Innovations in Agriculture
www.innovationsagriculture.com

ISSN: 2617-1295
DOI: https://doi.org/10.25081/ia.2023-04

Research Article

Detection: A convolutional network method for plant disease recognition

Enow Takang Achuo Albert*, Ngalle Hermine Bille, Ngonkeu Mangaptche Eddy Leonard

Department of Plant Biology, Faculty of Science, University of Yaoundé I, P.O. Box 812, Yaoundé, Center Region, Cameroon

*Corresponding Author, Email: albert.enow@facsciences-uy1.cm

ARTICLE INFO

Received: June 11, 2023
Revised: July 08, 2023
Accepted: July 08, 2023
Published: July 09, 2023

Keywords:

plant disease recognition, convnets, feature extraction layers, ablation testing

ABSTRACT

Within the last decade and a half, ConvNet models have shown increasingly impressive performances on image classification tasks. The continuous quest for better performing ConvNet models is of great importance in that it provides an avenue for the development of superior models which would serve to improve many much-needed services to humankind in domains such as crop pest and disease detection. This paper proposes a feature extraction ConvNet, called Detection, with aim to improve plant disease recognition. Detection contains four convolutional layers, four batch normalization layers and two max-pooling layers. For reference, its plant disease recognition performance was compared with the feature extraction layers for AlexNet, LeNet-5, ZFNet and VGGNet on four datasets from PlantVillage: bacterial spot disease (bell pepper), late blight disease (tomato), leaf mold disease (tomato) and yellow leaf curl disease (tomato). Prior to ablation testing, it achieved the second-highest overall classification accuracy over all four disease recognition datasets (65.50%) after AlexNet (84.33%). However, ablation tests revealed that the removal of the second convolutional layer from the network resulted in a 24.08% increase in overall accuracy on all four disease recognition datasets, up from 65.50% accuracy without the ablation. This surpassed AlexNet's overall accuracy on all four disease recognition datasets (84.33%) by 5.25%. Also, the removal of the second pooling layer from the network resulted in a 23.33% increase in overall accuracy on all four disease recognition datasets, up from 65.50% accuracy without the ablation. This also surpassed AlexNet's overall accuracy on all four disease recognition datasets (84.33%) by 4.40%. The results suggest that the proposed feature extraction ConvNet is a performant method for plant disease recognition.

INTRODUCTION

The continuous pursuit of better image classification AI models is crucial for several reasons. (1) Improved Accuracy: The primary goal of developing better models is to improve accuracy. Higher accuracy means the model can correctly classify more images, which is beneficial in many applications, including plant disease recognition. For instance, a more accurate model could lead to earlier detection of diseases [1]. (2) Efficiency: Better models can also be more efficient, requiring less computational resources or less time to train or make predictions. This is particularly important in real-time applications, such as image/video processing or in devices with limited computational power [2]. (3) Robustness: Improved models can be more robust to variations in the input data, such as changes in lighting, viewpoint, or scale. This is crucial for many real-world applications where these conditions can't be controlled [3]. (4) Generalization: Better models can generalize more effectively to new, unseen data. This is important because a model's ultimate purpose is to make accurate predictions on new data, not just the data it was trained on [4]. (5) Interpretability: As AI models are increasingly used in decision-making, it's important that they're interpretable, meaning that their predictions can be understood and explained by humans. Better models can provide more interpretable predictions, increasing trust and acceptance of AI systems [5]. (6) Fairness: Improved models can also be designed to be fairer, reducing bias in their predictions. This is particularly important in applications where biased predictions can have serious consequences [6]. Further to the foregoing, we set out to present an improved feature extraction ConvNet for better plant disease recognition (called *Detection*).

Basic ConvNet operations.

A convolution operation is a fundamental building block of a Convolutional Neural Network (CNN or ConvNet). It is a mathematical operation that takes two inputs: an image matrix and a filter or kernel. The former is the input data to the convolution operation. In the context of a CNN, this could be the raw pixel values of an image or the output of a previous layer in the network. The latter is a smaller matrix that is used to transform the image matrix. The values in the filter are learnable parameters, meaning they are updated during the training process to help the network better perform its task [7]. The convolution operation involves sliding the filter across the image matrix and at each position, performing an element-wise multiplication of the values in the filter and the values in the image that it currently covers. These multiplied values are then summed up to give a single output value. This process is repeated for every position in the image that the filter can slide to, resulting in a new matrix called a feature map or convolutional layer. The filter is typically much smaller than the image, but it extends through the full depth of the input. For example, if the input is a color image composed of three (RGB) channels, the filter also has a depth of three. This ensures that the filter operates on all color channels simultaneously.

The stride and padding are two important concepts in the convolution operation: The former is the number of pixels by which the filter matrix is slid over the input matrix. When the stride is 1, the filter moves one pixel at a time. When the stride is 2, the filter moves two pixels at a time, and so on. A larger stride results in a smaller feature map. The latter is a process of adding extra pixels around the

© The Authors. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/)

input image. Padding can be used to control the spatial size of the output volumes (i.e., the width and height dimensions). It is often used to ensure that the spatial dimensions of the output volume match the input volume. The convolution operation captures the local dependencies in the original image, such as the shapes of objects and the spatial relationships between pixels. By using multiple filters, a CNN can learn to detect various features in the input data, such as edges, corners, and color blobs, which can then be combined to recognize more complex patterns.

Pooling is a fundamental operation in Convolutional Neural Networks, typically applied after each convolutional layer. It serves as a form of non-linear down-sampling, reducing the spatial dimensions (width and height) of the input, which helps to decrease the computational complexity of the network, control overfitting, and make the network invariant to small translations [7]. There are two main types of pooling operations used in ConvNets: Max Pooling and Average Pooling. Max pooling involves scanning over the input with a filter (also known as a kernel) and at each step, returning the maximum value within the filter as a pixel in the output. This operation effectively reduces the size of the input and retains only the most significant information, i.e., the maximum values. This is particularly useful for feature detection since it retains the most activated features while discarding the less activated ones. Average pooling, on the other hand, calculates the average value of the elements within the filter. This results in a smoother representation of the input, retaining the overall structure and reducing the impact of outliers. While max pooling provides a more aggressive dimensionality reduction by focusing on the most activated features, average pooling provides a more balanced representation. In addition to these, there are also Global Max Pooling and Global Average Pooling, which compute the max or average of an entire feature map, respectively [8]. These operations reduce the entire feature map to a single value, summarizing the presence of a feature in the entire input.

The concept of strides is also important in pooling. The stride parameter can further influence the dimensionality reduction. Pooling layers provide several benefits in a CNN architecture: (1) Dimensionality Reduction: By reducing the spatial dimensions of the input, pooling layers significantly decrease the computational complexity of the model, making the network faster and less memory-intensive. (2) Translation Invariance: Pooling provides a form of translation invariance, meaning the network will be able to recognize an object regardless of where it is located in the image. (3) Overfitting Control: By providing a form of spatial summarization, pooling layers also help to control overfitting by providing a more abstract representation of the input [7]. In practice, the choice between max pooling and average pooling depends on the specific application and the type of data. Max pooling tends to be more commonly used, but in some cases, average pooling can provide better performance.

