisite}$ (c)
Save augmented images with prefix 'aug_'
Merge $N_{original}$ (c) + $N_{augment}$ (c)

End

Algorithm 1. Process of data augmentation and random downsampling.

Resizing and rescaling

Resizing operations in deep learning return homogeneous images by downscaling the input images. The performance and training time of deep learning models are significantly influenced by image resolution[75–78]. In general, CNNs operate with image resolutions that are typically low to mid-level, commonly ranging between 64 × 64 and 256 × 256[76]. Several studies have shown that the deep learning model performs better at 256 × 256 image resolution than at other resolutions[76,77]. Therefore, to resize the images into a target resolution of 256 × 256 pixels, we utilized the 'flow_from_directory' method from the 'ImageDataGenerator' class from the 'Keras' API. After resizing, we applied a rescaling (also called normalization) operation to the images using ImageDataGenerator (rescale = 1/255) from the tf.keras.preprocessing.image module. With this rescaling technique, the input images' pixel values were rescaled from the Numpy arrays of [0, 255] to the Numpy arrays of [0, 1]. Numpy arrays are a memory-efficient, faster data structure for numerical computation, model training, and evaluation. These numpy arrays were passed as input to the custom CNN model for feature extraction.

Proposed deep hybrid learning (DHL) approach

In the proposed Bayesian optimized deep hybrid learning approach, we employed deep learning for feature extraction and machine learning for feature categorization. A custom CNN model was used as the deep learning framework, and seven advanced supervised learning algorithms were applied as the machine learning framework. The hyperparameters of the deep learning and machine learning models were optimized using the Bayesian optimization approach.

Bayesian optimization

Machine learning algorithms require precise adjustment and optimization of learning parameters and hyperparameters. Hyperparameters (e.g., the kernel in SVM, n_estimators in a random forest, n_neighbors in KNN;

learning rate, hidden layers, etc. in a neural network) are those technical terms that we can alter according to the nature of the dataset and the specific problem at hand. Mathematically, hyperparameter optimization can be represented[79] as $\mathbf{x}'=\mathbf{argmin\ f(x)}$, where $x \in$ domain X, f(x) = objective function (i.e., loss), x′ hyperparameters that give the lowest value of f(x). There are various techniques available to optimize the hyperparameters. Sequential Model-based Optimization (SMBO), also known as Bayesian optimization, is considered to be the most promising approach when compared to other existing hyperparameter tuning methods such as manual search, grid search, and random search. Manual search takes a lot of time, effort, and domain knowledge to locate the ideal hyperparameters. Grid search and random search are also time-expensive processes compared to Bayesian optimization[42]. Bayesian optimization (BO) is a reliable iterative method that is well-known for its effectiveness since it uses previous results to inform future evaluations. It is particularly effective for optimizing costly black-box functions with multiple external dependencies. BO's effectiveness hinges on two primary components: surrogate models and acquisition functions. The surrogate model mimics the objective function using a probability distribution, and the acquisition function directs the search by balancing exploration and exploitation. Exploration involves investigating the unexplored region, whereas exploitation focuses on sampling locations where the surrogate model predicts a high objective value. For Bayesian optimization, Gaussian processes (GP) are the popular surrogate models since they are simple to use, easily adaptable to new data, and offer a confidence level for each of their predictions. The Gaussian process model generates a probability distribution over possible functions. A mean function, or the average appearance of these possible functions, and a kernel function, or the degree to which these functions change across inputs, define this distribution. On the other hand, the commonly used acquisition functions are GP Upper Confidence Bound (GP-UCB), Probability of Improvement (PI), and Expected Improvement (EI).

The Bayesian optimization relies on constructing and updating a probabilistic model of the objective function, which can be computationally intensive[80], especially when approaching high-dimensional or noisy functions. Additionally, finding the accurate balance between exploration and exploitation is crucial for effective Bayesian optimization[81]. Zoubin Ghahramani[82] noted that the current most effective global optimization methods uphold a Bayesian representation of the probability distribution over the uncertain function 'f' undergoing optimized and utilize this uncertainty to determine the next query location within the search space 'X'. Hence, despite its limitations, Bayesian optimization is widely favored in professional settings due to its efficient optimization of costly evaluations using probabilistic models. The present study implemented Bayesian optimization (a Gaussian process-based surrogate) during the deep learning phase, utilizing the KerasTuner framework. The optimization was performed using 100 trials, and the optimal hyperparameters were obtained from the trial with the highest validation accuracy.

In addition to the conventional Bayesian implementation, the study considered one of the advanced Bayesian optimization techniques called Tree-structured Parzen Estimator for the optimization of machine learning models.

Tree-structured Parzen Estimator. TPE[83], or Tree-structured Parzen Estimator, represents a sophisticated variant of the Bayesian optimization method that employs the Gaussian mixture model for learning model hyperparameters. It considerably outperforms random search techniques regarding optimization efficiency[84] and finds better hyperparameters in the same number of trials with low trial time[79]. Furthermore, with the TPE approach, the computing time for each iteration scales linearly, whereas for the Gaussian process, it scales cubically.

TPE diverges from the conventional Bayesian optimization approach by modeling P(x|y), which signifies the probability of the hyperparameters (x) given the value (y) of the objective function. In TPE, the algorithm transforms the configuration space, replacing uniform with truncated Gaussian mixtures, log-uniform with exponentiated truncated Gaussian mixtures, and categorical variables with re-weighted categorical. By incorporating different observations $\{x^1,\ldots, x^k\}$ into the non-parametric densities, these substitutions represent a learning algorithm capable of generating various densities across the configuration space.

TPE maintains two densities (Eq. 1) based on a threshold value (y*). This threshold divides the hyperparameter search space into two distinct categories: the "good" category, denoted as l(x), and the "bad" category, denoted as g(x).

$$P(x|y) = \begin{pmatrix} l(x); & \text{if } y < y^* \\ g(x); & \text{if } y \geq y^* \end{pmatrix} \tag{1}$$

where l(x) represents the density consisting of observations x^i such that the corresponding loss $f(x^i)$ is less than the threshold (y*) and the g(x) represents the density comprising the remaining observations. The TPE algorithm relies on the threshold y* that exceeds the best observed f(x) to allow for the formation of l(x) using certain points. In this approach, y* is selected as some quantile $\boldsymbol{\gamma}$ of the observed y values, ensuring that $p(y < y^*) = \gamma$.

The parameterization in the TPE algorithm is specifically designed to streamline the optimization of Expected Improvement (EI). As indicated in the study by Bergstra et al.[83], the expected improvement in the TPE algorithm can be represented as

$$EI_{y*}(x) \propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1}$$

This expression indicates that, in order to maximize improvement, we aim for points x that have a high probability under $\ell(x)$ and a low probability under g(x). The tree-like structure of ℓ and g simplifies the process of drawing numerous candidates based on ℓ and assessing them based on $\mathbf{g(x)/\ell(x)}$. During each iteration, the procedure selects the candidate x^* with the highest expected improvement.

For utilizing the TPE, we used the Hyperopt[85], which is a hyperparameter optimization library consisting of 4 distinct phases, such as (a) search space or domain space, which includes the hyperparameters to be tuned and their corresponding value ranges (categorical, integer, or float); (b) objective function, that is, a loss function to minimize; (c) search algorithm; and (d) trails database. Hyperopt is supported by the SMBO methodology adapted to work with the TPE algorithm[83].

The configurations of the proposed Bayesian optimization procedure are thoroughly presented in Algorithm 2.

Hyperparameter selection procedure. The hyperparameter selection procedure adopted for this study is divided into several steps. The steps are described below.

- First, we trained the models using the default hyperparameter settings to observe their performance. Default hyperparameters may not exploit the full potential of the model for a given problem. Additionally, an unoptimized learning rate might lead to slow convergence or overshooting. In most cases of this study, the default values of hyperparameters did not provide sufficient regularization, leading to overfitting, and in some cases, they provided too much regularization, causing underfitting.
- Next, we tuned the hyperparameters manually to get an idea of the best possible range of hyperparameters. This manual tuning approach slightly reduced overfitting issues.
- Finally, we applied Bayesian search to tune the hyperparameters. The manual hyperparameter tuning helped in selecting the proper interval of hyperparameters for the Bayesian search space. The Bayesian search space searched for the best set of hyperparameters for the employed models. We achieved the best hyperparameters by combining Bayesian search with manual search results.

All hyperparameter tuning strategies were implemented on the training dataset and validated using the validation data. The best set of hyperparameters obtained by the validation was then used for the final training of the models.

Architecture of the proposed Bayesian optimized CNN Model

Convolutional Neural Network[86] (CNN) is a prominent deep learning method that utilizes convolution operations to extract distinct features from input images and classifies them using fully connected layers or dense layers with activations. Several layers, including the convolutional, max pooling, and average pooling, are used in feature extraction. An output layer with the requisite number of classes is added as a final layer. The proposed CNN architecture consists of a series of convolution, batch normalization, and max pooling layers. It also includes flatten, dropout, and dense layers. We used the Bayesian optimization technique for tuning the hyperparameters of the CNN model (i.e., number of 2D convolutional layers, number of filters, kernel size, activation function, regularization on kernel weights, number of MaxPooling2D layers, pool size, etc.). The Bayesian optimized hyperparameters for the proposed CNN is shown in Table S3.

Convolution2D. A 2D convolution layer, or conv2D, is the most common type of convolutional layer. It performs convolution operations on the 2D input image and passes the convolution result to the succeeding layer. It consists of several filters, or kernels. The kernel convolves horizontally and vertically over the entire image, performs the dot product with the input image, and outputs a scalar value each time. Such a combination of scalar values is called a feature map. This way, it decreases the input 2D image size and extracts the relevant features. The primer conv2D layer extracts low-level features such as edge, texture, etc., whereas the final conv2D layer takes the output from the earlier layer and extracts more task-specific, high-level, and complex features. Stride and padding are the two common elements of the conv2D layer that affect convolution operations. Stride detonates the step size of the conv2D window, and padding impacts the spatial dimension of the image. According to Bayesian hyperparameter tuning, our custom CNN model includes four conv2D layers with filters of 32, 64, 16, and 64, respectively. The kernel size is 2×2 (3×3 for third conv2D), which corresponds to the width and height of the conv2D frame. We used "same" padding, so after a convolution, the output size remains the same as the input size. Total padding amount on rows and columns can be determined using Eq. (2). We also set the stride value to 1, so after elementwise multiplication of the filter with the image, the filter window moves across the image both vertically and horizontally by 1 pixel. The output of a CNN layer considering "same" padding and stride can be found by using Eq. (3).

$$\begin{pmatrix} P_r = (R' - 1) * S + (K_r - 1) * D_r + 1 - R \\ P_c = (C' - 1) * S + (K_c - 1) * D_c + 1 - C \end{pmatrix} \tag{2}$$

where (R', C') denote the output dimensions, (R, C) are the input dimensions, S = stride, (D_r, D_c) are the dilations, (K_r, K_c) are the filter dimensions. The dilation rate was maintained at its default setting of (1, 1).

Padding ("same") is applied in the following manner:

$P_r/2$ on the left, $P_r - P_r/2$ on the right, $P_c/2$ on top, and $P_c - P_c/2$ on bottom. When P_r or P_c is odd, the right and bottom receive more padding than the top and left.

$$\text{Output shape}\left(\text{"same" padding}\right) = \text{math.floor}\left(\left(\text{input shape} - 1\right)/\text{strides}\right) + 1 \tag{3}$$

Activation function. The activation function of a neural network develops non-linearity in a neuron's output. It regulates whether or not a particular neuron should be triggered. Activation function makes decisions based

on weighted sum and bias terms. Using Bayesian optimization, we identified that 'ReLU' is the best activation function for the majority of the conv2D layers, and we implemented this using the 'activation' argument available in the conv2D layer. It is a well-known activation function for CNN that outperforms the other activation functions in terms of computational efficiency. ReLU returns the same value if the input is positive; otherwise, it returns zero. However, for the second Conv2D layer, the Bayesian search identified spotted 'Tanh' as the best fit. Tanh maps the inputs in the range of − 1 to + 1. The mathematical equations for these activations are presented in Eq. (4).

$$\begin{pmatrix} f_{ReLU}(x) = \max(x, 0) \\ f_{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \end{pmatrix} \tag{4}$$

MaxPooling2D. A pooling layer is commonly added after a convolutional layer. It performs a downsampling operation on the output of the conv2D layer and outputs a lower-dimensional feature map to the next layer. Among the several pooling operations, MaxPooling2D is the most widely implemented option. It requires minimal computing expenses to extract the most salient features from the input data. We implemented MaxPooling2D using layers. MaxPooling2D() class from TensorFlow's Keras API. The MaxPooling2D frame is defined by the dimensions of the pooling window. For most of the pooling layers, we used the 3 × 3 pooling window, which indicates that from the convolutional output, the window selects the top-left 3 × 3 region and fetches the max value from the 9 elements of the 3 × 3 block and stores this into the output feature map of the MaxPooling2D. Likewise, the pooling window covers all the convolutional output until it reaches the bottom right 3 × 3 region of the convolutional output. The window moves along the spatial dimensions (i.e., width and height) of the convolutional output specified by the stride size. Like conv2D, MaxPooling2D utilizes "valid" padding and "same" padding. In our proposed study, we employed three MaxPooling2D layers with a 3 × 3 stride and "same" padding, and one with a 2 × 2 stride and "valid" padding. The output shape of a feature map using "valid" padding can be estimated by Eq. (5).

$$\text{Output shape}\left(\text{"valid" padding}\right) = \text{math.floor}\left((\text{input shape - pool size}) / \text{strides}\right) + 1 \tag{5}$$

Kernel regularizer. During optimization, we can apply penalties to layer parameters using regularizers. By using the kernel regularizer (L2), a penalty was applied to the layer's kernel. L2 regularization, also known as Tikhonov's[87] or ridge regularization, helps to develop robust models. The penalties are added to the loss function in order to minimize the weight matrix (w) values, which aids in reducing overfitting significantly. We used the most commonly used categorical cross-entropy loss for our proposed multiclass classification. The mathematical equation of the cost function of L2 regularization is shown in Eq. (6).

$$\text{Cost function} = \text{Loss} + \frac{\lambda}{2m}\left[\sum \|w\|^{2}\right] \tag{6}$$

λ is the regularization parameter, and m is the number of inputs. According to Bayesian tuning, the values of λ in the 4 conv2D layers are 0.0001282, 0.0004593, 0.01, and 0.0011262, respectively.

