International Journal of Innovative Research and Scientific Studies, 8(2) 2025, pages: 897-907

IJIRSS
International Journal of Innovative Research & Scientific Studies

ISSN: 2617-6548

URL: www.ijirss.com

Black pepper leaf disease detection using deep learning

Jagadeesha B G[1*], Ramesh Hegde[2], Ajith Padyana[3]

[1]*Department of Computer Science, The National College, Basavanagudi, Bengaluru, Karnataka-560004, India.*
[2]*Research Supervisor, Acharya Institute of Technology, Soladevanhalli, Bengaluru, Karnataka-560107, India.*
[3]*Department of Computer Science and Engineering, Acharya Institute of Technology, Soladevanhalli, Bengaluru, Karnataka-560107, India.*

Corresponding author: Jagadeesha B G1 (*Email: bgjsmg@gmail.com*)

Abstract

Advances in deep learning techniques have achieved spectacular success in the detection of plant diseases. A new method for detecting black pepper leaf disease using deep learning was proposed. In the proposed scheme, the SqueezeNet model is used, which is a Convolutional Neural Network (CNN), where the CNN is a subset of deep learning networks. The disease detection is based on the visual characteristics of the black pepper leaves. Thus, the proposed method is an image classification scheme using a trained SqueezeNet that detects whether the pepper leaves are healthy or diseased. The detection accuracy is found to be more than 99%. The early detection of defects, such as deformation and discoloration of pepper leaves, forewarns the onset of diseases, and the cultivator of pepper wines can undertake appropriate countermeasures.

Keywords: Black pepper diseases, Convolutional Neural Networks, Deep learning networks, Image data store, Leaf image classification, SqueezeNet.

DOI: 10.53894/ijirss.v8i2.5389
Funding: This study received no specific financial support.
History: Received: 27 January 2025 / **Revised:** 28 February 2025 / **Accepted:** 6 March 2025 / **Published:** 14 March 2025

Competing Interests: The authors declare that they have no competing interests.
Authors' Contributions: All authors contributed equally to the conception and design of the study. All authors have read and agreed to the published version of the manuscript.
Transparency: The authors confirm that the manuscript is an honest, accurate, and transparent account of the study; that no vital features of the study have been omitted; and that any discrepancies from the study as planned have been explained. This study followed all ethical practices during writing.
Publisher: Innovative Research Publishing

1. Introduction

In botanical and agricultural sciences, the health of a plant can be determined by the visual observation of its leaves. Any abnormalities in the texture, color, and shape of the leaves indicate the presence of some disease in those plants. Several conventional image processing methods are available to distinguish healthy images from diseased ones. Conventional methods extract the characteristic features of the target image and match them with the knowledge-based features of healthy/diseased images to arrive at a decision. The major drawbacks are computationally expensive feature extraction algorithms, imperfect knowledge bases, and inaccuracies due to noise, distortion, etc. On the other hand, it has

been well established that a powerful, well-trained Convolutional Neural Network (CNN) can accurately predict the presence of diseased leaf images from healthy ones in a set of mixed images, even in the presence of relatively large noise and distortion. Therefore, it is not difficult to build and train a CNN to detect abnormalities in the texture, coloration, and shape of plant leaves based on their images. Once it is known that a plant (or creeper, as in the case of black pepper) is afflicted with a disease, its type, degree of severity, etc., can be further determined using more powerful CNNs capable of multi-object classification. The visual differences between healthy and diseased pepper leaf images are shown in Figure 1.