Related literature

Previous works in plant disease identification are noteworthy. Abbas et al. [9] set to develop a deep learning-based framework for the recognition of tomato plant diseases. The aim was to investigate tomato leaf images and create a method that would assist farmers in classifying diseases affecting tomato cultivation. The goal was to provide a cost-effective alternative to expert analysis by allowing farmers to simply take an image of diseased leaves for disease classification. The proposed model for tomato plant disease detection was a deep learning-based framework that combined the power of Conditional Generative Adversarial Network (C-GAN) for generating synthetic images and the efficiency of the DenseNet121 model for classification. The model architecture consisted of three main components: data preprocessing, synthetic image generation, and classification. In the data preprocessing step, the tomato leaf images were resized to 224x224 pixels and normalized to improve the model's performance. In the synthetic image generation step, the C-GAN model was used to generate synthetic images. The C-GAN model was trained on a subset of the tomato leaf images to generate synthetic images that were similar to the original images. The synthetic images were then added to the original dataset to increase the size of the training dataset. In the classification step, the DenseNet121 model was used to classify the tomato leaf images into ten categories of diseases. The DenseNet121 model is a convolutional neural network (CNN) that has been pre-trained on the ImageNet dataset. The model consists of a series of convolutional layers, followed by dense blocks and transition layers. The dense blocks contain multiple convolutional layers that are connected to each other, while the transition layers reduce the spatial dimensions of the feature maps. The DenseNet121 model was fine-tuned on the augmented dataset, which included both the original and synthetic images. The model was trained using the categorical cross-entropy loss function and the Adam optimizer. The performance of the model was evaluated using various metrics such as accuracy, precision, recall, and F1-score. The evaluation was conducted on three different classification tasks: 5-class classification, 7-class classification, and 10-class classification. For the 5-class classification task, which involved classifying tomato leaf images into five disease categories, the proposed method achieved a classification accuracy of 99.51%. For the 7-class classification task, which involved classifying tomato leaf images into seven disease categories, the proposed method achieved a classification accuracy of 98.65%. For the 10-class classification task, which involved classifying tomato leaf images into ten disease categories, the proposed method achieved a classification accuracy of 97.11%. These results indicated that the proposed model performed exceptionally well in accurately classifying tomato leaf images into different disease categories. The high accuracy scores demonstrated the effectiveness of the model in disease detection and highlighted its potential for practical implementation in the agricultural sector.

Atila et al. [10] developed a plant leaf disease classification system using the EfficientNet deep learning model. The researchers aimed to improve the accuracy and precision of disease diagnosis by eliminating the need for manual feature extraction and utilizing the power of deep learning algorithms. Their goal was to provide a timely and accurate diagnosis of plant diseases, which is crucial for sustainable and correct agriculture. The EfficientNet deep learning architecture is based on a compound scaling method that uniformly scales the depth, width, and resolution of the network. It has several variants, including B0 to B7, with B0 being the smallest and B7 being the largest. In this study, the B4 and B5 models of the EfficientNet architecture were used, which have approximately 19 million and 30 million parameters, respectively. The EfficientNet architecture uses a transfer learning approach, where all layers of the model are set to be trainable. The architecture consists of a stem convolutional layer, multiple blocks of convolutional layers, and a top layer with a global average pooling layer and a fully connected layer. The blocks of convolutional layers are composed of a combination of depth-wise separable convolutions and pointwise convolutions, which reduces the number of parameters and computational cost while maintaining high accuracy. The architecture also uses Swish activation function, which has been shown to outperform ReLU in some cases. The

performance results of the EfficientNet models in this study were quite impressive. The B4 and B5 models showed very close performance, with the B4 model being slightly more successful than the B5 on the augmented dataset. In terms of accuracy, the B5 model achieved the highest accuracy of 99.91% on the original dataset, while the B4 model achieved the highest accuracy of 99.97% on the augmented dataset. When evaluating the models in terms of precision, it was observed that the values varied between 90.26% and 99.39%, showing more significant differences compared to accuracy values. The B4 model, which offered the best performance on the augmented dataset, showed improvement in both accuracy and precision, with an increase of 0.13% and 2.16% respectively. This indicates that as the number of data increased, the number of false positive predictions decreased, leading to an increase in precision. It is worth noting that all models used in the study achieved high accuracy values ranging from 99.45% to 99.97%. Additionally, the success of all models on the augmented dataset was higher than the original dataset, as shown by the test accuracy curves.

Al-Adhaileh et al. [11] customized a convolutional network to improve early detection of potato late blight. Their architecture consisted of two blocks, each containing a pair of convolution layers followed by a pooling layer. Their model was implemented using TensorFlow version 2.6.4. It was trained and evaluated using a total of 2,652 potato leaf images, which were distributed almost equally across three classes. During training, 2,108 augmented images were used for 45 epochs, and the model's performance was assessed using 288 actual images. The model had a total of 839,203 trainable parameters. An Adam optimizer with a learning rate of 0.001 was used to adjust the weights of neurons in the neural network. The training time for the model was 183 seconds. A confusion matrix was used to represent the class wise performance of the model. The classes in this study were early blight, late blight, and healthy. Their model achieved an inclusive accuracy of 99%.

Albattah et al. [12] proposed a computationally efficient technique for plant leaves disease detection using a one-stage object detection framework. The aim was to demonstrate the robustness of the proposed approach by conducting extensive experimentation and comparing it to other latest plant disease detection approaches on a standard PlantVillage database. The research focused on addressing various distortions such as blurriness, chrominance, intensity variations, high-density noise, rotational, and scaling variations. Their proposed technique was called CenterNet. The CenterNet model is a deep learning architecture that uses a single convolutional neural network (CNN) to simultaneously predict object locations and categories. It consists of three main components: a backbone network, a feature pyramid network (FPN), and a detection network. The backbone network is responsible for extracting features from the input image, while the FPN is used to generate a set of feature maps at different scales. The detection network then takes these feature maps as input and predicts the object locations and categories. The backbone network used in this study was a modified version of the ResNet-18 architecture, which is a popular CNN model for image classification tasks. The FPN is based on a top-down architecture that combines low-level and high-level features to generate a set of feature maps at different scales. The detection network is composed of a set of convolutional layers that predict the object locations and categories. Their model achieved a precision of 0.992, recall of 0.982, and F1-score of 0.987 for the Apple-Scab class. The model also achieved perfect precision, recall, and F1-score values for some classes such as Apple-Healthy, Blueberry-Healthy, and Cherry-Healthy. In addition, the proposed model was compared to other machine learning (ML) classifiers such as SVM and KNN, and it achieved the highest accuracy value of 99.982%. The SVM classifier obtained the second-best results with an accuracy of 98.01%, while the KNN exhibited lower performance with an accuracy value of 91.01%. These results demonstrated that the proposed model was more robust to plant disease detection and classification than the ML-based classifiers due to its ability to deal with over-fitted training data.

Devisurya et al. [13] developed an improved deep learning algorithm for the early detection of major diseases in turmeric plants. The model used in this study was the Improved YOLOV3-Tiny model, which is a deep learning-based technique. It was designed to simplify the recognition of turmeric plant leaf diseases by adopting a single-phase detection model instead of using multiple steps such as image pre-processing, feature extraction and feature classification in the conventional method. The performance results of the Improved YOLOV3-Tiny (IY3TM) model in detecting turmeric leaf diseases were compared with other deep learning models such as YOLO-V2, YOLO-V3, YOLOV3-tiny, and Faster R-CNN with VGG-16. The IY3TM model demonstrated better performance than these models in terms of F1 score, detection speed, and AUC values. Additionally, the F1 score of the IY3TM model with Cycle-GAN augmentation was higher than other models when evaluated on the same datasets.

Li et al. [14] developed a method called WG-MARNet for the classification of maize leaf diseases. This method aimed to eliminate noise in maize images, enhance the focus characteristics of maize, and achieve high-precision recognition of maize disease images. WG-MARNet utilizes deep learning and convolutional neural networks for disease classification. Additionally, a structure block scheme is adopted, which involves a nonlinear mapping function for a single structure. The performance results of the WG-MARNet model were evaluated in terms of accuracy and loss value. The model achieved an accuracy of 97.18% and a single image detection time of 0.278s.