Batch normalization. Like normalized input, batch normalization[88] normalizes the output or activation of a preceding layer using mini-batches rather than the entire dataset. In the proposed CNN architecture, we used four batch normalization layers between each conv2D and MaxPooling2D layer. It substantially smooths out the optimization landscape, which causes the gradients to behave more predictably and consistently, allowing for speedier training and convergence[89]. Batch normalization also impacts weight initialization and controls overfitting by inducing minor regularization through randomness among batch-dependent mean and variance for each neuron activation. The batch's activation values following this process had a zero mean and unit standard deviation. The mathematical equation of the batch normalization operation is expressed in Eq. (7). In addition to this, it also uses two learnable parameters named γ (true mean of activation) and β (true variance of activation) for each neuron to calibrate the mean and variance expressed in Eq. (8).

$$x^{(i)}_{normalized} = \frac{x^{(i)} - \mu_{B}}{\sqrt{\sigma_{v}^{2} - \varepsilon}} \tag{7}$$

$$y^{(i)} = \gamma x^{(i)}_{normalized} + \beta \tag{8}$$

where μ_B and σ_v^2 are the mini-batch mean and mini-batch variance, respectively; ε is a constant for numerical stability.

Flatten layer

The output of the last pooling layer was flattened before passing through the dense layers. Flatten layer computes dimensions from MaxPooling2D output tensors. It reshapes the output tensor into a vector (1D array) with a shape equal to the entire tensor's constituents (without batch dimension). So, after flattening '5, 5, 64' into "1600," we obtained a total of 1600 features for our classification.

Training the custom CNN model

To employ the proposed CNN as an effective feature extraction tool, we trained it using the conventional classifier. The typical CNN allocates one or more fully connected layers (and also some regularization layers) for classification tasks. The present study optimized the configuration of such layers using the Bayesian algorithm. We employed a single dense layer with 32 neurons, a kernel regularizer (l2) of $1.35e^{-05}$, and "relu" as activation. Following this layer, we used a batch normalization layer and a dropout layer[90]. We set a dropout rate of 0.3 (i.e., randomly drop 20% of the neurons) to prevent overfitting. In the output layer, we used 10 neurons, which correspond to the 10 classes of tomato leaves, and the probabilistic 'softmax' function as activation. Additionally, we added two callback operations: EarlyStopping and ReduceLROnPlateau. EarlyStopping tracked the validation loss and stopped the training process if there was no improvement after 20 epochs, hence reducing the overfitting concerns. On the other hand, ReduceLROnPlateau altered the learning rate based on validation loss, lowering it by 0.2 if no improvement (i.e., learning stagnated) was seen after 5 epochs. Finally, we compiled the CNN model with the 'categorical cross-entropy' loss function and the 'Adam' optimizer, with a learning rate of 0.001 and beta_1 and beta_2 values of 0.9 and 0.999, respectively. The model was trained for 96 epochs.

Assessment of the custom CNN model. The performance of the CNN model considering the number of epochs used during the model's training phase has been shown in Figs. S4 and S5. The proposed model has effectively learned from the training data, as evidenced by the continuous reduction in training loss (Fig. S4). Additionally, the consistently decreasing validation loss suggests that the model exhibits good generalization capabilities, avoiding overfitting to the training data. Regarding accuracy (Fig. S5), the model's performance has consistently improved across epochs, both in training accuracy and validation accuracy. The gradually rising validation accuracy underscores the model's predictive capabilities to enhance its accuracy on unseen data. In essence, the convergence between training and validation metrics in Figures S4 and S5 justifies the proposed CNN model's streamlined design, as well as its adequacy and reliability for capturing underlying data patterns and its effectiveness as a feature extraction framework.

Feature extraction and collection

The classical CNN captures multidimensional features using a flatten layer, which are then fed into dense layers for classification. In a dense layer, each neuron is connected to the preceding layer's neurons. When a model has an excessive number of dense layers and neurons within them, the model's parameters rise dramatically, which makes the model complex, computationally expensive, and time-consuming. Moreover, the interpretation of the model becomes complicated due to the complex network. Machine learning (ML), on the other hand, has an extensive range of advanced classifiers with a diverse set of hyperparameters to boost performance. These models offer powerful capabilities, faster execution times, and are simple to interpret. However, the most important considerations in adopting ML algorithms are their low computational costs and data efficiency. Therefore, in our suggested tomato leaf disease study, we took into account the benefits of these ML methods for the classification of CNN-generated features. To accomplish this, we first separated the feature extraction framework from the traditional classification framework by eliminating the last four layers of our custom convolutional neural network. After that, our model was left with 14 layers, i.e., 1 input, 4 conv2D, 4 MaxPooling2D, 4 batch normalization, and 1 flatten layer. We then extracted features for training, validation, and testing datasets using the 'predict()' method. The extracted features and corresponding labels are appended using Python's 'append()' method to six Python lists (e.g., x_train_hybrid, y_train_hybrid; x_val_hybrid, y_val_hybrid; and x_test_hybrid, y_test_hybrid).

Finally, using Numpy's 'concatenate()' function, we concatenated all the feature and corresponding label arrays. For a single image, our proposed framework generated 1600 features. Therefore, the shapes of training, validation, and testing independent and dependent values are (17,920, 1600), (17,920,); (2728, 1600), (2728,); and (2716, 1600), (2716), respectively. The feature extraction framework required approximately 5.8 min (0.019 s/image), 0.79 min (0.017 s/image), and 0.8 min (0.018 s/image) of extraction time for training, validation, and testing datasets, respectively. Some of the extracted features are shown in Fig. S6.

Feature selection by Boruta

Feature selection in ML and DL is a crucial task for improving model generalization, reducing overfitting, and enhancing computational efficiency by prioritizing relevant features. This process is particularly valuable in addressing model complexity, minimizing noise, and acquiring domain-specific insights. Many prior studies have explored dimensionality reduction techniques and feature selection algorithms to select relevant features. While feature selection algorithms may excel at choosing non-redundant features, they can inadvertently overlook significant redundant ones. Dimensionality reduction techniques also fail to convey the importance of individual features for subsequent analysis[91]. Therefore, this study considered a wrapper algorithm named Boruta[92] for filtering the important features for the classification of diseases. The Boruta algorithm works by creating randomness in the system and iteratively searching for relevant features, along with feature ranking.

The Boruta feature selection procedure is a methodical process that involves the following steps:

1. Duplication of original features: Initially, the original features are duplicated to generate shadow features.
2. Shuffling and merging: The duplicate features undergo shuffling to eliminate correlations with the response variable, and subsequently, they are merged with the original features.
3. Training tree-based machine learning classifier: The expanded dataset, comprising both original and shadow features, is used to train the tree-based machine learning classifier (e.g., random forest).
4. Z-scores: Then, the computed Z scores for the features are gathered, and the shadow feature with the maximum Z-score (MZS) is identified.

5. 'Hit' assignment: Each feature that scores better than the MZS receives a hit.
6. Two-sided equality test: The feature with undetermined importance is subjected to a two-sided equality test with the MZS.
7. Identification of important features: Original features with significantly higher importance than the MZS (i.e., Z-scores of original features exceeding the MZS) are designated as important, while those with significantly lower importance are labeled as unimportant and permanently removed.
8. Removing shadow features: Then all shadow features are removed.
9. Iterative evaluation: These above steps are iterated until the importance is assigned for all features or a fixed iteration is reached, ensuring a comprehensive assessment of the features' significance relative to the MZS.

We implemented the Boruta feature selection algorithm by leveraging the 'BorutaPy' class from the 'boruta' Python package. The selection procedure involved instantiating the Boruta feature selector object using the syntax 'BorutaPy (RandomForestClassifier (n_estimators = 491), n_estimators = 'auto', perc = 100, alpha = 0.05, max_iter = 100)' and subsequently fitting it to the training features (x_train_hybrid, y_train_hybrid). BorutaPy provides the 'support_' attribute, which yields a boolean array indicating the significance of each corresponding feature, and the 'rank_' attribute furnishes the feature ranking based on importance. Through successive iterations, the algorithm identified 1175 (out of 1600) important features with corresponding ranks. Once the relevant features were identified using the 'transform()' method for training, validation, and testing sets, we stored them in three distinct Numpy arrays. Afterward, the Bayesian optimized machine learning models were trained using the Boruta-selected features.

The proposed novel feature extraction and selection mechanisms are elaborately depicted in Fig. 3.

Proposed Bayesian optimized classification approaches

In this study, we examined seven Bayesian optimized machine learning models for classification: Multinomial logistic regression (CNN-MLR), Gaussian Naïve Bayes (CNN-GNB), Support vector machine (CNN-SVM), K-Nearest Neighbours (CNN-KNN), XGBoost (CNN-XGBoost), Random Forest (CNN-RF), and Stacking (CNN-Stacking) classifiers. The XGBoost, Random Forest, and Stacking classifiers are commonly known as Ensemble Learning (EL) techniques. EL is a subset of machine learning where the predictions of different learning algorithms are integrated into the prediction pipeline to achieve improved decisions. It is capable of solving sophisticated problems with higher performance metrics where standalone models may not perform well. Ensemble learning strategies, for instance, bagging (e.g., random forest) reduces variance, boosting (e.g., XGBoost) reduces bias, and stacking improves the prediction capabilities.

Multinomial logistic regression. Multinomial logistic regression (MLR), often known as the softmax classifier, is a probability-based classification technique. MLR is an extension of logistic regression in which the outcomes can be more than two. For k target classes, we feed the model with a set of features $X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n)$ and it outputs a computed probability vector $Y(y_1, y_2, y_3, \ldots, y_k)$ where $\sum_{i=1}^{k} y = 1$. The predicted class is the probability of the highest class. In our study, k = 10, therefore the softmax algorithm can be defined as (Eq. 9):

$$P(y = i|z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}; \quad \text{where } z_i = \text{logit function} = w_i \cdot x + b_i \tag{9}$$

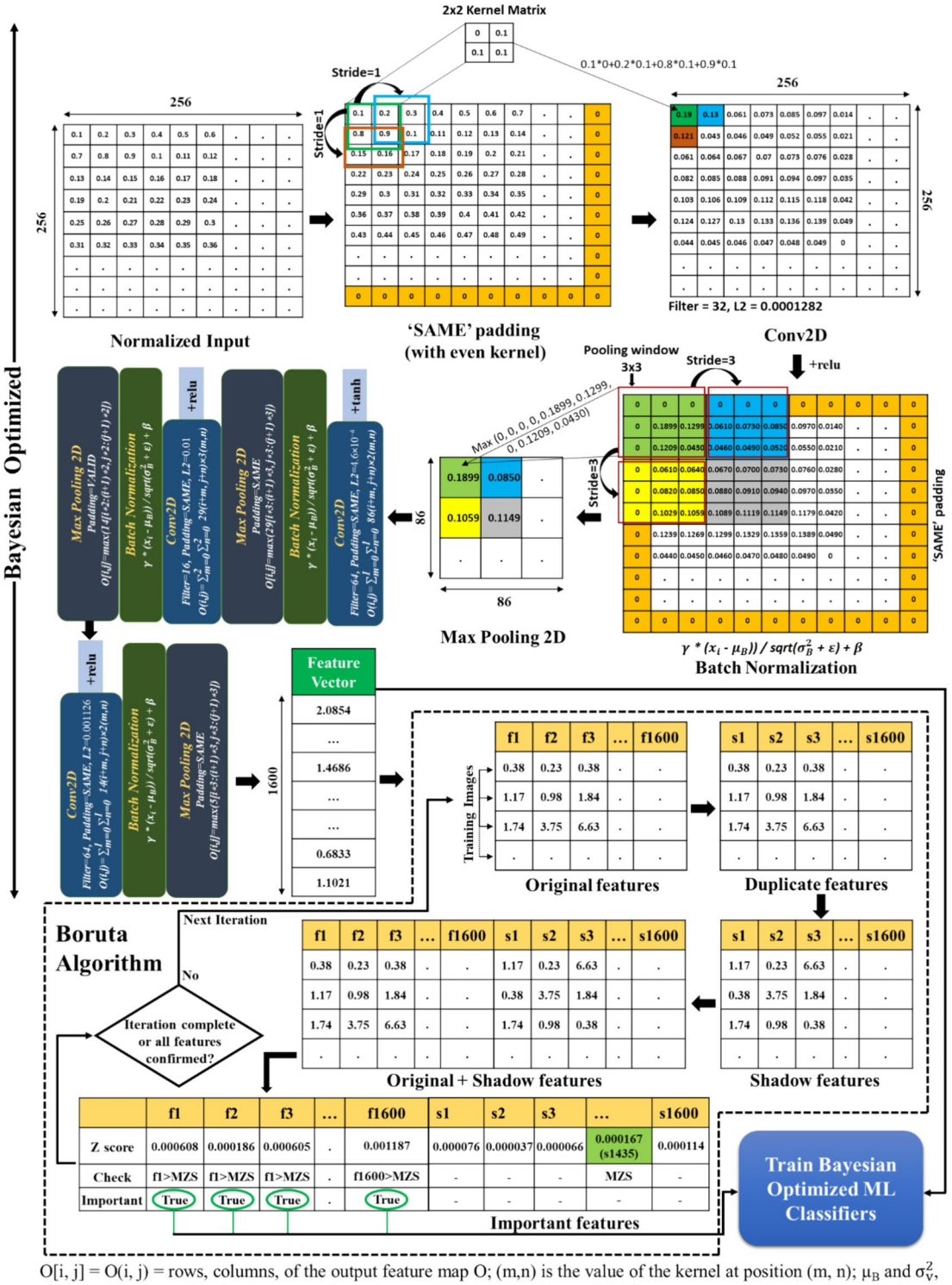
where $P(y = i|z_i)$ is the probability of the target class y being class i given the linear score z_i. The loss function (difference between the predicted probabilities and the actual class labels) for MLR is the cross-entropy loss, or negative log-likelihood. Equation 10 shows the loss function for only the correct class (k).

$$\text{Cross entropy loss (ypred, y)} = -\log(\text{ypred(k)}) = -\log\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{10}$$

KNN. K-Nearest Neighbors (KNN) is a simple but powerful algorithm used for classification and regression problems. It is an instance-based algorithm, meaning it relies on instances (or examples) from the training data to make predictions. For our proposed scenario, we classified 10 types of tomato leaves based on 1175 features. Given these features, we defined a tomato leaf in our dataset as a point in a multi-dimensional space where the axes represented the features. To classify a new tomato leaf, we plotted it on this graph, then found the 'k' tomato leaves closest to it (where 'k = 2' is chosen by Bayesian tuning). New tomato leaf was then classified according to the majority class of these 'k' neighbors. For this, we calculated the 'euclidean' distance between the new entry and the rest of the entries in the dataset using Eq. (11).

$$d = \text{sqrt}[(x_2^1 - x_1^1)^2 + (x_2^2 - x_1^2)^2 + \ldots\ldots\ldots + (x_2^{1175} - x_1^{1175})^2] \tag{11}$$

where d is the distance and x_2^1, x_1^1 are the 1st feature of new datapoint and existing datapoint respectively. Using this formula, we calculated the distances for all the datapoints, and then we sorted all the distances in ascending order and picked the first 'k = 2' points. Finally, the new tomato leaf data point was assigned to the class that had the majority among the 'k = 2' nearest neighbors.