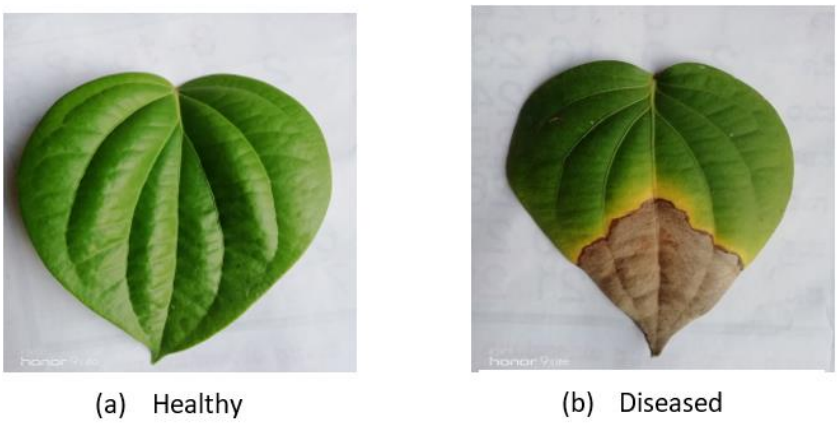

Figure 1.
Healthy and diseased pepper leaf images.

Due to technological advancements, several sophisticated software CNN models and tools are available [1],which are well-tested, debugged, and built-in models from well-known academic institutions and information technology companies. Some of them are LeNet, AlexNet, VGG16, ResNet50, GoogleNet, Inception V3, SqueezeNet, and so on. In this study, we used SqueezeNet CNN for black pepper leaf image classification.

1.1. SqueezeNet

Although several CNN architectures [1] are available to achieve a given level of accuracy, they are burdened by a very large number of parameters, which results in a huge storage requirement in the range of 20–500 MB (approx.). On the other hand, Squeezenet [2] is designed with fewer parameters for the same level of accuracy, which results in a reduced memory space of approximately 5 MB. It has lower computational complexity with faster execution of the training process. The advantages of smaller size are:

- Lower Computational overhead for training and testing. Different training parameters can be optimally chosen for shorter training periods with higher accuracy.
- Cost of exporting trained models. Due to the reduced size of the model, export cost is relatively low.
- Compact FPGA hardware realization.

Squeezenet (SN) is a supervised learning model with backpropagation. Squeezenet is typically designed for classification instead of regression. In PLC-SN, Squeezenet is used as the binary classifier, where the output is the class name, either 1 (healthy) or 2 (diseased).

1.2. Main Objective

The main objective of this paper is to use a CNN formed by SqueezeNet (SN) for classifying the images of black pepper leaves to detect disease-afflicted pepper creepers from healthy ones. The proposed method is denoted as Pepper Leaf Classification using SqueezeNet (PLC-SN), which is a binary (two-level) classifier as it detects whether the input image is healthy or diseased.

1.3. Organization of the Paper

The remaining part of the paper is organized as follows. Section 2 briefly reviews the existing works on image classification of plant leaves using CNNs. In Section 3, the basic workings of PLC-CNN are presented. Section 4 contains the performance evaluation of PLC-SN based on simulated experimental results. Section 5 provides the conclusion.

2. Literature Review

In this section, only those papers related to plant leaf image classification based on ANNs and CNNs are reviewed briefly. In Bhadur and Rani [3], the authors have reviewed the application of CNNs for the detection of diseases in plants based on their leaf images. Various layers of CNN and their functions are described in the context of leaf image classification. Reviews of existing works in detecting diseases in various agricultural/fruit plants, such as maize, apple, tomato, potato, etc., have been presented in detail. In Taterwal [4], pepper leaf image classification for disease detection has been presented. The author has described the custom-designed CNN, based on a three-block VGGnet, to suit the classification of pepper leaf images. Standard libraries related to machine learning have been used to build the custom CNN. The author has compared the proposed method with KNN (K-Nearest Neighbor algorithm), Naive Bayes Algorithm,

Decision Tree Algorithm, and Logistic Regression Algorithm. The disadvantage of this work is the large size of the VGG net, which prohibits its application to mobiles and FPGA/ASICs. In Liu, et al. [5], the authors have used a modified AlexNet architecture in combination with parts of GoogleNet's inception model to detect unhealthy apple leaf images. From a relatively small set of unhealthy images, a larger set is derived using image rotation, brightness control, and contrast enhancement. The proposed method is compared with AlexNet, GoogLeNet, ResNet-20, and VGGNet-16 and is found to be superior to them by achieving an accuracy of about 97%. In Lin, et al. [6], a matrix-based convolutional neural network (M-bCNN) has been proposed for fine-grained classification of wheat leaf images. The authors have claimed that the accuracy attained by M-bCNN is about 91%. Here, the matrix-based architecture has a larger size and higher computational complexity. However, the accuracy achieved is less than the achievable accuracy of 98-99%.