Mohanty et al. [15] explored the use of deep learning for image-based plant disease detection and investigated whether their neural network actually learnt the "notion" of plant diseases or if it was just learning the inherent biases in their dataset. The architecture of the model used in this study was based on the AlexNet architecture, which is a deep convolutional neural network (CNN). The performance results of the model were evaluated using different train-test set splits, ranging from 80-20 to 20-80. The overall accuracy of the model varied depending on the split, but the top accuracy achieved was 99.35%. The study also compared the performance of the model using different versions of the dataset, including colored, grayscale, and segmented images. The model performed best with the colored version of the images, followed by the segmented images, and slightly lower with the grayscale images. The worst performance still resulted in an observed mean F1 score of 0.8524, corresponding to an overall accuracy of 85.53%.

Ramcharan et al. [16] investigated the use of transfer learning applied to the Inception-v3 deep learning model for in-field disease detection in cassava plants using convolutional neural networks. The researchers aimed to determine if transfer learning could improve the accuracy of disease detection, particularly when working with relatively small image datasets. The accuracies ranged from 73% to 91% for the original cassava dataset and from 80% to 93% for the leaflet cassava dataset, depending on the training-testing data splits used.

Finally, Reddy and Sashi Rekha [17] proposed a methodology for plant disease detection using advanced convolutional neural networks with region of interest awareness. Their methodology

involved the following steps: (1) Data Collection: The authors collected the PlantVillage dataset, which contains images of healthy and diseased plant leaves. (2) Data Preprocessing: The collected images were preprocessed to remove noise and enhance the contrast and brightness of the images. (3) Region of Interest (ROI) Extraction: The authors used an algorithm called ROI Feature Map Creation (ROI-FMC) to extract the regions of interest from the preprocessed images. This algorithm identifies the areas of the image that are most likely to contain disease symptoms and extracts them for further analysis. (4) Model Training: The authors used pre-trained deep CNN models such as AlexNet, DenseNet19, Inception-v3, ResNet34, Sqeezenet1-1, and VGG13 to train their model on the extracted ROIs. Transfer learning was used to optimize the training and testing phases of the model. (5) Model Evaluation: The authors evaluated the performance of their model using various metrics such as accuracy, precision, recall, and F1 score. They compared the performance of their model with other state-of-the-art models for plant disease detection. The results showed that the usage of DenseNet consistently increased performance in disease detection. Also, VGG16 showed the highest accuracy compared to other deep learning models and machine learning models like RF, SVM, and SGD. Furthermore, with retraining, there was a performance enhancement observed in terms of reduced evaluation time and improved accuracy. The proposed methodology of using Inception_V3 with transfer learning showed the highest performance with an accuracy of 99.7637223%.

MATERIALS AND METHODS

Dataset

The dataset used for this research was obtained from the PlantVillage collection. The PlantVillage dataset [18] is a publicly available collection of images that are used for plant disease detection. It was created by the team at PlantVillage, a research and development unit of Penn State University. The dataset is often used in machine learning and artificial intelligence research to develop models that can identify plant diseases from images. The PlantVillage dataset contains over 50,000 images, spanning 38 classes of diseased and healthy plant species. The images are in color and have been preprocessed to remove the background. The dataset includes a wide variety of plant species, including but not limited to apple, blueberry, cherry, corn, grape, orange, peach, bell pepper, potato, raspberry, soybean, squash, strawberry, and tomato. Each species in the dataset has images showing healthy leaves as well as leaves with various diseases. The diseases represented in the dataset include but are not limited to apple scab, black rot, cedar apple rust, cherry powdery mildew, corn gray leaf spot, grape black rot, orange huanglongbing, peach bacterial spot, bell pepper bacterial spot, potato early blight, raspberry leaf spot, soybean rust, squash powdery mildew, strawberry leaf scorch, and tomato bacterial spot [19].

For this study, the dataset consisted of seven-thousand two-hundred (7,200) images, divided into four subsets as follows: (1) 1,800 images for bacterial spot disease recognition in bell pepper, (2) 1,800 images for late blight disease recognition in tomato, (3) 1,800 images for leaf mold disease recognition in tomato, and (4) 1,800 images for yellow leaf curl disease recognition in tomato. For each subset, the 1,800 images were organized as follows: a training dataset folder and a testing dataset folder were created. In each of the aforementioned folders, a ‘disease’ and a ‘healthy’ folder were created. Next, 500 images of symptomatic leaves and 500 images of asymptomatic leaves were put into the ‘disease’ and ‘healthy’ folders, respectively, in the training dataset folder, while 400 images of symptomatic leaves and 400 images of asymptomatic leaves were put into the ‘disease’ and ‘healthy’ folders, respectively, in the testing dataset folder. Figure 1 presents instances of data from each subset.

The reason why multiple disease recognition datasets were used for this study was to test the generalizability of the proposed feature extraction ConvNet for better plant disease recognition. Generalizability in machine learning refers to the model's ability to adapt effectively to new, previously unseen data, drawn from the same domain/distribution as the one used to create the model. It's a fundamental concept in machine learning, as the main goal of these models is to make accurate predictions on new, unseen data, not just the data they were trained on. To evaluate a model's generalizability, the data is typically split into a training set and a test set. The model is trained on the training set and then evaluated on the test set. The model's performance on the test set gives an indication of how well the model is likely to perform on unseen data [20]. The following concepts relate to model generalizability: (1) Overfitting and underfitting: Overfitting occurs when a model learns the training data too well, including its noise and outliers, and performs poorly on unseen data. This usually happens when the model is too complex, such as having too many parameters. Underfitting, on the other hand, occurs when the model is too simple to learn the underlying structure of the data, resulting in poor performance on both the training and test sets. Both overfitting and underfitting are undesirable, as they reduce the model's generalizability [21]. (2) Bias-Variance Tradeoff: This is a fundamental concept related to generalizability. Bias refers to the error due to the model's assumptions in the learning algorithm. High bias can cause underfitting. Variance refers to the error due to the model's sensitivity to fluctuations in the training set. High variance can cause overfitting. The goal is to find a balance where both bias and variance are minimized, leading to a model that generalizes well [22]. (3) Cross-Validation: This is a technique used to assess the generalizability of the model. The most common method is k-fold cross-validation, where the data is split into k subsets. The model is trained on k-1 subsets and tested on the remaining subset. This process is repeated k times, with each subset used exactly once as the test set. The average performance across the k iterations gives a better measure of the model's generalizability [23]. (4) Regularization: This is a technique used to improve the model's generalizability by adding a penalty term to the loss function to reduce the complexity of the model. Examples include L1 and L2 regularization [24]. (5) Model Complexity and Capacity: The complexity or capacity of a model, which can be controlled by parameters such as the number of layers in a neural network or the depth of a decision tree, can influence its generalizability. A model with too much capacity can easily overfit the training data, while a model with too little capacity might not capture the underlying patterns in the data, leading to underfitting [25]. (6) Data Augmentation: This technique involves creating new training samples via transformations such as rotations, translations, or flips. This can help improve the model's generalizability by providing more varied training data. The ultimate goal of a machine learning model is to make accurate predictions on new, unseen data. Therefore, generalizability is a key factor to consider when training and evaluating models [26].