O[i, j] = O(i, j) = rows, columns, of the output feature map O; (m,n) is the value of the kernel at position (m, n); μ_B and σ_V^2, mini-batch mean and variance; ε = small positive number; γ, β = scaling and shifting parameters; L2= kernel_regularizer

Fig. 3. Proposed novel feature extraction and selection mechanisms.

SVM. Support Vector Machines (SVM) is a supervised machine learning algorithm that is commonly used for classification and regression tasks. The SVM algorithm works by finding the hyperplane that maximizes the margin between the two classes. In 2-D space, this hyperplane is a line. Suppose we have a set of training examples $\{(x_1, y_1),\ldots, (x_n, y_n)\}$, where each $x_i \in R^d$(x_i is a d-dimensional real vector), and $y_i \in \{-1, 1\}$ (y_i represents the class label). The SVM algorithm seeks to find the optimal hyperplane that separates the two classes by solving the following (Eq. 12) optimization problem:

$$\text{Minimize } (1/2)\ ||w||^2 \text{ subject to } y\ (w \cdot x + b)\ \geq 1 \text{ for all } 1\ \leq\ i\ \leq\ n. \tag{12}$$

where w is the normal vector to the hyperplane, ||w|| is the Euclidean norm of w, b is the bias term, and '·' denotes the dot product. Let's denote features by $x_1, x_2,\ldots,x_{1175}$. The values $y = 1$ and $y = -1$ correspond to 'TMV class' and 'non-TMV class,' respectively. Suppose the SVM algorithm has found the optimal hyperplane to be $w \cdot x + b = 0$, where $w = [w_1, w_2,\ldots\ \ldots,w_{1175}]$ is the weight vector, $x = [x_1, x_2,\ldots\ \ldots,x_{1175}]$ is the input vector, and b is the bias term. We can interpret w_1 as the weight for feature-1 and w_2 as the weight for feature-2. A tomato leaf is classified as 'TMV class' if $w \cdot x + b \geq 1$, and 'non-TMV class' otherwise. SVM is effective for linearly separable data. If the data is not linearly separable, kernel tricks such as polynomial kernel, radial basis function (RBF) kernel, etc. are used to map the data to a higher-dimensional space where it is linearly separable. As SVM works mainly for binary classification (2 groups), we implemented this by using the one-vs-rest approach, where hyperplane separates a class (i.e., TMV) by keeping it in group-1 and the rest of the 9 classes (i.e., bacterial spot, early blight, and so on) in group-2. Therefore, the classifier utilizes 10 binary SVMs for 10 classes. For test data, each binary SVM predicts the probability, and the output class is determined by the maximum probability score.

RF. A random forest trains multiple decision trees on random subsets of the training data. The randomness here serves two purposes: each decision tree in the random forest is trained on a different set of data points, and each test in a decision tree is chosen from a random subset of features. Random forests make predictions by having each decision tree in the forest predict the class of the input and then choose the class that was most frequently predicted (majority voting). This is done to increase the diversity of the trees in the forest and decrease the variance of the predictions. Let $X = \{x_1, x_2,\ldots, x_n\}$ be the set of input features, and $Y = \{y_1, y_2, \ldots, y_n\}$ be the corresponding classes. Assume there are m different possible classes. A random forest with T decision trees is trained by: For $t = 1$ to T, choose a random subset of the training data (X_t, Y_t) from X and Y. Train the t-th decision tree by choosing a random subset of the features x_t. Finding the best tests that split X_t into two groups that minimize the impurity of Y_t. To make a prediction for a new input x: Let $Y_t(x)$ be the prediction of the t-th decision tree. The prediction of the random forest is argmax_i (sum_t $[Y_t(x) = i]$), which is the class that the most decision trees predicted. Here $[Y_t(x) = i]$ is the indicator function, which is 1 if $Y_t(x) = i$, and 0 otherwise, and sum_t is summing over all trees in the forest. The power of random forests comes from their ability to model complex, non-linear decision boundaries, and their robustness to overfitting due to the aggregation of multiple diverse trees.

XGBoost. XGBoost[93] is a machine learning algorithm based on gradient boosting framework, which is used for both regression and classification problems. It is a very flexible model, as we can define our own objective functions and evaluation criteria. The goal of XGBoost is to minimize the following (Eq. 13) objective function:

$$\text{Objective function } = \sum L\ (y_i,\ )\ + \sum \Omega(f_i) \tag{13}$$

Here, L is the loss function that measures the difference between the actual output y_i and the predicted output $\widehat{y_i}$. Ω is the regularization term that prevents the model from overfitting, and f are the decision trees. The regularization term Ω in XGBoost's objective function is given by (Eq. 14):

$$\Omega(f) = \gamma T\ + 1/2\lambda \sum w^2 \tag{14}$$

Here, T is the number of leaves in the tree, w is the score on the leaves, γ is the parameter for controlling the complexity of the tree, and λ is the L2 regularization term on the scores to avoid overfitting. XGboost serves different objective functions, such as 'multi:softprob' and default 'binary:logistic' based on the nature of the data. The earlier objective function is dedicated to multiclass classification, while the latter one is for binary classifiers. For our multiclass classification study, the objective function was "softprob," which calculates the predicted probability. The multiclass softprob loss, also known as the softmax cross-entropy loss, is commonly used for multiclass classification problems. In the context of tomato leaf classification, let's assume we have "C" classes of tomato leaves (e.g., bacterial spot, TMV, TYLCV, etc.). The softprob loss is defined as: For a single training example with input features x and ground truth label y (represented as a one-hot vector), the softmax function (Eq. 15) computes the predicted probabilities for each class.

$$P_i\ =\ \exp\ (Z_i)\ /\ \text{sum}\ \left(\exp\ \left(Z_j\right)\right) \text{ for } i\ =\ 1 \text{ to } C \tag{15}$$

where Z_i represents the logits (pre-softmax scores) for class i. The softprob loss is then computed as the cross-entropy between the predicted probabilities and the ground truth label:

$L = -\text{sum}\ (y * \log(P))$, where "*" denotes element-wise multiplication and log is the natural logarithm.

To train a classification model using this loss, we can apply gradient descent or another optimization algorithm to minimize the average loss over a training dataset.

GaussianNB. Gaussian Naïve Bayes (GNB) is a probabilistic classification algorithm that assumes the features are normally distributed. For tomato leaf disease classification, we used GNB to predict the class of a tomato leaf based on its features. Let's denote the features of a tomato as $X = \{X_1, X_2,\ldots, X_n\}$, where X_i represents the value of the i-th feature. We want to classify the tomato leaf into one of the classes $C = \{C_1, C_2, \ldots, C_k\}$, where C_j represents the j-th class. The Gaussian Naïve Bayes classifier estimates the probability $P(C_j|X)$ using Bayes' theorem (Eq. 16):

Hybrid model	Best hyperparameters
CNN-XGBoost	gamma = 0.0638, learning_rate = 0.09995, max_depth = 10, subsample = 0.15, reg_alpha = 0.6, min_child_weight = 9, n_estimators = 100, colsample_bylevel = 0.6677, reg_lambda = 0.4
CNN-SVM	C = 0.00606777, gamma = 0.0019349, kernel = 'poly', coef0 = 0.93458485, degree = 2, probability = True
CNN-RF	criterion = 'gini', max_depth = None, min_samples_leaf = 6, n_estimators = 100, min_samples_split = 9, max_features = 'sqrt'
CNN-MLR	max_iter = 100, multi_class = 'multinomial', penalty = 'l2', solver = 'newton-cg', C = 0.00159
CNN-KNN	n_neighbors = 2, p = 1, weights = 'uniform', metric = 'euclidean'
CNN-GaussianNB	var_smoothing = 0.00834
CNN-Stacking	Base Estimator: CNN-KNN, CNN-SVM, CNN-RF Final Estimator: CNN- XGBoost The hyperparameters of base and final estimators are the same as those of standalone models' hyperparameters

Table 2. Bayesian optimized hyperparameters for ML classifiers.

$$P(C_j|X) = (P(C_j) * P(X|C_j))/P(X) \tag{16}$$

where $P(C_j|X)$ is the posterior probability of class C_j given the features X; $P(C_j)$ is the prior probability of class C_j; $P(X|C_j)$ is the likelihood of the features X given class C_j; and P(X) is the evidence probability, which acts as a normalizing constant. Now, let's assume that the features X_i are continuous and follow a Gaussian distribution within each class C_j. We can represent the likelihood $P(X|C_j)$ as a product of individual feature probabilities by Eq. (17). And the individual feature probabilities can be estimated by Eq. (18) assuming a Gaussian distribution.

$$P(X|C_j) = P(X_1|C_j) * P(X_2|C_j) * ... * P(X_j|C_j) \tag{17}$$

$$P(X_i|C_j) = (1/\text{sqrt}(2\pi * \sigma^2)) * \exp(-(X_i - \mu)^2/(2 * \sigma^2)) \tag{18}$$

where μ is the mean of feature X_i within class C_j, σ^2 is the variance of feature X_i within class C_j. To classify a tomato leaf, we calculate the posterior probability $P(C_j|X)$ for each class C_j and assign the tomato leaf to the class with the highest probability.

Stacking. Stacking is a type of EL method where two-level learners are present. The low-level, or level-0, learners and the high-level, or level-1, learner. The level-0 learners are also called base models or estimators, and the level-1 learner is called meta model or final estimator. In stacking, or stacked generalization, the predictions of each base model are stacked together and fed to the meta model. A meta-model is trained based on this output and provides the final prediction. Stacking provides flexibility in model selection and helps to obtain models with robust predictive capabilities. It diminishes the biases[94] of base estimators. In this study, we used KNN, SVM, and RF as base estimators and XGBoost as the final estimator. Such a combination of models yielded an outstanding stacked classifier with higher generalization capabilities. The hyperparameters of the base and final estimators are the same as those of standalone models' hyperparameters, as they were already optimized. Figure S7 shows the hybrid stacking classifier architecture.

The Bayesian optimized hyperparameters for all ML classifiers are shown in Table 2.

Implementation

The proposed hybrid models were trained using mini batch gradient descent of batch size 8, which implies that parameters were updated after a batch of 8 samples. Table S4 outlines the specifications of the environmental setup.

The following algorithm 2 provides a comprehensive depiction of the various phases encapsulated within the proposed methodology.

Algorithm 2. Steps of the Bayesian optimized hybrid model.

Start

Step 1: **Input** PlantVillage Dataset (kaggle.com) and necessary packages/libraries
Step 2: Perform preliminary **data analysis** and **visualization**
Step 3: Use ImageDataGenerator to **transform and split** the dataset
Step 4: Generate **augmented** images (for training)
ImageDataGenerator(rotation_range = 30, width_shift_range = 0.1, height_shift_range = 0.1, shear_range = 0.1, zoom_range = 0.2, horizontal_flip = True, fill_mode = 'reflect', brightness_range = (0.5, 1.5))
Step 5: Perform **data integration** (original training dataset and augmented training dataset)
Step 6: **Feature extraction framework:** define Bayesian search space for CNN architecture
Bayesian (Gaussian process-based surrogate) search space for CNN Using KerasTuner
def build_cnn_model(hp):
model = keras.Sequential()
input layer: 256, 256, 3
for loop:
Convolutional layers: No of conv2d layers: (min = 0, max = 7), Filters: [16, 32, 64], Kernel: [2, 3], Activation: ['relu', 'tanh'], Padding: 'same', Strides: 1, Kernel regularizer (l2): (min = $1e^{-5}$, max = $1e^{-2}$, sampling = 'log')
Batch normalization layer: Momentum: 0.99, Epsilon: 0.001
Max pooling 2D layer: Pool_size: [2, 3], Strides: None, Padding: 'same' if Pool_size > 2 else 'valid'
end loop
Flatten layer
for loop:
Dense layers: No of dense layers: (min = 1, max = 3), Units: [16, 32, 64], Activation: ['relu', 'tanh'], Kernel regularizer (l2): (min = $1e^{-5}$, max = $1e^{-2}$, sampling = 'log').
Batch normalization layer
Dropout layer: Dropout: (min = 0, max = 0.5, step = 0.1)
end loop
Output layer: 10 units and **softmax** activation.
Optimizer= hp.Choice ('optimizer', values = ['Adam', 'sgd'])
Compilation: model.compile (optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])
return model
keras_tuner = BayesianOptimization (build_cnn_model, objective = 'val_accuracy', max_trials=100, num_initial_points = 2, alpha = $1e^{-4}$)
keras_tuner.search (train_generator, validation_data = val_generator, epochs = 20, batch_size = 8, callbacks = EarlyStopping (patience = 3))
Step 7: **Fetch the best-performing Bayesian optimized hyperparameters**
Step 8: **Train the CNN** with the Bayesian optimized hyperparameters (from step 7)
Class = 10, inp_shape = (256,256,3), Conv2D hyperparameter order: filters, kernel_size, strides, padding, activation, kernel_regularizer (l2)
model_cnn=tf.keras.Sequential([
Convolutional layer: Conv2D (32, (2, 2), (1, 1), 'same', 'relu', 0.0001282)
Batch normalization layer: BatchNormalization (momentum = 0.99, epsilon = 0.001),
Max pooling layer: MaxPooling2D (pool_size = (3, 3), strides = (3, 3), padding = 'same'),
Convolutional layer: Conv2D (64, (2, 2), (1, 1), 'same', 'tanh', 0.0004593),
Batch normalization layer: BatchNormalization (momentum = 0.99, epsilon = 0.001),
Max pooling layer: MaxPooling2D (pool_size = (3, 3), strides = (3, 3), padding = 'same'),
Convolutional layer: Conv2D (16, (3, 3), (1, 1), 'same', 'relu', 0.01),
Batch normalization layer: BatchNormalization (momentum = 0.99, epsilon = 0.001),
Max pooling layer: MaxPooling2D (pool_size = (2, 2), strides = (2, 2), padding = 'valid'),
Convolutional layer: Conv2D (64, (2, 2), (1, 1), 'same', 'relu', 0.0011262)
Batch normalization layer: BatchNormalization (momentum = 0.99, epsilon = 0.001),
Max pooling layer: MaxPooling2D (pool_size = (3, 3), strides = (3, 3), padding = 'same'),
Flatten layer: layers.Flatten (),
Dense layer: layers.Dense (units = 32, activation = 'relu', kernel regularizer (l2) = $1.35e^{-05}$),
Batch normalization layer: BatchNormalization (momentum=0.99, epsilon=0.001),
Dropout layer: Dropout (rate = 0.3),
Classification layer: layers.Dense (class, activation = 'softmax')])
model_cnn.build (input_shape = inp_shape)
model_cnn.compile (loss = 'categorical_crossentropy', optimizer = Adam(), metrics = ['accuracy'])
model_cnn.fit(train_generator,validation_data =valid_generator,epochs=200,batch_size=8, callbacks=[early_stop, reduce_lr])