In Shijie, et al. [7], the authors have used a combination of the VGG16 network and transfer learning to classify tomato plant diseases by examining their leaf images. The method achieves an accuracy of about 89%, which is relatively low. In Prajwala, et al. [8], LeNet CNN is used for tomato leaf disease detection by implementing 10-level classifications. The classification algorithm is found to have an accuracy of 94.85%.

In Rangarajan, et al. [9], pre-trained VGG16 and AlexNet are used based on transfer learning to classify tomato leaf images. Here, the output is designed to detect seven distinct disease classes. The AlexNet model achieves 97.49% accuracy, while VGG16 achieves 97.23%. In Ma, et al. [10], the authors have designed a deep learning CNN for cucumber leaf image classification to recognize four types of cucumber diseases, namely anthracnose, downy mildew, powdery mildew, and target leaf spots. The classification accuracy of the deep CNN is found to be 93.4%.

In Dai, et al. [11], a study on the recognition of pepper leaf diseases using enhanced lightweight convolutional neural networks (CNNs), based on the GoogLeNet architecture, has been presented. The enhanced lightweight CNN model is designed to be computationally efficient, making it suitable for deployment on devices with limited resources, such as mobile or embedded systems. Here, six types of pepper leaf classification are presented that achieve an accuracy of 97.87.

In Mathew and Mahesh [12], the article focuses on the detection of diseases in bell pepper plants using YOLO (You Only Look Once) v5, a state-of-the-art object detection algorithm. The study provides improved accuracy and speed of disease detection in bell pepper leaves, which is critical for early intervention and effective crop management. YOLO v5 is chosen for its efficiency and ability to process images quickly, making it suitable for real-time applications.

In Alatawi, et al. [13], the application of the VGG-16 deep learning model for detecting plant diseases is explored by classifying the corresponding plant leaf images. This study highlights the potential of artificial intelligence (AI) in agriculture, particularly for the early and accurate identification of plant diseases, which is crucial for improving crop yield and reducing losses. The model has used 19 different classes of plant diseases and achieved a classification accuracy of 95.2%.

In Atila, et al. [14], the authors have used the EfficientNet deep learning model for classifying plant leaf diseases. The model was trained using the PlantVillage dataset. In this binary classification, a transfer learning technique is adopted to achieve an accuracy of 99.37%.

In Bezabh, et al. [15], the study presents a novel approach for classifying pepper diseases using deep learning techniques based on a concatenated convolutional neural network (CCNN) model, termed CPD-CCNN, which combines multiple CNN architectures to improve the accuracy of disease classification in pepper plants. Here, VGG16 and AlexNet are concatenated with suitable modifications to achieve an accuracy of 95.82%.

In Kini, et al. [16], the authors explore the application of transfer learning with convolutional neural networks (ConvNets) for the early detection of black pepper leaf diseases. The study focuses on leveraging pre-trained deep learning models to improve the accuracy and efficiency of disease prediction at an early stage, which is critical for effective crop management. Here, five types of black pepper diseases are classified to achieve an accuracy of 99.67%.

In all the above schemes, accuracy greater than 99.7% has not been reached, whereas in our proposed PLC-SN, the training accuracy is 100% and the classification (detection) accuracy is more than 99.7%. This high accuracy is achieved using a special way of dataset augmentation which is explained later in section 3. F.

3. Methodology

3.1. Architecture of SqueezeNet

SqueezeNet (SN) is a lightweight convolutional neural network architecture designed to achieve AlexNet-level accuracy on image classification tasks while using significantly fewer parameters. Its primary goal is to provide a smaller model with reduced computational and memory requirements, making it well-suited for resource-constrained environments like mobile and embedded systems. The key features of the SN architecture are as follows.