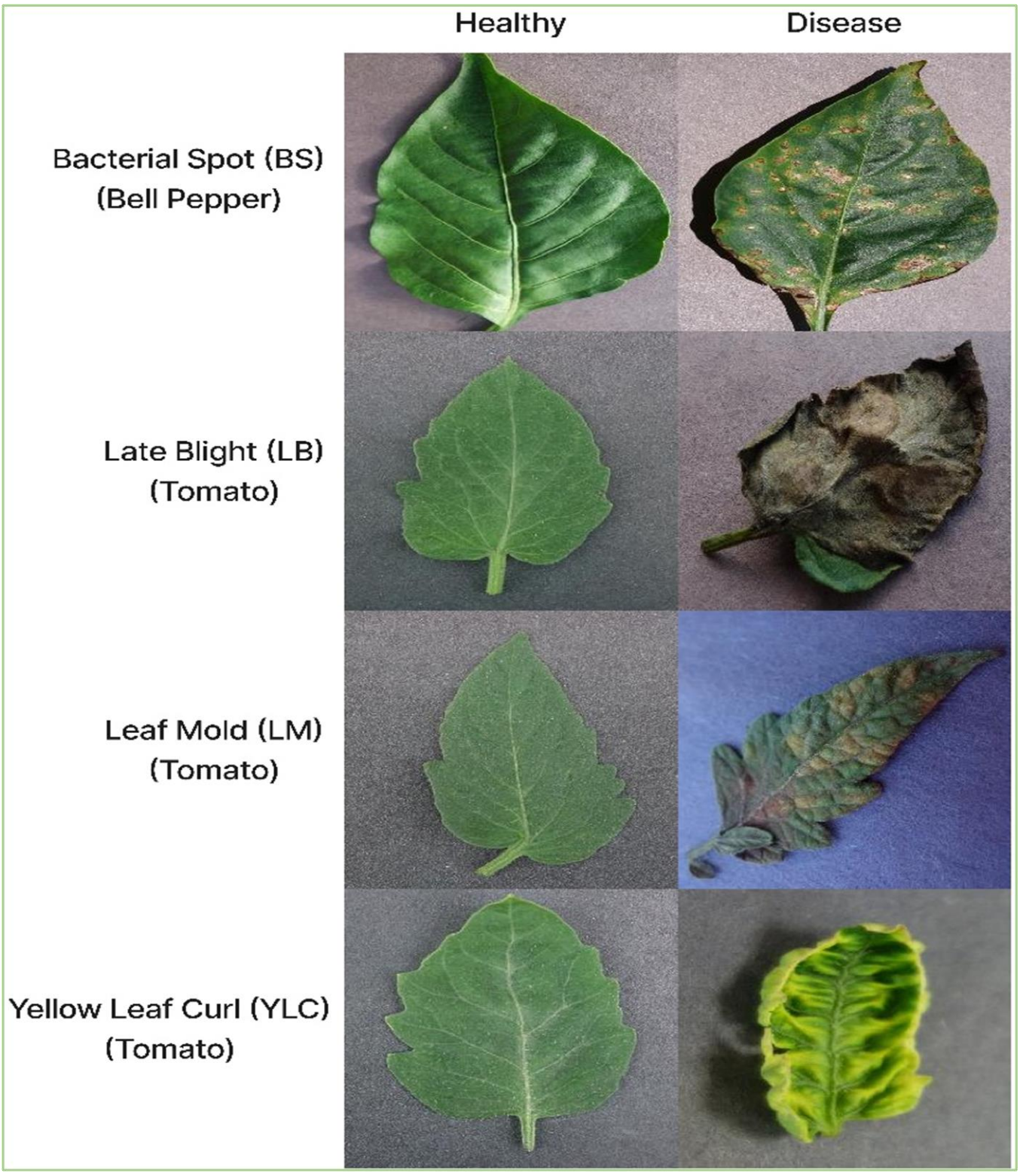

Fig. 1. Instances of data used for this study.

Analytical Procedure

It included: (1) loading the dataset, (2) defining the feature extraction networks (in addition to Detection, the feature extraction networks of four popular image classifier models, namely AlexNet, LeNet-5, VGGNet and ZFNet, were employed in order to determine the superiority of Detection, or the lack thereof), (3) Visualizing feature maps from all of the networks defined in the penultimate step, (4) fitting the feature extraction networks, (5) defining a random forest algorithm as feature extraction network classifier, (6) evaluating the feature extraction capabilities of each network using

the accuracy metric (since balanced datasets were used), and (7) performing an ablation test to determine the contribution of each of Detection's layers to its feature extraction capability. The scripts used for the afore-mentioned computational phases can be accessed from [27].

Architectures of popular ConvNets used for this study

AlexNet is a deep learning architecture that was proposed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in their paper "ImageNet Classification with Deep Convolutional Neural Networks" presented at the Neural Information Processing Systems Conference in 2012 [28]. This architecture significantly influenced the field of Deep Learning, particularly for its application in Image Classification tasks. The AlexNet architecture is as follows: (1) Input Layer: The input to the network is a 224x224 RGB image which passes through the first convolutional layer. (2) Convolutional Layers: AlexNet contains five convolutional layers. The first convolutional layer filters the 224x224x3 input image with 96 kernels of size 11x11x3 with a stride of 4 pixels. The second convolutional layer takes the output from the first layer, applies max-pooling, and then convolution with 256 kernels of size 5x5x48. The third, fourth, and fifth convolutional layers are connected without any pooling layers in between. They used 384, 384, and 256 kernels of size 3x3x256 respectively. (4) Pooling Layers: Max pooling is performed over a 3x3 spatial window with stride 2 in the first, second, and fifth convolutional layers. (4) Normalization Layers: Local Response Normalization (LRN) was applied after the first and second convolutional layers. LRN encourages lateral inhibition by normalizing over local input regions. (5) Fully Connected Layers: After the convolutional layers and max pooling, the network has three fully connected layers. The first two have 4096 neurons each. These layers are followed by a SoftMax layer for output, which has 1000 neurons corresponding to the 1000 ImageNet classes. (6) Activation Function: The Rectified Linear Unit (ReLU) is applied as an activation function after every convolutional and fully connected layer. (7) Dropout: To reduce overfitting, the authors used a dropout rate of 0.5 in the first two fully connected layers during training. (7) Splitting on Multiple GPUs: The original AlexNet was trained on two GPUs, with the architecture specifically designed to keep the GPUs as specific as possible. For example, the response-normalization layers' activities are pooled across GPUs to remain consistent. This architecture won the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) by a large margin, reducing the top-5 error from 26% to 15.3%.

LeNet-5 is a convolutional neural network architecture developed by Yann LeCun, Leon Bottou, Yosuha Bengio, and Patrick Haffner in 1998 [29]. It was designed for handwritten and machine-printed character recognition and is considered one of the pioneering works in the field of deep learning. The architecture of LeNet-5 consists of seven layers, not including the input, each of which contains trainable parameters (weights). The LeNet-5 architecture is as follows: (1) Input Layer: The input for LeNet-5 is a 32x32 grayscale image. The extra padding around the 28x28 image allows features to be learned from the corners and edges of the input image. (2) C1 - Convolutional Layer: This layer contains 6 kernels of size 5x5, resulting in feature maps of size 28x28. Each unit in each feature map is connected to a 5x5 neighborhood in the input, allowing the network to detect local features. (3) S2 - Subsampling (Pooling) Layer: This layer contains 6 feature maps of size 14x14. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C1. The four inputs to a unit in S2 are added, then multiplied by a trainable coefficient, and added to a trainable bias. The result is passed through a sigmoidal function. This layer helps to provide a form of translation invariance. (4) C3 - Convolutional Layer: This layer contains 16 kernels of size 5x5, resulting in feature maps of size 10x10. The connections are not made to all S2 feature maps, which reduces the complexity of the network. (5) S4 - Subsampling (Pooling) Layer: This layer is similar to S2, but it operates on C3. It contains 16 feature maps of size 5x5. (6) C5 - Convolutional Layer: This layer is a fully connected layer with 120 units. It's labeled as a convolutional layer because if LeNet-5 input were made larger with everything else kept constant, the feature map dimension would be larger than 1x1. (7) F6 - Fully Connected Layer: This layer is a fully connected layer with 84 units. (8) Output Layer: The output layer is a Euclidean Radial Basis Function layer with 10 units, corresponding to the 10-digit classes. The network uses average pooling and tanh or sigmoid activation functions, which were standard at the time but are now often replaced with max pooling and ReLU activation functions in more modern architectures. LeNet-5 was trained with Stochastic Gradient Descent (SGD) and was one of the first successful applications of convolutional networks.