Step 9: **Evaluate** the **Bayesian optimized CNN** using accuracy, and loss
Step 10: **Extract features**
Remove last **4 layers** described in step 8.
Extract and reshape 1600 features from 'Flatten layer'
Step 11: **Perform feature selection:** employ **Boruta algorithm**
Import the dataset into a new Python file
Generate **shadow features**
Merge shadow features with original features
Train the tree-based ML classifier and **compute Z scores** for features
Identify the shadow feature with the maximum Z-score (**MZS**)
Label original features with Z-scores greater than MZS (**important features**)
Iterate the above steps to accept or reject the original features
Step 12: **Classification framework:** define **Bayesian search space** for **ML classifiers**
Bayesian search space for ML classifiers using Hyperopt (TPE algorithm)
algorithm = MLR, GNB, SVM, KNN, XGBoost, and RF classifiers
search_space = {'hyperparameters': define value ranges/values}
def objective (params): #Define the **objective function** to optimize
classifier = algorithm (**params)
score = cross_val_score (classifier, x_train, y_train, cv = 5, scoring="accuracy").mean()
return -score
trials = Trials ()
best_hyperparameter = fmin (fn = objective, space = search_space, algo = tpe.suggest, #Perform **TPE optimization**
max_evals = 100, trials=trials)
print("Best hyperparameters", best_hyperparameter) #**Collect** the best set of hyperparameters
Step 13: **Train ML classifiers** with Bayesian optimized hyperparameters
XGBoost: gamma=0.0638, learning_rate=0.09995, max_depth=10, subsample=0.15, reg_alpha=0.6, min_child_weight=9, n_estimators=100, colsample_bylevel= 0.6677, reg_lambda=0.4
SVM: C=0.00606777, gamma=0.0019349, kernel= 'poly', coef0=0.93458485, degree=2, probability=True
RF: criterion='gini', max_depth=None, min_samples_leaf=6, n_estimators=100, min_samples_split=9, max_features='sqrt'
MLR: max_iter=100, multi_class='multinomial', penalty='l2', solver='newton-cg', C=0.00159
KNN: n_neighbors=2, p=1, weights='uniform', metric= 'euclidean'
GNB: var_smoothing=0.00834
Stacking: Base Estimator-KNN, SVM, RF; Final Estimator-XGBoost. The hyperparameters of base and final estimators are the same as those of standalone models' hyperparameters.
Step 14: **Evaluate** the performance of the **CNN-ML classifiers** on the validation dataset
Test the classifiers on unseen images (testing dataset)
Record training and testing time and performance
Test the best-performing classifier on real-life images and record performance and testing time
Step 15: **Assess the efficacy of best-performing hybrid model with and without the Boruta algorithm**

End

Evaluation metrics

Machine learning models need to be evaluated properly to interpret their performance. We used a wide range of evaluation metrics to assess the performance of our proposed Bayesian optimized hybrid models, namely accuracy, precision, recall, f1-score, and Matthew's correlation coefficient (MCC). Accuracy is a performance metric that provides the overall performance of the model across all classes. It is the ratio of the sum of correct predictions to the total number of predictions. Precision is measured by the ratio of correctly classified positive samples to the total number of classified positive samples, either correctly or incorrectly. Recall is calculated by the ratio of correctly classified positive samples to the total number of actual positive (ground truth positive) samples. It measures the capability of the model to identify positive outcomes. Ideally, if precision increases, recall

Hybrid model	Training		Testing	
	F1-score	Accuracy	F1-score	Accuracy
CNN-GNB	90.914	90.876	89.863	89.728
CNN-RF	99.509	99.509	94.195	94.256
CNN-KNN	98.286	98.287	94.39	94.367
CNN-XGBoost	98.392	98.393	95.713	95.729
CNN-MLR	99.799	99.799	98.078	98.085
CNN-SVM	99.721	99.721	98.415	98.417
CNN-Stacking	99.777	99.777	98.525	98.527

Table 3. Average f1-score and overall accuracy on the training and testing datasets.

decreases. So, there is another metric named f1-score to tradeoff between precision and recall. It is a balanced metric commonly used to analyze the performance of a classification model. The f1-score is the harmonic mean of precision and recall. Precision, recall, and f1-score range between 0 and 1 (0–100%), where 1 means perfect and 0 means worst performance. The mathematical equations for accuracy, precision, recall, and f1-score are presented in Eqs. (19), (20), (21), and (22), respectively. Another quality measurement metric for multiclass classification is the MCC. For multiclass classification, it ranges between − 1 and + 1 or 0 and + 1, where + 1 means perfect classification. A high MCC value denotes that the model performs well in all positive (TP, FP) and negative (TN, FN) categories. Precision, recall, and f1-score are asymmetric measures, which means if we change the class labels, it affects the metrics values. MCC is a symmetric measure that determines the correlation between the true class and the predicted class. For 'K' classes, using the confusion matrix (C), we can define MCC using Eq. (23).

$$\text{Accuracy} = \frac{\sum \text{Correct Predictions}}{\text{Total number of predictions}} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \tag{19}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \tag{20}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{21}$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{22}$$

where TP = true positive (model correctly classified a positive item as positive), TN = true negative (model correctly classified a negative item as negative), FP = false positive (model incorrectly classified a negative item as positive), and FN = false negative (model incorrectly classified a positive item as negative).

$$\text{MCC} = \frac{\text{ab} - \sum_k^K P_k t_k}{\text{sqrt}\left[\left(b^2 - \sum_k^K P_k^2\right)\left(b^2 - \sum_k^K t_k^2\right)\right]} \tag{23}$$

where $a = \sum_k^K C_{kk}$, the number of samples correctly predicted (total), $b = \sum_i^K \sum_j^K C_{ij}$, the total number of samples, $t_k = \sum_i^K C_{ik}$, the number of times 'class k' actually occurred, and $P_k = \sum_i^K C_{ki}$, the number of times 'class k' predicted.

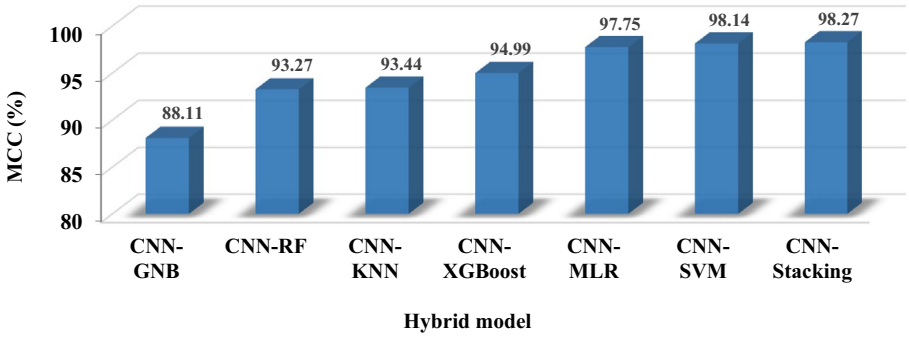


Fig. 4. MCC scores of the proposed hybrid models.

Results and discussion

Performance analysis of the proposed hybrid models

Accuracy and F1-score

In our proposed Bayesian optimized multiclass classification models, we employed appropriate metrics to showcase their robustness. Table 3 presents a comparative analysis of the seven hybrid models in terms of overall accuracy and f1-score. To illustrate the models' robustness and generalization, we included both training and testing metrics in Table 3.

Accuracy measures the model's performance when all classes are weighted equally, making it suitable for balanced datasets. However, we also included other metrics to assess the models' efficiency from various perspectives, given our dataset's near-balanced nature.

As shown in Table 3, the CNN-Stacking classifier achieves the highest accuracy (98.527%) among the seven hybrid models, followed by CNN-SVM (98.417%), CNN-MLR (98.085%), CNN-XGBoost (95.729%), CNN-KNN (94.367%), CNN-RF (94.256%), and CNN-GNB (89.728%). These accuracy scores demonstrate the remarkable performance of all seven hybrid models. Additionally, we calculated the average accuracy score of 95.587% for the hybrid models on the testing dataset. The average training and validation accuracies are 98.052% and 95.727%, respectively. Typically, training accuracy exceeds testing accuracy since the models are trained on the training data. The gaps between training and testing accuracy are minimal, at only 2.47%, and between validation and testing accuracy, they are just 0.14%. Therefore, based on both individual and average accuracy across the seven hybrid models, we confidently assert that they are not overfitted, performed exceptionally well on the test data, and demonstrate strong generalization capabilities.

Among all seven hybrid models, the CNN-Stacking model achieves the highest f1-score (weighted F1-score) of 98.525%. A high f1-score indicates a strong balance between precision and recall, reflecting the effectiveness of a classification model. On testing, the average f1-score of the seven hybrid models is 95.597%. Analyzing the f1-scores of these models in relation to training reveals minimal differences. We also computed the macro f1-score for each model. The average macro f1-score across the seven models is 94.863%, which closely matches the average weighted f1-score. Thus, all seven hybrid models demonstrate outstanding performance in terms of f1-score.

Precision

Figure S8 displays the average precision (weighted) of the training and testing datasets, evaluating the reliability of our proposed hybrid models in classifying true positive samples. From the line plots in Fig. S8, it is evident that testing precision closely mirrors training precision, indicating strong performance of our hybrid models on unseen data. The precision percentages for the CNN-GNB, CNN-RF, CNN-KNN, CNN-XGBoost, CNN-MLR, CNN-SVM, and CNN-Stacking models are 90.615, 94.35, 94.862, 95.797, 98.101, 98.43, and 98.533, respectively. The average precision on unseen data across these models is 95.813%. Additionally, we calculated an average macro precision of 94.937% on the test dataset. Therefore, from the precision analysis, it is evident that all seven models excel in correctly classifying the majority of positive classifications, with CNN-Stacking demonstrating the highest performance in classifying tomato leaf diseases.

Recall

To assess the recall performance of our hybrid models, we constructed a bar chart (Fig. S9) displaying recall values for both training and testing datasets. Recall is a crucial metric for evaluating a model's ability to correctly identify instances of a specific type of diseased tomato leaf among those that actually have it. For the seven hybrid models depicted in Fig. S9, recall values (%) on the testing dataset are 89.728, 94.256, 94.367, 95.729, 98.085, 98.417, and 98.527. The side-by-side bar plots of training and testing recalls strongly suggest that the hybrid models generalize well in terms of recall. Among these models, CNN-Stacking achieves the highest average recall of 98.527%.

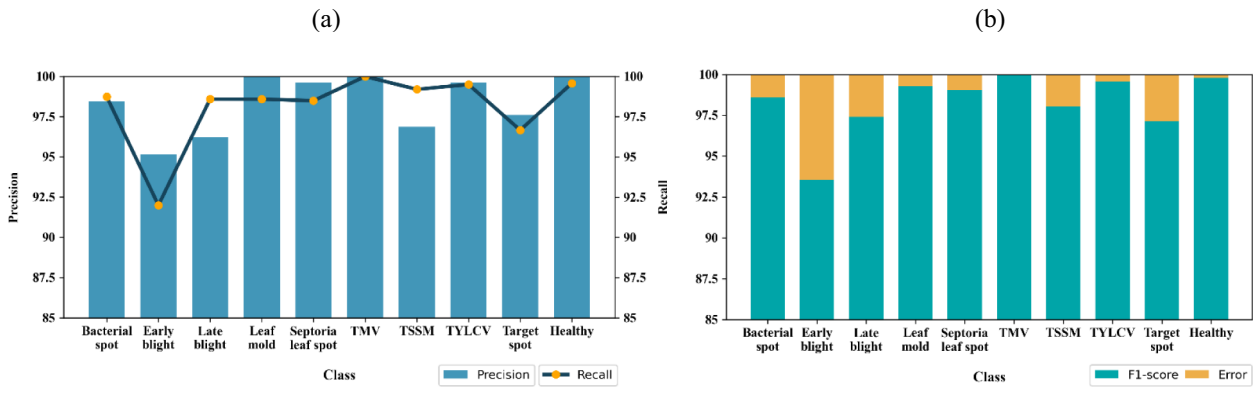


Fig. 5. Performance of the CNN-Stacking model (in %) using (**a**) Precision and Recall; (**b**) F1-score.