3.1.1. Fire Modules

The building block of SqueezeNet is the Fire Module, which consists of two main layers:

- Squeeze Layer: A 1x1 convolutional layer that reduces the number of input channels.
- Expand Layer: A combination of 1x1 and 3x3 convolutional filters that expands the feature maps after the squeeze layer.

This design reduces the computational cost by performing expensive 3x3 convolutions on fewer input channels.

3.1.2. Global Average Pooling

Instead of fully connected layers, SqueezeNet uses global average pooling at the end of the network. This reduces the number of parameters and prevents overfitting.

3.1.3. Parameter Reduction Strategies

- 1x1 Convolutions: Reduce the depth of feature maps, thereby decreasing the number of parameters.
- Delayed Downsampling: Downsampling (via max pooling) is postponed until later in the network, allowing the convolutional layers to operate on larger feature maps for better feature extraction.

3.1.4. Different Layers of SqueezeNet

SN has 68 layers in total and it is described as follows.

1. Input Layer:
   Input image size: 256×256×3
   (modified to match the image size)
2. Initial Convolution Layer:
   96 filters of size 7×7 with stride 2 (conv1)
   Followed by ReLU activation and max pooling (3×3, stride 2)
3. Fire Modules:
   Fire Modules 2 through 9, grouped as follows:
   Fire2 to Fire4
   Max pooling layer (3×3, stride 2)
   Fire5 to Fire8
   Max pooling layer (3× 3, stride 2)
   Fire9
4. Final Layers:
   Convolution layer with 2 filters (for binary Image classes).
   Global average pooling layer.
   Softmax activation for classification

The first layer is the "Image Input Layer" that accepts the input image for classification. The last layer is the "Classification Layer". The functions of different layers and the working of SN are given in Iandola, et al. [2].

3.2. Dataset Organization

In Machine Learning (ML), especially in tasks like image classification, object detection, or segmentation, image files are stored in disk folders rather than directly in memory for the following reasons.

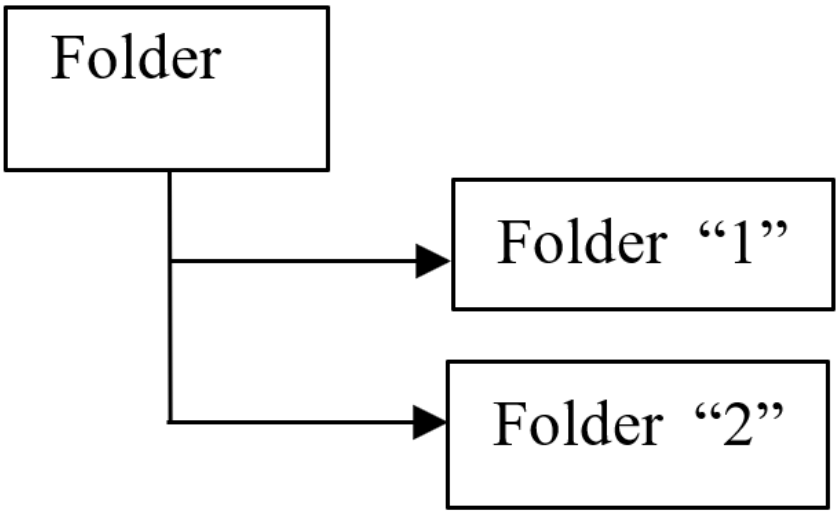

Figure 2.
imds folder structure.

Images can take up significant amounts of memory, especially if the dataset is large. Storing all images in memory (RAM) can quickly exceed the capacity of most systems. By storing images on disk, they can be loaded into memory only when needed during training or inference, reducing memory usage. Images stored in disk folders can be easily accessed and shared across distributed systems or cloud storage solutions. Preprocessing operations on images, like scaling, filtering, rotation, etc., are supported by Matlab or Python, before loading the images into the ML model.