ZFNet, also known as Zeiler & Fergus Net, is a Convolutional Neural Network (CNN) that was developed by Matthew Zeiler and Rob Fergus in their paper "Visualizing and Understanding Convolutional Networks" in 2013. The architecture is a modification of AlexNet and was the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2013 [30]. The ZFNet architecture is described as follows: (1) Input Layer: The input to the network is a 224x224 RGB image. (2) Convolutional Layers: ZFNet consists of five convolutional layers. The first convolutional layer filters the 224x224x3 input image with 96 kernels of size 7x7 with a stride of 2 pixels. The second convolutional layer takes the output from the first layer, applies max-pooling, and then convolution with 256 kernels of size 5x5. The third, fourth, and fifth convolutional layers are connected without any intervening pooling layers. They use 512, 1024, and 512 kernels of size 3x3 respectively. (3) Pooling Layers: Max pooling is performed over a 3x3 spatial window with stride 2 in the first, second, and fifth convolutional layers. (4) Normalization Layers: Local Response Normalization (LRN) is applied after the first and second convolutional layers. LRN encourages lateral inhibition by normalizing over local input regions. (5) Fully Connected Layers: After the convolutional layers and max pooling, the network has three fully connected layers. The first two have 4096 neurons each. These layers are followed by a SoftMax layer for output, which has 1000 neurons corresponding to the 1000 ImageNet classes. (6) Activation Function: The Rectified Linear Unit (ReLU) is applied as an activation function after every convolutional and fully connected layer. The main difference between ZFNet and AlexNet is the size of the kernels in the first convolutional layer. In AlexNet, the size of the kernels in the first layer is 11x11, while in ZFNet, the size is reduced to 7x7. This change was made because Zeiler and Fergus found that the smaller filter size in the first layer improved the performance of the network.

VGGNet, also known as Visual Geometry Group Network, is a convolutional neural network (CNN) architecture that was proposed by K. Simonyan and A. Zisserman from the University of Oxford in their 2014 paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [31]. The main contribution of their work was a thorough evaluation of networks of increasing depth, which showed that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16-19 weight layers. This finding was in contrast to the common belief back then that deeper networks are harder to train due to the

problem of vanishing gradients. The architecture of VGGNet is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a SoftMax classifier. There are two popular variants of VGGNet, namely VGG16 and VGG19. VGG16 has 16 layers (13 convolutional layers, 3 fully connected layers), and VGG19 has 19 layers (16 convolutional layers, 3 fully connected layers). One of the key aspects of VGGNet is that, despite its architecture's depth, it's conceptually very simple. It consists of only convolutional layers and fully connected layers. However, it's worth noting that VGGNet is more memory intensive and has more computational cost than some other models, such as AlexNet, due to the large number of parameters involved in its fully connected layers. Despite these drawbacks, VGGNet has been widely used in the deep learning community, both because of its simplicity and because it has performed well in a variety of applications, including image recognition and transfer learning. It is worth mentioning that the implementation of the feature extraction layers of these state-of-the-art networks fully relied on the algorithms provided in [8].

Feature extraction network design for Detection.

(1) The input to the network is a 128x128 RGB image which passes through the first convolutional layer. (2) The second layer is a convolutional layer with properties as follows: sigmoid activation; batch input shape 128, 128, 3; dilation rate 1, 1; 32 filters; kernel shape 3, 3; same padding; stride shape 1, 1. (3) The third layer is a batch normalization layer with gamma=32; beta=32; moving mean=32; moving variance=32; axis=3. (4) The fourth layer is a convolutional layer with properties as follows: sigmoid activation; dilation rate 1, 1; 32 filters; kernel shape 3, 3; same padding; stride shape 1, 1. (5) The fifth layer is a batch normalization layer with gamma=32; beta=32; moving mean=32; moving variance=32; axis=3. (6) The sixth layer is a max pooling layer with strides 2, 2; pool size 2, 2. (7) The seventh layer is a convolutional layer with properties as follows: sigmoid activation; dilation rate 1, 1; 64 filters; kernel shape 3, 3; same padding; stride shape 1, 1. (8) The eighth layer is a batch normalization layer with gamma=64; beta=64; moving mean=64; moving variance=64; axis=3. (9) The ninth layer is a convolutional layer with properties as follows: sigmoid activation; dilation rate 1, 1; 64 filters; kernel shape 3, 3; same padding; stride shape 1, 1. (10) The tenth layer is a batch normalization layer with gamma=64; beta=64; moving mean=64; moving variance=64; axis=3. (11) The eleventh layer is a max pooling layer with strides 2, 2; pool size 2, 2. Figures 2 and 3 show visualizations of the Detection architecture using a LeNet and an AlexNet style respectively.

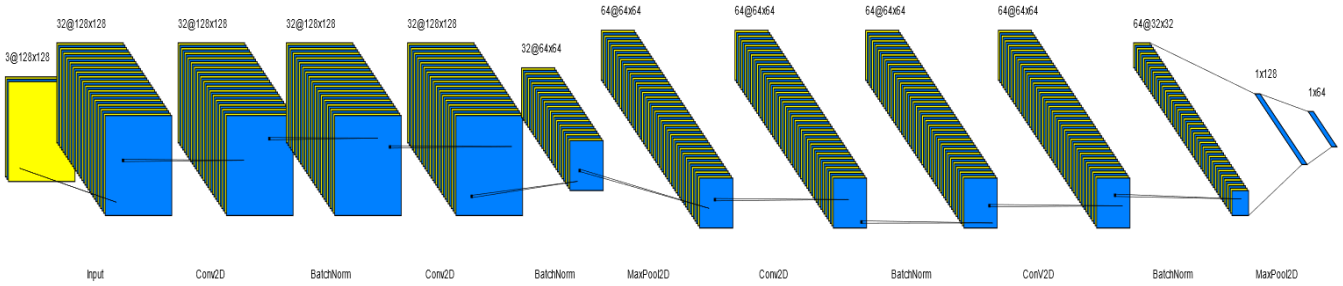

Fig. 2. Model architecture (LeNet Style)

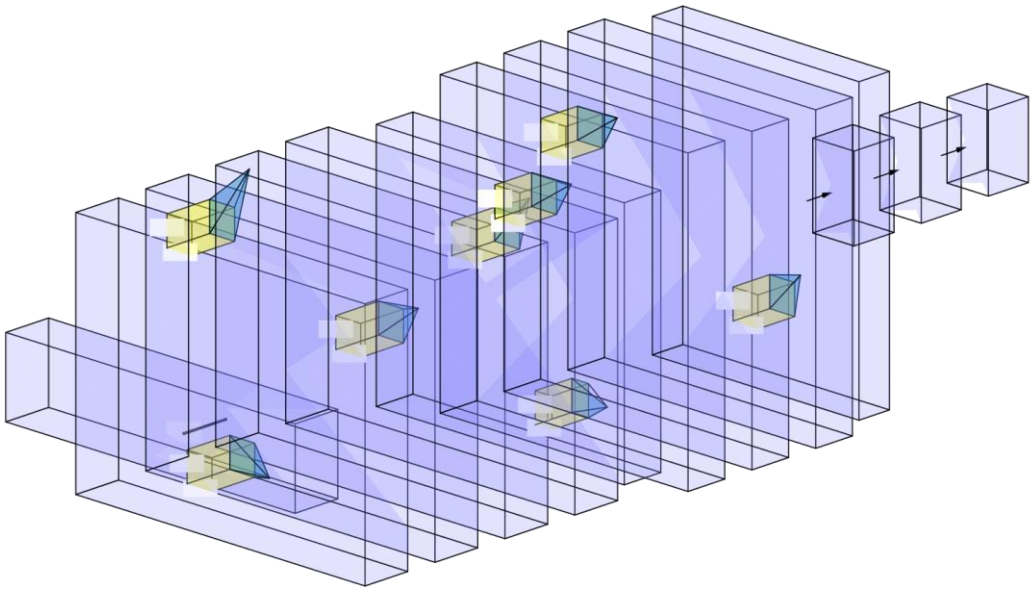

Fig. 3. Model architecture (AlexNet Style)