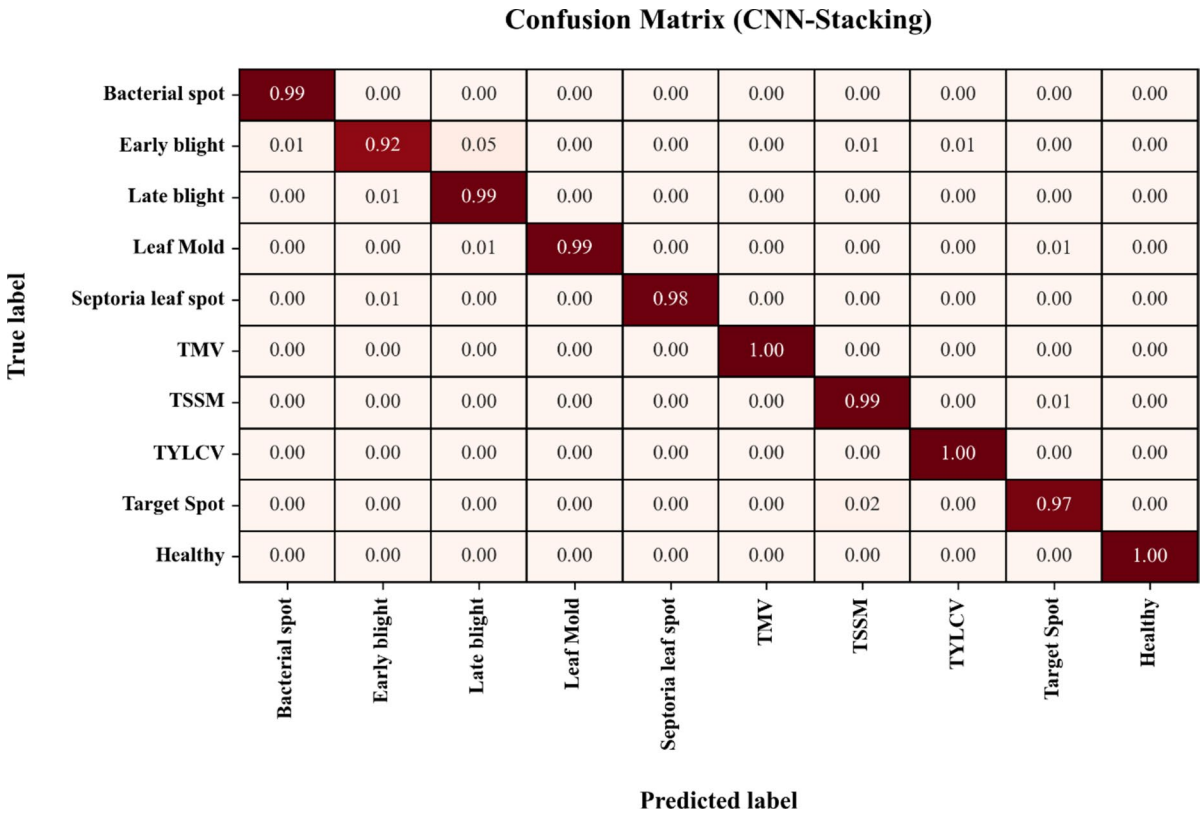


Fig. 6. Confusion matrix of the proposed Bayesian optimized CNN-Stacking model.

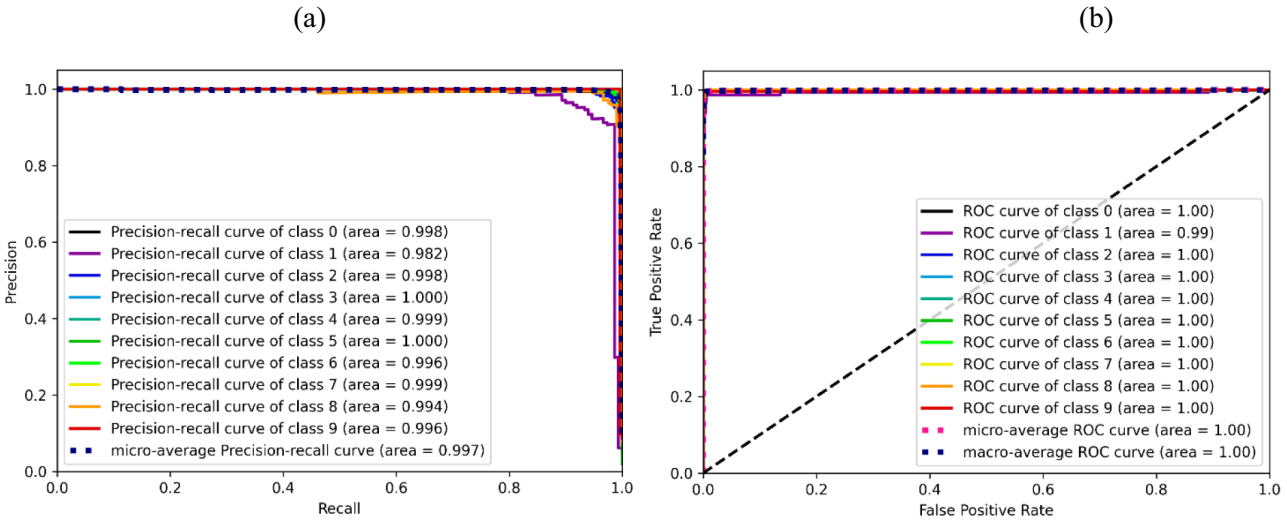


Fig. 7. (**a**) Precision-Recall curve; (**b**) ROC curve of the proposed CNN-Stacking hybrid model.

MCC

Figure 4 displays the MCC scores for the models utilized in our study. MCC, an advanced metric for classification models, is often overlooked in existing literature. The bar chart in Fig. 4 demonstrates that the MCC scores for most models exceed 90%. This indicates remarkably strong correlations between true labels and predicted labels in our proposed models. Among them, CNN-Stacking achieves the highest score, followed by CNN-SVM, CNN-MLR, CNN-XGBoost, CNN-KNN, CNN-RF, and CNN-GNB.

Class-specific performance assessment of the hybrid models

To assess the class-specific performance of our proposed hybrid models, we analyzed precision and recall measures across all classes in the dataset. Our study included ten classes: nine tomato leaf disease classes (Bacterial

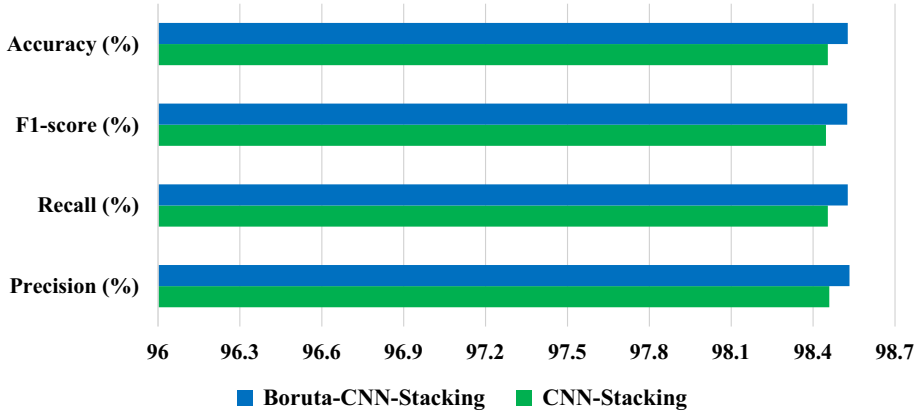


Fig. 8. Performance analysis of the CNN-Stacking model with and without the Boruta feature selector.

spot, Early blight, Late blight, Leaf mold, Septoria leaf spot, Target spot, TMV, TSSM, and TYLCV) and one healthy class. This analysis helps estimate how accurately our models classify ground-truth entities.

Class-specific precision and recall performances across seven hybrid models are visualized using boxplots in Figs. S10 and S11, respectively. A boxplot is a statistical visualization tool for comparing and analyzing distributions of data across different groups. In both figures, we plotted ten boxplots corresponding to the ten classes, considering the seven hybrid models.

From Fig. S10, the mean precision values across the seven models for each class are as follows: Bacterial spot (94.31%), Early blight (91.54%), Late blight (94.41%), Leaf mold (96.49%), Septoria leaf spot (96.49%), TMV (95.53%), TSSM (88.22%), TYLCV (99.21%), Target spot (93.61%), and Healthy (99.58%). Additionally, the boxplots provide insights into the minimum, maximum, median, and quartile precision values for each class.

Notably, for the TYLCV and Healthy classes, the precision values are consistently high, approaching 1 across all hybrid models, resulting in very compact boxplots. These classes exhibit the highest precision values among all classes across the hybrid models.

According to Fig. S11, the mean recalls (%) for the ten classes are 98.52, 82, 94.34, 94.37, 94.88, 100, 98, 96.51, 92.52, and 99.28, respectively. We observed that all models achieve the highest recall for the TMV class.

Based on these class-specific precision and recall tests, we confidently conclude that all seven optimized hybrid models perform extremely well, demonstrating remarkably high precision and recall rates across the ten classes. This underscores the outstanding generalization capabilities of our models.

Classification result of the CNN-stacking model

Upon comparing the overall performance of the proposed seven hybrid models, we observed that the Bayesian optimized CNN-Stacking model exhibits the highest performance across all evaluation metrics. Therefore, we conducted additional analyses using various measures to underscore its reliability in classifying tomato leaf diseases.

In Fig. 5a, we illustrate the classification efficacy of the CNN-Stacking model through precision and recall scores. The model achieves precision scores (%) of 98.43, 95.17, 96.23, 100, 99.62, 100, 96.88, 99.63, 97.60, and 100 for the Bacterial spot, Early blight, Late blight, Leaf mold, Septoria leaf spot, TMV, TSSM, TYLCV, Target spot, and Healthy classes, respectively. These precision scores clearly demonstrate the model's ability to distinguish and accurately classify different classes of tomato leaf diseases.

Additionally, the line plot in Fig. 5a depicts the trends in recall scores across these classes. The CNN-Stacking model achieves recall rates (%) of 98.74, 92, 98.60, 98.59, 98.49, 100, 99.20, 99.50, 96.67, and 99.58, respectively, showcasing its robustness with remarkably high recall rates. Consequently, CNN-Stacking successfully identifies all types of diseased tomato leaves from the ground truth.

Considering the error rates of the ten classes, we evaluated the success rate of our CNN-Stacking model's classification using the f1-score in Fig. 5b. With an average and minimum error of 1.76% and 0%, respectively, the model shows appealing f1-scores in all ten classes.

Confusion matrix, PR curves, and ROC curves

To further highlight the CNN-Stacking model's performance, we constructed a 10×10 confusion matrix, precision-recall curves (PR curves), and Receiver Operating Characteristic (ROC) curves based on the ten classes in the test dataset, shown in Figs. 6, 7a and b, respectively. A confusion matrix compares ground-truth classes with model predictions, offering an overview of overall model performance. The ROC curve illustrates the classifier's performance by plotting the true positive rate against the false positive rate at various thresholds, while the precision-recall curve demonstrates the trade-off between precision and recall metrics.

In Fig. 6, the confusion matrix highlights the CNN-Stacking model's exceptional discrimination between classes, achieving 100% correct predictions in 3 classes and 99% correct predictions in 4 classes, underscoring its strong generalization capabilities. Moreover, Fig. 7a and b show ROC and PR curves where the AUC scores are close to 1, indicating excellent classifier performance.

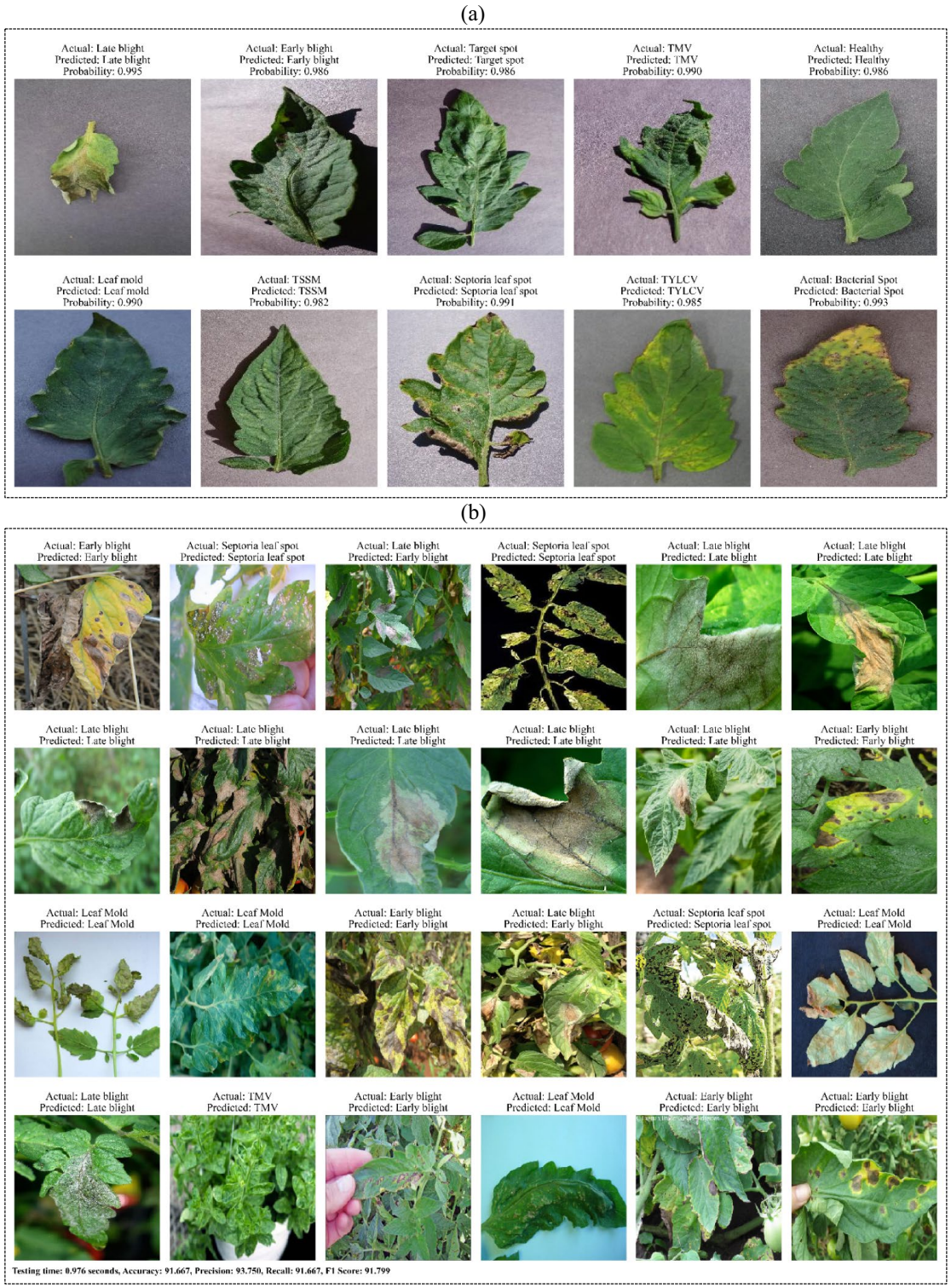


Fig. 9. Classification results of (**a**) Controlled dataset, Source: PlantVillage Dataset (kaggle.com); (**b**) Real-life dataset, Source: PlantDoc Dataset.