In MATLAB, the image data are stored in the imds (Image Data Store) [17] folder with two sub-folders named "1" and "2." The RGB leaf images are of size 256×256×3. The status of the leaf images is determined by an expert after careful examination. The healthy leaf images are stored in "1" with label values = 1, and the diseased images in "2" with label

values = 2. The folder structure is shown in Figure 2. In general, the number of healthy images in folder "1" is equal to the number of diseased images in folder "2."

The dataset is gathered by taking the photograph of leaf images directly by visiting the plantation sites and classified by the experts from the local horticulture organization.

3.3. Preprocessing of Leaf Images

In PLC-SN, each image, either healthy or diseased, is preprocessed as follows.

Let *A* be an original leaf image matrix of size 256×256×3 as shown in Figure 3.

Figure 3.
Leaf Image ***A*** of size 256×256×3.

Then, A is flipped left-right using the fliplr(A) function. The flipped image is horizontally concatenated with A to get B as,

Figure 4.
Leaf image B of size 256×512×3.

$$B = [A, \text{fliplr}(A)] \quad (1)$$

Image ***B*** is shown in Figure 4.

The size of *B* is 256×512×3. Now, *B* is flipped up-down to obtain *C* as *C* = flipud(*B*). Then, the composite image matrix *D* is obtained as the vertical concatenation of *B* and *C* as,

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{C} \end{bmatrix} = \begin{bmatrix} \boldsymbol{B} \\ \text{flipud}(\boldsymbol{B}) \end{bmatrix} \quad (2)$$

The composite image matrix ***D*** is shown in Figure 5.

Now, the height of ***D*** is the double of that of ***B***, and the size of ***D*** is 512×512×3. From Figure 5, we observe that the preprocessing of ***A*** generates a symmetric image ***D***. This preprocessing operation (flip and concatenate) is applied to all the images of the image data store (imds), including folders “1”, and “2”. The preprocessed collection image ***D***’s of size 512×512×3 form the dataset as,

$$CD = [\, D\{1\}, D(2\}, \ldots, D\{k\}, \ldots, D\{K\}] \quad (3)$$

Here, k varies from 1 to K, where K is the total number of images in imds. In PLC-SC, we have 500 healthy leaf images and 500 diseased images, making a total of 1000 images. Thus, K = 1000. After a few trials and errors, it was found that the composite images, D{k}’s, perform better as inputs to the SN than the original A’s. Now, the collection CD acts as the effective imds that forms the overall dataset.

3.4. Training, Validation, and Test Data Image Matrices

For the correct operation of the SN, the total number of images should be split into 3 sub-groups as Train, Validation, and Test datasets. This operation is carried out using the splitEachLabel(…) function as,

[imdsTrain, imdsVal,imdsTest] = splitEachLabel(imds, 0.7,0.15,0.15,'randomized'); (4)

The argument, 'randomized' means, the split operation randomizes the order of images, including their labels, stored in imds. Equation specifies that in the splitting operation, the standard split ratios is used as,

imdsTrain:imdsVal:imdsTest=[0.7,0.15,0.15] (5)

Since the total size is K =1000, allocations for the images are,

imdsTrain=700, imdsVal=150, imdsTest=150 (6)

When the overall dataset is split, the corresponding labels are also split accordingly.

3.5. Working of the Squeezenet

The Squeezenet is constructed as described in section 3.A. The Squeezenet classifier works in three stages.

- Training Phase
- Test Phase
- Application Phase

The block diagram of the training phase followed by the test phase is shown in Figure 6.

3.5.1. Training Phase

A PLC-SN is built based on supervised training. In Figure 6, imdsTrain is the training input, that is a subset of imds as indicated by (4).