RESULTS

Evaluation of network performances

The performances of the feature extraction networks on four selected image classification datasets from PlantVillage are detailed in Table 1. On the bacterial spot disease recognition dataset, AlexNet performed best with a 77.80% accuracy score. This was closely followed by VGGNet with a 71.16% accuracy score. The least performant feature extraction network was LeNet-5 with a 50.20% accuracy score, which very closely followed Detection (with a 50.80% accuracy score. On the late blight disease recognition dataset, Detection performed best with an 80.80% accuracy score. This was closely followed by AlexNet with a 79.60% accuracy score. The least performant feature extraction network was ZFNet with a 49.20% accuracy score, which very closely followed LeNet-5 (with a 52.50% accuracy score). On the leaf mold disease recognition dataset, AlexNet performed best with an 86.50% accuracy score. This was closely followed by Detection with an 80.00% accuracy score. The least performant feature extraction network was LeNet-5 with a 47.68% accuracy score, which closely followed both VGGNet and ZFNet (with accuracy scores of 50.80% each). On the yellow leaf curl disease recognition dataset, AlexNet performed best with a 93.40% accuracy score. This was followed by LeNet-5 with a 52.00% accuracy score. The least performant feature extraction network was VGGNet with a 49.00% accuracy score, which very closely followed ZFNet (with a 49.30% accuracy score). Detection obtained a 50.40% accuracy score on this task. AlexNet obtained an average accuracy score of 84.33% over all four disease recognition datasets, followed by Detection with 65.50% accuracy score. The margin between both networks was 18.83%. LeNet-5 obtained an average accuracy score of 50.60% over all four disease recognition datasets, recording the lowest performance. VGGNet and ZFNet both obtained average accuracy scores of 60.47% and 51.15% respectively.

In order to understand the consistently poor performance of LeNet-5, feature maps were employed. Feature maps, also known as activation maps, are the outputs of the layers in a Convolutional Neural Network (CNN). They are essentially the results of applying filters (also known as kernels or convolutional kernels) to the input data or the output of a previous layer. In a CNN, the input data (usually an image) is convolved with a set of filters. Each filter is a small matrix of weights, and these weights are learned during the training process. The filter is slid across the input data, and at each position, it performs an element-wise multiplication with the part of the input it currently covers, and all these multiplied values are summed up to get a single value. This operation is repeated for every position on the input, resulting in a feature map. This process is done for each filter in the layer, resulting in multiple feature maps for each input. Each filter is designed to detect a specific feature in the input, such as edges, corners, or more complex patterns in higher layers of the network [32]. The feature map is a spatial map that represents the areas in the input data where the features that the filter is trying to detect are found. For example, if a filter is designed to detect edges, the corresponding feature map will have high values in the positions where there are edges in the input and low values elsewhere. The values in the feature map are called activations, indicating the degree to which that feature is present at each location in the input [33]. After the convolutional layer, there is often a pooling layer, which reduces the spatial dimensions (width and height) of the feature maps while keeping the depth the same. This is done to reduce the computational complexity of the network and to provide a form of translation invariance. The most common type of pooling is max pooling, where the maximum value in a certain window on the feature map is taken as the representative value of that area (see Figure 4) [34]. As one moves deeper into the network, the feature maps represent more complex features. In the initial layers, the feature maps might represent simple features like edges and corners, but in deeper layers, they might represent more complex features like parts of objects (in the case of an image recognition task). This is because each layer's feature maps are a combination of the features from the previous layer's feature maps [35]. Visualizing feature maps can be a powerful tool for understanding what features a CNN has learned to recognize. This can be done by passing an input through the network, storing the feature maps, and then plotting them. This can give insights into what each layer in the network is doing and can also be useful for debugging the network [36]. Figure 4 presents feature maps for the first four (4) feature extraction layers of each feature extraction network. It can be observed from Figure 4 that instead of learning the low-level features first (as did the feature extraction layers for AlexNet, Detection, ZFNet and VGGNet), the first four feature extraction layers of LeNet-5 began learning the high-level features first, leading to the presentation of low-level features at the fully connected layers, resulting in low model performance.

In order to understand why the accuracy results were so, we ranked the networks according to different criteria (Table 2). It can be observed that on the basis of depth, number of convolutional layers and number of pooling layers, VGGNet which topped all of the criteria emerged third, while Detection which ranked fourth in all of the criteria surpassed VGGNet (Table 2). This suggested that the mere presence of a greater number of layers does not guarantee top performance. However, the order in which layers are stacked does indeed affect the performance of the model. The architecture of a CNN is typically designed with a specific order of layers to extract hierarchical features from the input data. (1) Convolutional Layers: These are the first layers in a CNN. They apply a convolution operation to the input, passing the result to the next layer. The first convolutional layers are generally designed to detect low-level features such as edges and corners, while deeper layers combine these to detect higher-level features such as shapes or objects. (2) Pooling Layers: These are usually placed after one or more convolutional layers. The pooling operation (often max pooling) reduces the spatial dimensions of the input, which helps to decrease the computational complexity of the network, provides a form of translation invariance, and helps the network learn more complex features. (3) Fully Connected Layers: These are typically placed at the end of the network, after several convolutional and pooling layers. They take the high-level features learned by the previous layers and use them to classify the input into various categories. (4) Normalization Layers: Batch normalization layers are often inserted after convolutional layers and before activation functions. They help to stabilize the learning process and reduce

the number of training epochs required to train deep networks. The order of these layers is not arbitrary. For example, placing a fully connected layer immediately after the input layer would not make sense in a CNN, because the spatial structure of the input would be lost. Similarly, placing a pooling layer before a convolutional layer would not be effective, because the pooling layer would reduce the resolution of the image before the convolutional layer had a chance to detect features. This hierarchical structure of CNNs, where low-level features are learned in the early layers and high-level features in the later layers, is a key factor in their success at tasks such as image classification [37].

Table 1. Feature extraction network performances (% accuracy) on four selected image classification datasets from PlantVillage

	Bacterial Spot (Bell Pepper)	Late Blight (Tomato)	Leaf Mold (Tomato)	Yellow Leaf Curl (Tomato)	Average Accuracy
AlexNet	77.80	79.60	86.50	93.40	84.33
LeNet-5	50.20	52.50	47.68	52.00	50.60
VGGNet	71.16	70.90	50.80	49.00	60.47
ZFNet	55.30	49.20	50.80	49.30	51.15
Detection	50.80	80.80	80.00	50.40	65.50

Table 2. Feature extraction network ranking according to different criteria.

	Criterion			
Rank	**Depth**	**Number of conv. layers**	**Number of pooling layers**	**Average accuracy**
First	VGGNet	VGGNet	VGGNet	AlexNet
Second	AlexNet	AlexNet	AlexNet	Detection
Third	ZFNet	ZFNet	ZFNet	VGGNet
Fourth	Detection	Detection	Detection	ZFNet
Fifth	LeNet-5	LeNet-5	LeNet-5	LeNet-5

.