In summary, the customized hybrid CNN-Stacking model, combining deep learning (CNN) and machine learning (KNN, SVM, RF, XGBoost), excels in classifying tomato leaf diseases, as evidenced by its outstanding performance across multiple evaluation metrics and visualization analyses.

Performance analysis of the Boruta feature selector

Figure 8 compares the performance of the CNN-Stacking model with and without the Boruta feature selector. Both approaches demonstrate comparable performance across evaluation criteria. These comparative results

highlight the effectiveness and reliability of the primary model's feature extraction methods, indicating its ability to capture important features and maintain consistent classification accuracy.

Moreover, this suggests that CNN-Stacking enhances model flexibility by capturing complex relationships, thereby bolstering its robustness and versatility in diagnosing and managing tomato leaf diseases. Thus, the primary CNN-Stacking model not only maintains strong performance but also offers cost-effective, streamlined, and quicker solutions for addressing challenges related to tomato leaf diseases.

Model testing

The classification results of the CNN-Stacking hybrid model on the test dataset are depicted in Fig. 9a. As shown in the figure, our model correctly classified all ten classes with high probability scores.

Furthermore, we evaluated our proposed model using 24 samples from a real-world dataset (the PlantDoc Dataset) and the classification results are illustrated in Fig. 9b. The model correctly identified 22 of 24 samples (i.e., 91.67% accuracy) with a testing time of only 0.976 s. Furthermore, the model achieves a remarkable precision of 93.75%, recall of 91.67%, and f1-score of 91.80% on these real-life images. This demonstrates the model's ability to handle real-world scenarios effectively and produce reliable classifications. Although the accuracy dropped by 6.96% compared to the PlantVillage dataset, this can be attributed to the model's exclusive training on the controlled dataset. Accuracy can also drop abruptly when the sample size is small. After all, the model's outstanding accuracy, despite being trained on any real-life dataset, highlights its potential for successful application to diverse and complex images, as well as its considerable future usage.

Complexity analysis of the proposed CNN-stacking classifier

The algorithmic complexity analysis of the proposed CNN-Stacking Classifier can be delineated in terms of both time complexity and space complexity.

Time complexity

The number of operations executed during the training and prediction phases determines the time complexity of a Bayesian hybridized CNN model for tomato leaf disease classification. The CNN layers are the most significant contributors to time complexity.

CNN Time Complexity: The temporal complexity of a 2D Convolutional Neural Network (CNN) layer is contingent upon the spatial dimensions of the input feature maps, the quantity of filters, and the dimensions of the convolutional filters. The time complexity of the CNN layer can be estimated as $O(\sum_{l=1}^{L} M_l^2 * K_l^2 * F_l * F_{l-1})$ [95], where M_l denotes the spatial dimensions of the feature map, K_l signifies the size of filters, F_l is the number of filters, and L represents the number of layers.

Stacking Classifier Time Complexity: The calculation of time complexity for a stacking classifier involves analyzing the computational efficiency of the algorithm in terms of the input size. Stacking classifiers are ensemble learning techniques that combine the predictions of multiple base classifiers to improve overall performance. The time complexity of a stacking classifier depends on several factors, including the complexity of the individual base classifiers, the number of layers in the stack, and the size of the training dataset. In general, if we denote n as the number of instances in the training set and m as the number of features, the time complexity of training a single base classifier is often expressed as O(f(n, m)), where f is a function representing the computational cost. If there are k base classifiers in the stack, the overall time complexity may be influenced by the stacking process, which involves training each base classifier, generating predictions, and training a meta-classifier on top of them.

The exact time complexity can vary depending on the specific algorithms used for the base and meta-classifiers. It is common to consider the training time of each base classifier and the meta-classifier separately and then sum them up to get an estimate of the overall time complexity. Additionally, factors such as hyperparameter tuning and cross-validation can further impact the computational cost.

Overall Time Complexity: The overall time complexity of our best-performing model, the CNN-Stacking model, is determined by the combined complexities of the Stacking classifier and the CNN model, expressed as: $O(f(n, m)) + O(\sum_{l=1}^{L} M_l^2 * K_l^2 * F_l * F_{l-1})$.

Reference	Approach	Method	Dataset	Precision (%)	Recall (%)	F1-score (%)	Accuracy (%)
[26]	DL	CNN	Plant Village	–	–	–	91.2
[27]	Hybrid	CNN-RNN		–	–	–	81.75
[40]	Hybrid	ADC + LR		–	–	–	96.6
[45]	DL	LeNet		90–95	90–95	90–95	90–95
[52]	DL	Modified AlexNet		98	95	97	96
[58]	Hybrid	ResNet50 + SeNet		96.77	96.81	96.79	96.81
[59]	Hybrid	C-GAN-DenseNet121		97	97	97	97.11
Proposed	BO Hybrid	CNN-Stacking		98.533	98.527	98.525	98.527

Table 4. Comparison of the proposed approach with the latest approaches (tomato leaf 10 classes). N.B. BO for Bayesian optimized.

Reference	Approach	Method	Plant leaf (class)	Dataset	Precision (%)	Recall (%)	F1-score (%)	Accuracy (%)
[41]	BO Hybrid	ADSNN-BO	Rice (4)	–	92.6	87.4	89.6	94.65
[42]	BO DL	AlexNet ResNet50 SqueezeNet	Maize (4)	PV	96 95 95	95 96 96	95 96 96	97 97 97
[50]	DL	DenseNet-201	Grape (4)	PV	98.31	98.27	98.28	98.27
[54]	DL	YOLOv7	Tea leaf (5)	Own	96.7	96.4	96.5	97.3
[55]	ML	DLQP + SVM	Potato (3)	PV	–	–	–	96.2
	ML	DLQP + SVM	Apple (3)	PV	–	–	–	97.8
[60]	Hybrid	CNN + Autoencoders	Tomato + Potato + Maize (6)	PV	91	91	91	86.78
[61]	DL	Inception-v3	Maize (4)	Own	95.94	95.96	95.94	95.99
[62]	Hybrid	CAE + CNN	Peach (2)	PV	98	98.72	98.36	98.38
[66]	Hybrid	VGG-19 + LR	Potato (3)	PV	97.8	97.8	97.8	97.8
[68]	Hybrid	ResNet-50 + SVM	14 types (39)	PV	–	97.50	97.13	98.01
[69]	DL	Custom CNN	Tea leaf (4)	Own	–	–	–	96.65
Proposed	BO Hybrid	CNN-Stacking	Tomato (10)	PV	98.533	98.527	98.525	98.527

Table 5. Comparison of the suggested approach with recently established models for various crops. N.B. PV for PlantVillage, BO for Bayesian optimized.

Space complexity

CNN Space Complexity: The space complexity of a Bayesian hybridized CNN model refers to the amount of memory required to store its parameters, including weights, biases, and other trainable components, as well as intermediate results during computation. In the context of a Bayesian hybridized CNN model, the space complexity is influenced by factors such as the number of layers, the size of each layer, the number of filters in convolutional layers, and the dimensions of the kernel. The space complexity of a CNN model is determined by the formula $O(\sum_{l=1}^{L} K_l^2 * F_l * F_{l-1})$[95], where K_l signifies the size of filters, F_l is the number of filters, and L represents the number of layers.

Stacking Classifier Space Complexity: The equation for the space complexity of a stacking classifier involves considering the memory requirements associated with storing the individual base classifiers, as well as any additional data structures needed for the stacking process. Let's denote the number of base classifiers as *n*. The space complexity can be expressed as follows:

$$O(n) + O(m)$$

Here, $O(n)$ represents the space required to store the parameters and structures of each base classifier, and $O(m)$ represents any additional space needed for the stacking mechanism.

Overall Time Complexity: The total space complexity of the Stacking-CNN model equals the combined space complexities of its Stacking layer and CNN layer, represented as: $O(n) + O(m) + O(\sum_{l=1}^{L} K_l^2 * F_l * F_{l-1})$.

Comparative evaluation with previous works

In this section, we have presented the superiority of the proposed Bayesian optimized CNN-Stacking hybrid framework compared to the established DL, ML, and hybrid methods in the literature for the classification of tomato leaf diseases. The comparison focuses on evaluating the performance of the suggested strategy against several potent deep learning and transfer learning techniques, including LeNet, CNN, AlexNet, RNN, ResNet, SeNet, C-GAN, DenseNet, NASNetMobile, VGG16 net, MobileNet V2, RCNN, YOLO, CAE, Inception-v3, etc., as well as machine learning techniques, including DT, RF, SVM, KNN, and MLR. Besides, in comparison, we considered several factors, i.e., the type of methodology employed, the source of the dataset, and the size of the classes.

Proposed CNN-stacking vs. existing methods: 10-class tomato leaf disease classification

Considering four comparison metrics (precision, recall, f1-score, and accuracy), the existing methods for the classification of tomato leaf diseases (ten classes) are presented in Table 4. Using the PlantVillage dataset, the average (%) precision, recall, f1-score, and accuracy of the state-of-the-art models are 96.69, 95.95, 96.45, and 93.50, respectively, while our model scores 98.533, 98.527,98.525, and 98.527, respectively, on the same metrics. So, we have achieved a performance improvement of 1.84% for precision, 2.57% for recall, 2.08% for f1-score, and 5.03% for accuracy. This demonstrates that our proposed CNN-Stacking outperforms the state-of-the-art models. Besides, most of the previous methods[26,27,40,45,52,58,59] shown in Table 4, employed only deep learning techniques, both for feature extraction and classification, which are computationally expensive and require high training time.

Proposed CNN-stacking hybrid method vs. existing classification methods (< 10 classes)

As many studies in the existing literature considered less than ten classes of tomato leaf diseases, we separately analyzed these studies in Table S5. This table shows that the class ranges from 4 to 9. Among the seven stated previous studies (Table S5), the minimum accuracy is 76.1%[1] and the maximum accuracy is 97.49%[46]. The average

Reference	Model	Model parameters in million (approx.)	Reference	Model	Model parameters in million (approx.)
[19]	GoogLeNet	5	63	EfficientNetB3	11.19
	AlexNet	60		LDDTA	0.18
[54]	YOLOv7	36.9	66	VGG-19 + LR	143
[60]	CNN + Autoencoders	3.27	67	ResNet-152	60
[63]	Xception	21.40		Inception-v3	23
	Inception ResnetV2	54.74	68	ResNet-50 + SVM	25
	Densenet201	18.82	69	Custom CNN	0.25
	EfficientNetB4	18.14	Proposed	CNN-Stacking	0.023

Table 6. Comparison of the proposed model's training parameters with state-of-the-art models.

(%) precision, recall, f1-score, and accuracy of the state-of-the-art models[1,6,43,46,47,53,55,57] of various class sizes are 83.78, 86.97, 84.86, and 93.73, respectively. Our robust hybrid model exceeds this performance by an increment of 14.76%, 11.56%, 13.67%, and 4.79% for precision, recall, f1-score, and accuracy, respectively. Therefore, with respect to various class sizes as well, our stated study performs admirably.

Proposed CNN-stacking vs. existing methods for other types of leaf diseases
Finally, we examined the performance our proposed method with the models available in the literature for various types of plant leaves. Table 5 depicts the latest state-of-the-art studies on Rice, Maize, Grape, Tea leaf, Potato, Apple, Peach, and other types of leaf diseases. Most of these studies used PlantVillage dataset with different class sizes. However, the methodologies used in these studies[41,42,50,54,55,60–62,66,68,69] are complex, high-resource-demanding, laborious, and time-consuming. In comparison to the latest studies' average precision (95.64%), recall (95.46%), f1-score (95.60%), and accuracy (96.35%) values, our study demonstrates higher average precision (98.533%), recall (98.527%), f1-score (98.525%), and accuracy (98.527%). The proposed study obtained a performance gain of 2.90% for precision, 3.07% for recall, 2.92% for f1-score, and 2.18% for accuracy. Therefore, considering the performance of different types of plant leaf disease classification models, our automated hybrid system possesses a noteworthy advancement.

Computational complexity analysis: proposed vs. existing methods
This section thoroughly examines the proposed method's computational complexity compared to existing state-of-the-art methods. The section is structured considering two perspectives that are addressed below.

Performance and speed tradeoff
Computational complexity is typically characterized by computation time (i.e., speed) and memory usage. It is essential to consider model performance while investigating computational complexity to gain a solid understanding of the tradeoff between speed and performance. Hence, in Table S6, we have provided a comparison of our proposed Bayesian optimized CNN-Stacking method's performance and speed tradeoff with state-of-the-art models. Based on the data presented in Table S6, the ResNet-34 based Faster-RCNN model[64] requires a test time of 0.23 s to gain an accuracy of 99.97%, while models such as Improved Yolo V3[48], and LDDTA[63] exhibit varying processing times of 0.02 s (92.39% accuracy) and 0.022 s (97.5% accuracy), respectively. Additionally, Mahadevan et al.[65] reported a Generative Adversarial Network (GAN) based DSGAN^2-IAPO model that exhibits a time complexity of 0.098 s to maintain an accuracy score of 98.41%. In comparison to these models performance and speed tradeoff, the proposed approach demonstrates a substantial improvement. Our stated CNN-Stacking model requires only 0.174 s of testing time for analyzing a single disease image. This time duration (i.e., operations end time − operations start time) was determined by averaging the testing times of five randomly chosen disease images and encompasses critical processes such as importing, feature extraction, feature selection, and classification. While other models demand less time, the performance and speed ratio reveal outstanding findings for the proposed model. In Table S6, we have reserved a column to show the percent performance gain per second by the CNN-Stacking model by applying the following mathematical equation: $(\text{CNN-Stacking}_{\text{accuracy}} - \text{State-of-the-art model}_{\text{accuracy}}) \div (\text{CNN-Stacking}_{\text{time}} - \text{State-of-the-art model}_{\text{time}})$. This comparison clearly illustrates the trade-offs between speed and performance between the state-of-the-art models and the proposed model. According to the finding the suggested strategy outperforms the state-of-the-art techniques by a significant margin, ranging from 1.56 units to 39.95 units. Furthermore, the subtle increase in time complexity can be justified by one key factor: the system configuration. The systems employed for the current models, as indicated in Table S6, are either high-end or mediocrely configured, which greatly contributes to reducing time complexity. Nevertheless, even with a low-end machine, the significant improvement in performance-time evaluation indicates that the proposed architecture is well-constructed in comparison to existing leaf disease classification models.

Model parameters
The measurement of time (shown in Table S6) is significantly influenced by the hardware configurations in use, such as CPU, GPU, and RAM. Moreover, it is important to analyze not only the speed of processing but also the impact of model size on efficiency[63]. Therefore, we have considered a fundamental approach that involves

assessing the computational complexity of the model based on the number of training or model parameters. It is evident that as the number of training parameters increases, the models become more complex and exhibit higher time and space complexity. Through an extensive literature review, we have compiled a list of training parameters for some recent and prominent leaf disease detection approaches, which are compared with our proposed model in Table 6.

In our feature extraction framework, the custom CNN model comprises 74,458 total parameters. After removing the classification layers, 22,768 total parameters remain, out of which 22,416 are trainable and 352 are non-trainable. In comparison to the number of training parameters (expressed in million) in the state-of-the-art models[19,54,60,63,66–69] (e.g., GoogLeNet: 5, AlexNet: 60, YOLOv7: 36.9, CNN + Autoencoders: 3.27, Xception: 21.40, Inception ResnetV2: 54.74, Densenet201: 18.82, EfficientNetB4: 18.14, EfficientNetB3: 11.19, LDDTA: 0.18, VGG-19 + LR: 143, ResNet-152: 60, Inception-v3: 23, ResNet-50 + SVM: 25, and Custom CNN: 0.25), the parameter count of our model is drastically low (0.023 million) while still maintaining remarkable performance, attributed to its streamlined design. This also justifies that, irrespective of hardware configurations, our proposed hybrid model is extremely lightweight and requires minimal training and prediction time.

Based on the above-mentioned extensive comparative evaluation, we can assert that our proposed Bayesian optimized CNN-Stacking model demonstrates significant improvements over the state-of-the-art leaf disease classification methodologies, establishing a new benchmark in the field.

Limitations of the study

The effectiveness of the proposed approach in addressing different plant leaf diseases depends on the specific characteristics of the agricultural setting, the quality of the data, and other related factors. Here are the potential challenges for the implementation of the proposed approach to various types of diseases.

1. The model may struggle to generalize effectively across various types of leaf diseases, as it was exclusively trained on tomato leaf diseases. This limitation can be tackled by periodically updating the model with data on new diseases.
2. Plant diseases often target specific parts of the plant, such as leaves, stems, or roots. Therefore, the model needs improvement to accurately locate the affected regions.
3. Deep hybrid learning models incorporating Bayesian techniques can be computationally intensive, especially for complex models. However, with advancements in cloud computing and hardware acceleration, the computational demands of deep learning models have become more manageable.

Conclusions

Our proposed study on tomato leaf disease classification using Bayesian optimized deep hybrid learning sheds light on the promising applications of artificial intelligence in agriculture, especially in disease detection and plant health management. One of the most significant features of this research lies in its innovative approach to disease detection in tomato leaves. By leveraging Bayesian optimization, the study optimally fine-tuned deep hybrid learning models, thereby enhancing their accuracy and efficiency. This not only streamlines the classification process but also contributes to early disease detection, ultimately minimizing crop loss. The present study developed seven Bayesian optimized deep hybrid learning models using the integration of deep learning and machine learning for the classification of ten types of tomato leaves from the research-oriented PlantVillage dataset. Amongst them, the Bayesian optimized CNN-Stacking model emerged as the top-performing hybrid model with accuracy, precision, recall, f1-score, and mcc scores of 98.527%, 98.533%, 98.527%, 98.525%, and 98.268%, respectively, on an unseen dataset. This performance reflects the strong generalization potential of the model. Through a comprehensive comparative examination, we have demonstrated that our newly devised, optimized hybrid model outperforms the existing classification models outlined in the literature across all classification metrics. Our approach also excels in cost-effectiveness, a more lightweight structure, and enhanced time efficiency, as justified by the Boruta algorithm. We hope this research paves the way for more efficient, accurate, and sustainable agricultural practices in the future, with the potential to impact food security and agricultural sustainability positively. In the future, we will extend this methodology to a broader range of plants and develop a smartphone app to offer farmers a real-time disease diagnosis tool and management tactics, allowing them to take the necessary steps at a low cost and on time.

Future work

Future research endeavors should explore the application of other hierarchical deep learning models for disease classification tasks. Moreover, efforts should be directed towards enhancing the model's capability to localize plant diseases by integrating advanced techniques such as object detection or segmentation algorithms. Additionally, future investigations may incorporate other feature filtering algorithms and classification methods to further refine disease identification and classification processes.

Data availability

Data will be available upon reasonable request to corresponding authors.

Code availability

Code will be available upon reasonable request on corresponding authors.

Received: 22 June 2023; Accepted: 5 September 2024
Published online: 14 September 2024

References

1. Batool, A. et al. Classification and identification of tomato leaf disease using Deep Neural Network. In *International Conference on Engineering and Emerging Technologies (ICEET)* (IEEE, 2020).
2. World Food and Agriculture–Statistical Yearbook 2021. FAO eBooks https://doi.org/10.4060/cb4477en (2021).
3. Panno, S. et al. A review of the most common and economically important diseases that undermine the cultivation of tomato crop in the Mediterranean Basin. *Agronomy* **11**, 2188. https://doi.org/10.3390/agronomy11112188 (2021).
4. Sardogan, M., Tuncer, A. & Ozen, Y. Plant Leaf Disease Detection and classification based on CNN with LVQ algorithm. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)* https://doi.org/10.1109/ubmk.2018.8566635(2018).
5. Savary, S., Ficke, A., Aubertot, J.-N. & Hollier, C. Crop losses due to diseases and their implications for global food production losses and Food Security. *Food Security* **4**, 519–537. https://doi.org/10.1007/s12571-012-0200-5 (2012).
6. Basavaiah, J. & Arlene Anthony, A. Tomato leaf disease classification using multiple feature extraction techniques. *Wirel. Pers. Commun.* **115**, 633–651. https://doi.org/10.1007/s11277-020-07590-x (2020).
7. Cengil, E. & Çınar, A. Hybrid convolutional neural network based classification of bacterial, viral, and fungal diseases on tomato leaf images. *Concurr. Comput. Pract. Exp.* https://doi.org/10.1002/cpe.6617 (2021).
8. Leite, G. L. & Fialho, A. Protection of tomatoes using bagging technology and its role in IPM of arthropod pests. *Sustain. Manag. Arthropod Pests Tomato* https://doi.org/10.1016/b978-0-12-802441-6.00014-0 (2018).
9. Harakannanavar, S. S., Shridhar, H., Premananda, R., Jambukesh, H. J. & Prashanth, C. R. Integrated Analysis of Tomato Plant leaf disease disorder using improved Machine Learning approach. *J. Posit. Sch. Psychol.* **6**, 1288–1297 (2022).
10. Bock, C. H., Poole, G. H., Parker, P. E. & Gottwald, T. R. Plant disease severity estimated visually, by digital photography and Image Analysis, and by Hyperspectral Imaging. *Crit. Rev. Plant Sci.* **29**, 59–107 (2010).
11. Islam, M., Anh Dinh, W. K. & Bhowmik, P. Detection of potato diseases using image segmentation and multiclass support vector machine. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)* https://doi.org/10.1109/ccece.2017.7946594 (2017).
12. Bock, C. H., Parker, P. E., Cook, A. Z. & Gottwald, T. R. Visual rating and the use of image analysis for assessing different symptoms of citrus canker on grapefruit leaves. *Plant Dis.* **92**, 530–541 (2008).
13. Wolfenson, K. D. M. Coping with the food and agriculture challenge: smallholders' agenda. Preprint at https://www.fao.org/fileadmin/templates/nr/sustainability_pathways/docs/Coping_with_food_and_agriculture_challenge__Smallholder_s_agenda_Final.pdf (2013).
14. Azlah, M. A., Chua, L. S., Rahmad, F. R., Abdullah, F. I. & Wan Alwi, S. R. Review on techniques for Plant Leaf Classification and recognition. *Computers* **8**, 77. https://doi.org/10.3390/computers8040077 (2019).
15. Bari, B. S. et al. A real-time approach of diagnosing rice leaf disease using deep learning-based faster R-CNN framework. *PeerJ Comput. Sci.* **7**, e432. https://doi.org/10.7717/peerj-cs.432 (2021).
16. Islam, M. N., Ahmed, F., Ahammed, M. T., Rashid, M. & Bari, B. S. Rice disease identification through leaf image and IOT based Smart Rice Field Monitoring System. In *Enabling Industry 4.0 through Advances in Mechatronics* 529–539 https://doi.org/10.1007/978-981-19-2095-0_45 (2022).
17. Gupta, O., Das, A. J., Hellerstein, J. & Raskar, R. Machine learning approaches for large scale classification of produce. *Sci. Rep.* https://doi.org/10.1038/s41598-018-23394-3 (2018).
18. Paudel, D. et al. Machine learning for large-scale crop yield forecasting. *Agric. Syst.* **187**, 103016. https://doi.org/10.1016/j.agsy.2020.103016 (2021).
19. Mohanty, S. P., Hughes, D. P. & Salathé, M. Using deep learning for image-based plant disease detection. *Front. Plant Sci.* https://doi.org/10.3389/fpls.2016.01419 (2016).
20. Ferentinos, K. P. Deep learning models for plant disease detection and diagnosis. *Comput. Electron. Agric.* **145**, 311–318. https://doi.org/10.1016/j.compag.2018.01.009 (2018).
21. Mokhtar, U. et al. SVM-based detection of Tomato leaves diseases. *Adv. Intell. Syst. Comput.* https://doi.org/10.1007/978-3-319-11310-4_55 (2015).
22. Mohanty, R., Wankhede, P., Singh, D. & Vakhare, P. Tomato plant leaves disease detection using machine learning. In *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)* https://doi.org/10.1109/icaaic53929.2022.9793302 (2022).
23. Johannes, A. et al. Automatic plant disease diagnosis using mobile capture devices, applied on a wheat use case. *Comput. Electron. Agric.* **138**, 200–209. https://doi.org/10.1016/j.compag.2017.04.013 (2017).
24. Kantale, P. & Thakare, S. Pomegranate disease classification using Ada-Boost ensemble algorithm. *Int. J. Eng. Res. Technol* **9**, 612–620 (2020).
25. Sujatha, R., Chatterjee, J. M., Jhanjhi, N. & Brohi, S. N. Performance of Deep Learning vs machine learning in plant leaf disease detection. *Microprocess. Microsyst.* **80**, 103615. https://doi.org/10.1016/j.micpro.2020.103615 (2021).
26. Agarwal, M., Singh, A., Arjaria, S., Sinha, A. & Gupta, S. TOLED: Tomato Leaf Disease Detection using convolution neural network. *Procedia Comput. Sci.* **167**, 293–301. https://doi.org/10.1016/j.procs.2020.03.225 (2020).
27. David, H. E., Ramalakshmi, K., Venkatesan, R. & Hemalatha, G. Tomato leaf disease detection using hybrid CNN-RNN model. *Adv. Parallel Comput.* https://doi.org/10.3233/apc210108 (2021).
28. Roska, T. & Chua, L. O. The CNN universal machine: An analogic array computer. *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.* **40**, 163–173. https://doi.org/10.1109/82.222815 (1993).
29. Guerrero-Ibañez, A. & Reyes-Muñoz, A. Monitoring tomato leaf disease through convolutional neural networks. *Electronics* **12**, 229. https://doi.org/10.3390/electronics12010229 (2023).
30. Zaremba, W., Sutskever, I. & Vinyals O. Recurrent Neural Network Regularization. Preprint at https://arxiv.org/abs/1409.2329v5 (2014).
31. Hasan, R. I., Yusuf, S. M. & Alzubaidi, L. Review of the state of the art of Deep Learning for plant diseases: A broad analysis and discussion. *Plants* **9**, 1302. https://doi.org/10.3390/plants9101302 (2020).
32. Yuan, Z.-W. & Zhang, J. Feature extraction and image retrieval based on Alexnet. *SPIE Proc.* https://doi.org/10.1117/12.2243849 (2016).
33. Thenmozhi, K. & Srinivasulu Reddy, U. Crop pest classification based on deep convolutional neural network and transfer learning. *Comput. Electron. Agric.* **164**, 104906. https://doi.org/10.1016/j.compag.2019.104906 (2019).
34. Vedaldi, A. & Zisserman, A. VGG Convolutional Neural Networks Practical. Preprint at https://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html (2017).
35. Kaya, A. et al. Analysis of Transfer Learning for deep neural network based plant classification models. *Comput. Electron. Agric.* **158**, 20–29. https://doi.org/10.1016/j.compag.2019.01.041 (2019).
36. Wang, D. et al. A review of deep learning in Multiscale Agricultural Sensing. *Remote Sens.* **14**, 559. https://doi.org/10.3390/rs14030559 (2022).