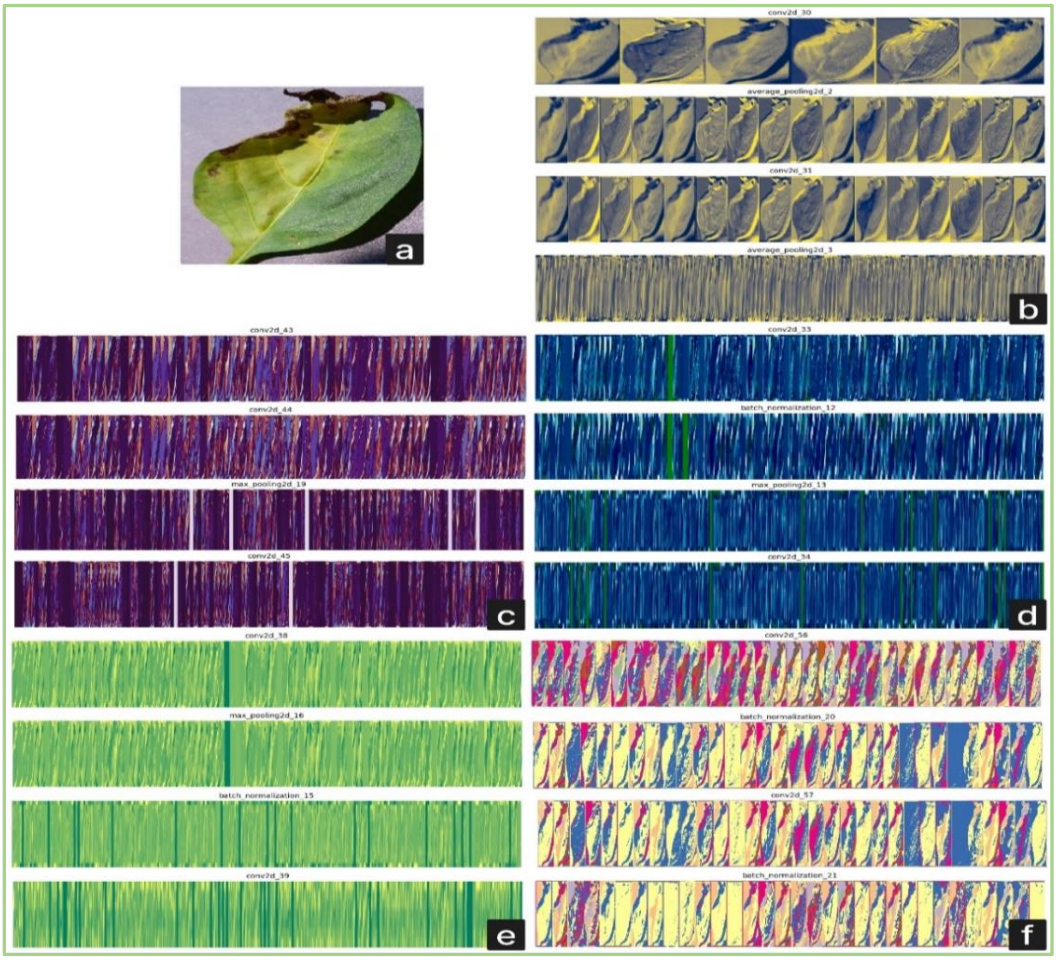

Fig. 4. Feature maps for the first four feature extraction layers of LeNet-5 (b), VGGNet (c), AlexNet (d), ZFNet (e) and Detection (f) ConvNet architectures. (a) is the data instance under processing.

Table 3. Ablation testing results on Detection (% Accuracy changes after removal of specific architectural components).

	Bacterial Spot (Bell Pepper)	Late Blight (Tomato)	Leaf Mold (Tomato)	Yellow Leaf Curl (Tomato)	Avg.
- Batch Norm (1)	58.50 (+7.70)	51.20 (-29.60)	48.40 (-31.60)	92.30 (+41.90)	62.60 (-2.90)
- Conv2D (2)	81.50 (+30.70)	89.80 (+9.00)	92.60 (+12.60)	94.40 (+44.00)	89.58 (+24.08)
- Batch Norm (2)	50.40 (-0.40)	50.00 (- 30.80)	72.40 (-7.60)	50.20 (-0.20)	55.80 (-9.70)
- Max Pool (1)	65.62 (+14.82)	50.78 (-30.02)	55.94 (-24.06)	47.77 (-2.63)	55.03 (-10.47)
- Conv2D (3)	64.90 (+14.10)	81.10 (+1.00)	75.70 (-4.30)	72.40 (+22.00)	73.53 (+8.03)
- Batch Norm (3)	49.20 (-1.60)	49.20 (-31.60)	82.60 (+2.60)	93.20 (+42.80)	68.55 (+3.05)
- Conv2D (4)	50.40 (-0.40)	88.40 (+7.60)	86.60 (+6.60)	94.00 (+43.60)	79.85 (+14.35)
- Batch Norm (4)	51.00 (+0.20)	50.00 (-30.80)	49.60 (-30.40)	49.80 (-0.60)	50.10 (-15.40)
- Max Pool (2)	82.00 (+31.20)	85.00 (+4.20)	92.30 (+12.30)	95.60 (+45.20)	88.73 (+23.23)

Ablation testing results

Ablation testing, also known as ablation analysis, is a strategy used in machine learning and deep learning to gain insights into the performance and functionality of a model. It involves systematically removing or altering components of a model to measure the impact on performance. This can help researchers understand the importance of different parts of the model and how they contribute to the overall performance [38]. The term "ablation" originally comes from biology and medicine, where it refers to the removal of body tissue. In the context of machine learning, it refers to the removal or alteration of parts of a model. The idea is to "ablate" a component and observe the effect on the model's performance. This can help identify which parts of the model are crucial for its performance and which parts have little or no effect. Ablation testing is important for several reasons: (1) Understanding Model Behavior: It helps in understanding the behavior of the model and the contribution of each component to the overall performance [39]. (2) Model Simplification: If a component does not contribute significantly to the model's performance, it might be possible to remove it, simplifying the model without a significant loss in performance [39]. (3) Debugging: If removing a component improves the model's performance, this might indicate a problem with that component. Ablation testing is typically performed by removing or altering one component of a model at a time and then measuring the effect on the model's performance. This is repeated for each component of the model. The components that might be ablated include features used as input to the model, layers in a neural network, or even individual neurons. Examples of how ablation testing might be used in deep learning include: (1) Feature Ablation: In a machine learning model, one might have a large number of features and want to understand which ones are most important. One could perform an ablation study by training the model multiple times, each time leaving out one of the features. The change in performance gives one an indication of the importance of each feature [40]. (2) Layer Ablation: In a deep learning model, one might want to understand the contribution of each layer to the model's performance. One could perform an ablation study by training the model multiple times, each time leaving out one of the layers. This can help understand the role of each layer in the model. While ablation testing can provide valuable insights, it also has some limitations. One limitation is that it can be computationally expensive, as it involves training the model multiple times. Another limitation is that it might not capture interactions between components. For example, removing one feature might not have a big effect on performance, but removing two features together might have a large effect [41]–[43]. The ablation testing results on the Detection feature extraction network are presented in Table 3. They show that the removal of the second convolutional layer from the network resulted in a 24.08% increase in overall accuracy on all four disease recognition datasets, up from 65.50% accuracy without the ablation. This surpassed AlexNet's overall accuracy on all four disease recognition datasets (84.33%) by 5.25%. Also, the removal of the second pooling layer from the network resulted in a 23.33% increase in overall accuracy on all four disease recognition datasets, up from 65.50% accuracy without the ablation. This also surpassed AlexNet's overall accuracy on all four disease recognition datasets (84.33%) by 4.40%.

CONCLUSION

The aim of this research was to propose a highly performant feature extraction ConvNet for plant disease recognition, called Detection. The results demonstrated that with ablation at two different layers, Detection succeeded to surpass the initially most-performant feature extraction convnet under comparison, AlexNet, by 5.25% and 4.40% respectively. Also, the order of the layers in any feature extraction ConvNet are more important that the number of the layers.

CONFLICT OF INTEREST

The authors declare that no competing interests exist.

AUTHORS CONTRIBUTIONS

This work was carried out in collaboration among all authors. All authors read and approved the final manuscript.