37. Hong, H., Lin, J. & Huang, F. Tomato disease detection and classification by Deep Learning. In *2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)* (https://doi.org/10.1109/icbaie49996.2020.00012 (2020).
38. Kamilaris, A. & Prenafeta-Boldú, F. X. Deep learning in agriculture: A survey. *Comput. Electron. Agric.* **147**, 70–90. https://doi.org/10.1016/j.compag.2018.02.016 (2018).
39. Li, L., Zhang, S. & Wang, B. Plant disease detection and classification by Deep Learning—A review. *IEEE Access* **9**, 56683–56698. https://doi.org/10.1109/access.2021.3069646 (2021).
40. Islam, M. S. et al. Multimodal hybrid deep learning approach to detect tomato leaf disease using attention based dilated convolution feature extractor with logistic regression classification. *Sensors* **22**, 6079. https://doi.org/10.3390/s22166079 (2022).
41. Wang, Y., Wang, H. & Peng, Z. Rice diseases detection and classification using attention based neural network and Bayesian optimization. *Expert Syst. Appl.* **178**, 114770 (2021).
42. Da Rocha, E. L., Rodrigues, L. & Mari, J. F. Maize leaf disease classification using convolutional neural networks and hyperparameter optimization. In *Anais Do XVI Workshop De Visão Computacional (WVC 2020)* https://doi.org/10.5753/Wvc.2020.13489 (2020).
43. Restrepo-Arias, J. F., Branch-Bedoya, J. W. & Awad, G. Plant disease detection strategy based on image texture and Bayesian optimization with small neural networks. *Agriculture* **12**, 1964 (2022).
44. Seno, A., Miyagusuku, R., Kurokura, T., Tabata, K. & Ozaki, K. Tomato leaf disease diagnosis using bayesian convolutional neural networks. In *2024 IEEE/SICE International Symposium on System Integration (SII)* https://doi.org/10.1109/Sii58957.2024.10417302 (2024).
45. Tm, P., Pranathi, A., SaiAshritha, K., Chittaragi, N. B. & Koolagudi, S. G. Tomato leaf disease detection using convolutional neural networks. In *2018 Eleventh International Conference on Contemporary Computing (IC3)* https://doi.org/10.1109/ic3.2018.8530532 (2018).
46. Rangarajan, A. K., Purushothaman, R. & Ramesh, A. Tomato crop disease classification using pre-trained deep learning algorithm. *Procedia Comput. Sci.* **133**, 1040–1047. https://doi.org/10.1016/j.procs.2018.07.070 (2018).
47. Zaki, S. Z., Asyraf Zulkifley, M., Mohd Stofa, M., Kamari, N. A. & AyuniMohamed, N. Classification of tomato leaf diseases using MobileNet V2. *IAES Int. J. Artif. Intell.* **9**, 290. https://doi.org/10.11591/ijai.v9.i2.pp290-296 (2020).
48. Liu, J. & Wang, X. Tomato diseases and pests detection based on improved Yolo v3 convolutional neural network. *Front. Plant Sci.* https://doi.org/10.3389/fpls.2020.00898 (2020).
49. Zhang, K., Wu, Q. & Chen, Y. Detecting soybean leaf disease from synthetic image using multi-feature fusion faster R-CNN. *Comput. Electron. Agric.* **183**, 106064. https://doi.org/10.1016/j.compag.2021.106064 (2021).
50. KP, A. & Anitha, J. Plant disease classification using deep learning. In *2021 3rd International Conference on Signal Processing and Communication (ICPSC)* https://doi.org/10.1109/icspc51351.2021.9451696 (2021).
51. Kibriya, H., Rafique, R., Ahmad, W. & Adnan, S. M. Tomato leaf disease detection using convolution neural network. In *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)* https://doi.org/10.1109/ibcast51254.2021.9393311 (2021).
52. Chen, H.-C. et al. Alexnet convolutional neural network for disease detection and classification of Tomato Leaf. *Electronics* **11**, 951. https://doi.org/10.3390/electronics11060951 (2022).
53. Alruwaili, M. et al. RTF-RCNN: An architecture for real-time tomato plant leaf diseases detection in video streaming using faster-RCNN. *Bioengineering* **9**, 565. https://doi.org/10.3390/bioengineering9100565 (2022).
54. Soeb Md., J. A. et al. Tea leaf disease detection and identification based on YOLOv7 (YOLO-T). *Sci. Rep.* https://doi.org/10.1038/s41598-023-33270-4 (2023).
55. Ahmad, W., Shah, S. M. & Irtaza, A. Plants disease phenotyping using quinary patterns as texture descriptor. *KSII Trans. Internet Inf. Syst.* https://doi.org/10.3837/tiis.2020.08.009 (2020).
56. Turkoglu, M., Hanbay, D. & Sengur, A. Multi-model LSTM-based convolutional neural networks for detection of apple diseases and pests. *J. Ambient Intell. Human. Comput.* **13**, 3335–3345. https://doi.org/10.1007/s12652-019-01591-w (2019).
57. Al-gaashani, M. S. A. M., Shang, F., Muthanna, M. S. A., Khayyat, M. & Abd El-Latif, A. A. Tomato leaf disease classification by exploiting transfer learning and feature concatenation. *IET Image Process.* **16**, 913–925. https://doi.org/10.1049/ipr2.12397 (2022).
58. Zhao, S., Peng, Y., Liu, J. & Wu, S. Tomato leaf disease diagnosis based on improved convolution neural network by attention module. *Agriculture* **11**, 651. https://doi.org/10.3390/agriculture11070651 (2021).
59. Abbas, A., Jain, S., Gour, M. & Vankudothu, S. Tomato plant disease detection using transfer learning with C-GAN synthetic images. *Comput. Electron. Agric.* **187**, 106279 (2021).
60. Khamparia, A. et al. Seasonal crops disease prediction and classification using deep convolutional encoder network. *Circuits Syst. Signal Process.* **39**, 818–836. https://doi.org/10.1007/s00034-019-01041-0 (2019).
61. Haque, Md. A. et al. Deep learning-based approach for identification of diseases of maize crop. *Sci. Rep.* **12**, 6334. https://doi.org/10.1038/s41598-022-10140-z (2022).
62. Bedi, P. & Gole, P. Plant disease detection using hybrid model based on convolutional autoencoder and convolutional neural network. *Artif. Intell. Agric.* **5**, 90–101. https://doi.org/10.1016/j.aiia.2021.05.002 (2021).
63. Alam, T. S., Jowthi, C. B. & Pathak, A. Comparing pre-trained models for efficient leaf disease detection: A study on custom CNN. *J. Electr. Syst. Inf. Technol.* https://doi.org/10.1186/s43067-024-00137-1 (2024).
64. Nawaz, M. et al. A robust deep learning approach for tomato plant leaf disease localization and classification. *Sci. Rep.* **12**, 18568. https://doi.org/10.1038/s41598-022-21498-5 (2022).
65. Mahadevan, K., Punitha, A. & Suresh, J. Automatic recognition of rice plant leaf diseases detection using deep neural network with improved threshold neural network. *e-Prime Adv. Electr. Eng. Electron. Energy* **8**, 100534 (2024).
66. Tiwari, D. et al. Potato leaf diseases detection using Deep Learning. In *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)* https://doi.org/10.1109/ICICCS48265.2020.9121067 (2020).
67. Sanga, S. L., Machuve, D. & Jomanga, K. Mobile-based deep learning models for banana disease detection. *Eng. Technol. Appl. Sci. Res.* **10**, 5674–5677 (2020).
68. Mohameth, F., Bingcai, C. & Sada, K. A. Plant disease detection with deep learning and feature extraction using Plant Village. *J. Comput. Commun.* **08**, 10–22 (2020).
69. Rahman, H. et al. Automated detection of selected tea leaf diseases in Bangladesh with convolutional neural network. *Sci. Rep.* **14**, 14097. https://doi.org/10.1038/s41598-024-62058-3 (2024).
70. Hughes, D. P. & Salathe, Marcel. An open access repository of images on plant health to enable the development of mobile disease diagnostics. Preprint at https://arxiv.org/abs/1511.08060 (2015).
71. Hammou, D. R. & Boubaker, M. Tomato plant disease detection and classification using convolutional neural network architectures technologies. *Netw. Intell. Syst. Security* **237**, 33–44. https://doi.org/10.1007/978-981-16-3637-0_3 (2021).
72. Dhaka, V. S. et al. A survey of deep convolutional neural networks applied for prediction of plant leaf diseases. *Sensors* **21**, 4749. https://doi.org/10.3390/s21144749 (2021).
73. Eunice, J., Popescu, D. E., Chowdary, M. K. & Hemanth, J. Deep learning-based leaf disease detection in crops using images for agricultural applications. *Agronomy* **12**, 2395. https://doi.org/10.3390/agronomy12102395 (2022).
74. Kukačka, J., Golkov, V. & Cremers, D. *Regularization for Deep Learning: A Taxonomy*. Preprint at https://arxiv.org/abs/1710.10686 (2017).

75. Saponara, S. & Elhanashi, A. Impact of image resizing on deep learning detectors for training time and model performance. *Lecture Notes in Electrical Engineering* 10–17 (2022). https://doi.org/10.1007/978-3-030-95498-7_2
76. Thambawita, V. et al. Impact of image resolution on deep learning performance in endoscopy image classification: An experimental study using a large dataset of endoscopic images. *Diagnostics* **11**, 2183 (2021).
77. Sabottke, C. F. & Spieler, B. M. The effect of image resolution on deep learning in radiography. *Radiol. Artif. Intell.* **2**, e190015. https://doi.org/10.1148/ryai.2019190015 (2020).
78. Kannojia, S. P. & Jaiswal, G. Effects of varying resolution on performance of CNN based image classification an experimental study. *Int. J. Comput. Sci. Eng.* **6**, 451–456. https://doi.org/10.26438/ijcse/v6i9.451456 (2018).
79. Koehrsen, W. A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning. *Medium* https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (2018).
80. Sun, H. & Betti, R. A hybrid optimization algorithm with Bayesian inference for probabilistic model updating. *Comput.-Aided Civil Infrastruct. Eng.* **30**, 602–619 (2015).
81. Jalali, A., Azimi, J. & Fern, X. Exploration vs exploitation in Bayesian optimization. Preprint at https://arxiv.org/abs/1204.0047v1 (2012).
82. Ghahramani, Z. Probabilistic machine learning and artificial intelligence. *Nature* **521**, 452–459. https://doi.org/10.1038/nature14541 (2015).
83. Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. Algorithms for hyper-parameter optimization. *Neural Inf. Process. Syst.* (2011).
84. Bergstra, J., Yamins, D. & Cox, D. D. *Making a Science of Model Search*. Preprint at https://doi.org/10.48550/arXiv.1209.5111 (2012).
85. Bergstra, J., Yamins, D. & Cox, D. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. *Proc. Python Sci. Conf.* https://doi.org/10.25080/majora-8b375195-003 (2013).
86. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444. https://doi.org/10.1038/nature14539 (2015).
87. Tikhonov, A. N. On the stability of inverse problems. *Proc. USSR Acad. Sci.* **39**, 195–198 (1943).
88. Ioffe, S. & Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Preprint at https://doi.org/10.48550/arXiv.1502.03167 (2015).
89. Santurkar, S., Tsipras, D., Ilyas, A. & Madry, A. *How Does Batch Normalization Help Optimization?* Preprint at https://arxiv.org/abs/1805.11604 (2018).
90. Hinton, G. E., Srivastava N., Krizhevsky A., Sutskever I. & Salakhutdinov R. R. Improving neural networks by preventing co-adaptation of feature detectors. Preprint at https://doi.org/10.48550/arXiv.1207.0580 (2012).
91. Hasan, M. J., Kim, J., Kim, C. H. & Kim, J.-M. Health State classification of a spherical tank using a hybrid bag of features and k-nearest neighbor. *Appl. Sci.* **10**, 2525. https://doi.org/10.3390/app10072525 (2020).
92. Kursa, M. B. & Rudnicki, W. R. Feature selection with the Boruta package. *J. Stat. Softw.* https://doi.org/10.18637/jss.v036.i11 (2010).
93. Chen, T. & Guestrin, C. *XGBoost: A Scalable Tree Boosting System*. Preprint at https://doi.org/10.1145/2939672.2939785 (2016).
94. Wolpert, D. H. Stacked generalization. *Neural Netw.* **5**, 241–259. https://doi.org/10.1016/s0893-6080(05)80023-1 (1992).
95. Li, B., Migan-Dubois, A., Delpha, C. & Diallo, D. Complexity analysis of convolutional neural network applied to PV fault diagnosis via image processing. *Space* **1**, 1 (2019).
96. Abadi, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems https://doi.org/10.48550/arXiv.1603.04467 (2016).
97. Chollet, F. Keras. *J. Chem. Inf. Model.* (2013).
98. Pedregosa, F. et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).

Acknowledgements

The authors would like to extend their sincere appreciation to the Researchers Supporting Project Number (RSP2024R410), King Saud University, Riyadh, Saudi Arabia. The authors express gratitude to the creators and contributors of the PlantVillage Dataset (kaggle.com) and the PlantDoc Dataset, which have been utilized in this research.

Author contributions

B. K., planned, designed, and conceptualized the research, implemented the hybrid models, and performed the coding, data curation, data analysis, visualization, validation, literature review, and writing of the original draft manuscript, manuscript reviewing, editing, and finalizing; S. D., contributed to the methodology, validation, literature review, manuscript writing, reviewing, editing, and finalizing; N.S.F., contributed to the literature review, manuscript writing, reviewing, editing; S. B., planned the research and contributed to validation, literature review, manuscript writing; S.K., was involved in literature review, validation, manuscript writing, and reviewing; A.R.M.T.I: Reviewing, editing and supervision and M.K.A.S., and H.A.O: Reviewing and editing. All authors read and approved the final manuscript.

Funding

Supported by Researchers Supporting Project Number (RSP2024R410), King Saud University, Riyadh, Saudi Arabia.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-024-72237-x.

Correspondence and requests for materials should be addressed to B.K. or A.R.M.T.I.