REFERENCES

1. Litjens G, Kooi T, Bejnordi BE, Setio AAA, Ciompi F, Ghafoorian M, et al. A survey on deep learning in medical image analysis. Med Image Anal. 2017;42:60–88. http://dx.doi.org/10.1016/j.media.2017.07.005

2. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017; http://dx.doi.org/10.48550/ARXIV.1704.04861
3. Geirhos R, Rubisch P, Michaelis C, Bethge M, Wichmann FA, Brendel W. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. 2018; http://dx.doi.org/10.48550/ARXIV.1811.12231
4. Zhang C, Bengio S, Hardt M, Recht B, Vinyals O. Understanding deep learning requires rethinking generalization. 2016; http://dx.doi.org/10.48550/ARXIV.1611.03530
5. Rudin C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell. 2019;1(5):206–15. http://dx.doi.org/10.1038/s42256-019-0048-x
6. Buolamwini J, Gebru T. Gender shades: Intersectional accuracy disparities in commercial gender classification. In: Friedler SA, Wilson C, editors. Proceedings of the 1st Conference on Fairness, Accountability and Transparency. PMLR; 23--24 Feb 2018. p. 77–91.
7. Schulz H, Behnke S. Deep learning: Layer-wise learning of feature hierarchies. KI - Künstl Intell. 2012 [cited 2023 Jul 8];26(4):357–63. https://www.deeplearningbook.org/
8. Izadkhah H. Deep Learning in Bioinformatics: Techniques and Applications in Practice. Elsevier Science;
9. Abbas A, Jain S, Gour M, Vankudothu S. Tomato plant disease detection using transfer learning with C-GAN synthetic images. Comput Electron Agric. 2021;187(106279):106279. http://dx.doi.org/10.1016/j.compag.2021.106279
10. Atila Ü, Uçar M, Akyol K, Uçar E. Plant leaf disease classification using EfficientNet deep learning model. Ecol Inform. 2021;61(101182):101182. http://dx.doi.org/10.1016/j.ecoinf.2020.101182
11. Al-Adhaileh MH, Verma A, Aldhyani THH, Koundal D. Potato blight detection using fine-tuned CNN architecture. Mathematics. 2023;11(6):1516. http://dx.doi.org/10.3390/math11061516
12. Albattah W, Nawaz M, Javed A, Masood M, Albahli S. A novel deep learning method for detection and classification of plant diseases. Complex Intell Syst. 2022;8(1):507–24. http://dx.doi.org/10.1007/s40747-021-00536-1
13. Devisurya V, Priya RD, Anitha N. Early detection of major diseases in turmeric plant using improved deep learning algorithm. Bull Pol Acad Sci Tech Sci. 70(2). http://dx.doi.org/10.24425/bpasts.2022.140689
14. Li Z, Zhou G, Hu Y, Chen A, Lu C, He M, et al. Maize leaf disease identification based on WG-MARNet. PLoS One 2022;17(4):e0267650. http://dx.doi.org/10.1371/journal.pone.0267650
15. Mohanty SP, Hughes DP, Salathé M. Using deep learning for image-based plant disease detection. Front Plant Sci. 2016;7:1419. http://dx.doi.org/10.3389/fpls.2016.01419
16. Ramcharan A, Baranowski K, McCloskey P, Ahmed B, Legg J, Hughes DP. Deep learning for image-based cassava disease detection. Front Plant Sci. 2017;8. http://dx.doi.org/10.3389/fpls.2017.01852
17. Reddy TV. Plant Disease Detection Using Advanced Convolutional Neural Networks with Region of Interest Awareness. Stem Cell Res Int. 2022;6(2):121–31.
18. Mohanty SP. PlantVillage-Dataset: Dataset of diseased plant leaf images and corresponding labels.
19. Hughes DP, Salathe M. An open access repository of images on plant health to enable the development of mobile disease diagnostics. 2015; http://dx.doi.org/10.48550/ARXIV.1511.08060
20. Maleki F, Ovens K, Gupta R, Reinhold C, Spatz A, Forghani R. Generalizability of machine learning models: Quantitative evaluation of three methodological pitfalls. 2022; http://dx.doi.org/10.48550/ARXIV.2202.01337
21. Bashir D, Montanez GD, Sehra S, Segura PS, Lauw J. An Information-Theoretic Perspective on Overfitting and Underfitting. 2020; http://dx.doi.org/10.48550/ARXIV.2010.06076
22. Yang Z, Yu Y, You C, Steinhardt J, Ma Y. Rethinking bias-variance trade-off for generalization of neural networks. 2020; http://dx.doi.org/10.48550/ARXIV.2002.11328
23. Bates S, Hastie T, Tibshirani R. Cross-validation: what does it estimate and how well does it do it? 2021; http://dx.doi.org/10.48550/ARXIV.2104.00673
24. Kukačka J, Golkov V, Cremers D. Regularization for deep learning: A taxonomy. 2017; http://dx.doi.org/10.48550/ARXIV.1710.10686
25. Hu X, Chu L, Pei J, Liu W, Bian J. Model complexity of deep learning: A survey. 2021; http://dx.doi.org/10.48550/ARXIV.2103.05127
26. Yang S, Xiao W, Zhang M, Guo S, Zhao J, Shen F. Image data augmentation for deep learning: A survey. 2022; http://dx.doi.org/10.48550/ARXIV.2204.08610
27. Takang E. detection: Acknowledging reliance on Habib Izadkhah's "Deep Learning in Bioinformatics" for the code relating to the feature extraction layers for ZFNet, AlexNet, LeNet-5 and VGGNet.
28. Nips.cc. [cited 2023 Jul 8]. https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
29. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE Inst Electr Electron Eng. 1998;86(11):2278–324. https://www.academia.edu/8932919/Gradient_Based_Learning_Applied_to_Document_Recognition
30. Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. 2013; http://dx.doi.org/10.48550/ARXIV.1311.2901
31. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv. 2014:1409.1556. https://ui.adsabs.harvard.edu/abs/2014arXiv1409.1556S/abstract?
32. Dumoulin V, Visin F. A guide to convolution arithmetic for deep learning. 2016; http://dx.doi.org/10.48550/ARXIV.1603.07285
33. Tang Y, Wang Y, Xu Y, Shi B, Xu C, Xu C, et al. Beyond dropout: Feature map distortion to regularize deep neural networks. 2020; http://dx.doi.org/10.48550/ARXIV.2002.11022
34. Gholamalinezhad H, Khosravi H. Pooling methods in Deep Neural Networks, a review. 2020; http://dx.doi.org/10.48550/ARXIV.2009.07485
35. Elharrouss O, Akbari Y, Almaadeed N, Al-Maadeed S. Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches. 2022; http://dx.doi.org/10.48550/ARXIV.2206.08016

36. Zhang X, Yin Z, Feng Y, Shi Q, Liu J, Chen Z. NeuralVis: Visualizing and interpreting deep learning models. 2019; http://dx.doi.org/10.48550/ARXIV.1906.00690
37. Yan Z, Zhang H, Piramuthu R, Jagadeesh V, DeCoste D, Di W, et al. HD-CNN: Hierarchical deep convolutional neural network for large scale visual recognition. 2014; Available from: http://dx.doi.org/10.48550/ARXIV.1410.0736
38. Meyes R, Lu M, de Puiseau CW, Meisen T. Ablation studies in artificial neural networks. 2019; http://dx.doi.org/10.48550/ARXIV.1901.08644
39. Hameed I, Sharpe S, Barcklow D, Au-Yeung J, Verma S, Huang J, et al. BASED-XAI: Breaking ablation studies down for explainable artificial intelligence. 2022; http://dx.doi.org/10.48550/ARXIV.2207.05566
40. Merrick L. Randomized Ablation Feature Importance. 2019; http://dx.doi.org/10.48550/ARXIV.1910.00174
41. Li M. Scaling distributed machine learning with the parameter server. In: Proceedings of the 2014 International Conference on Big Data Science and Computing. New York, NY, USA: ACM; 2014.
42. Molchanov P, Tyree S, Karras T, Aila T, Kautz J. Pruning convolutional neural networks for resource efficient inference. 2016; http://dx.doi.org/10.48550/ARXIV.1611.06440
43. Morcos AS, Barrett DGT, Rabinowitz NC, Botvinick M. On the importance of single directions for generalization. 2018; Available from: http://dx.doi.org/10.48550/ARXIV.1803.06959