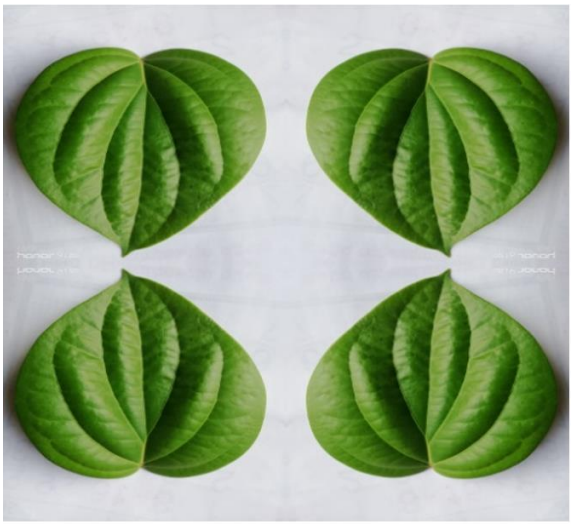

Figure 5.
Composite image *D*.

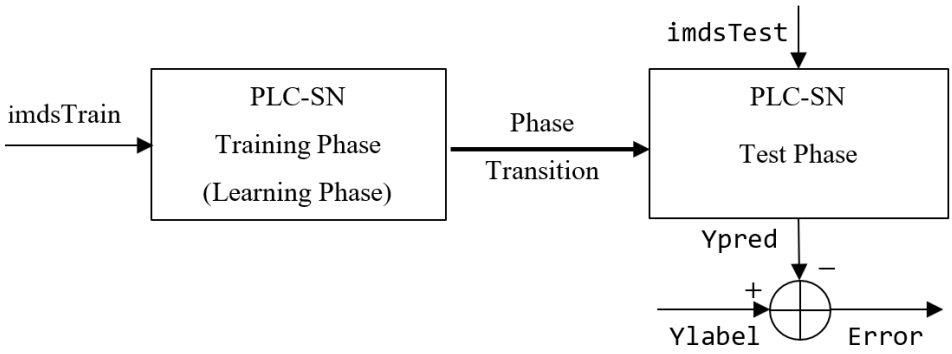

Figure 6.
Basic Block diagram of a PLC-SN Classifier.

In MATLAB, imdsTrain is a structure that contains the images and their respective labels. During the training process, the sequence of training matrices from imdsTrain and the matching labels are applied to the PLC-SN. The training process involves multiple iterations, and the PLC-SN model learns the complex relationship between the training images and their labels. That is, during the training, the model learns the functional relationship by successively adjusting its internal parameters based on the backward propagation principle.

3.6. Training Options

The training of the SN is carried out using suitable training option. The training option used in PLC-SN is as follows.

```
options = trainingOptions('sgdm', ...
'InitialLearnRate',0.001, ...
'MaxEpochs',20, ...
'Shuffle','every-epoch', ...
'LearnRateDropFactor',0.1, ...
'LearnRateDropPeriod',20, ...
'ValidationData',imdsVal, ...
'ValidationFrequency',20, ...
'MiniBatchSize',30, ...
'Verbose',true,...
'Plots','training-progress');                    (7)
```

The descriptions of different terms used in (7) are given in reference [2]. The numerical values are decided by conducting several trials and fine tuning the parameters as shown by the flowchart of Figure 7. With inputs imdsTrain,layers,options ready the training process is started by,

```
net =trainNetwork(imdsTrain,layers,options)    (8)
```

After training with given number of epochs, testing is carried out to find the *PTE*. If PTE is greater than the specified error threshold *ET*, say 0.5%, then the hyper parameters are readjusted and fine-tuned further until $PTE \leq ET$. Adjustment of hyper parameters, by fine tuning process, is shown in the flowchart of Figure 7.

3.6.1. Validation

Validation [18] in machine learning refers to the process of assessing a model's performance on the imdsVal dataset, which is a subset of imds. The goal of validation is to tune the model's hyperparameters and prevent overfitting while ensuring that it generalizes well to unseen data.

3.6.2. Test Phase

Once the training phase is fully completed, the PLC-SN enters the test phase, as shown in Figure 6. During the test phase, the trained PLC-SN accepts the test input matrices imdsTest and predicts the label output vector Ypred. The ground truth values, stored in vector form and denoted by Ylabel, are given by,

```
Ylabel = imdsTest.Labels   (9)
```

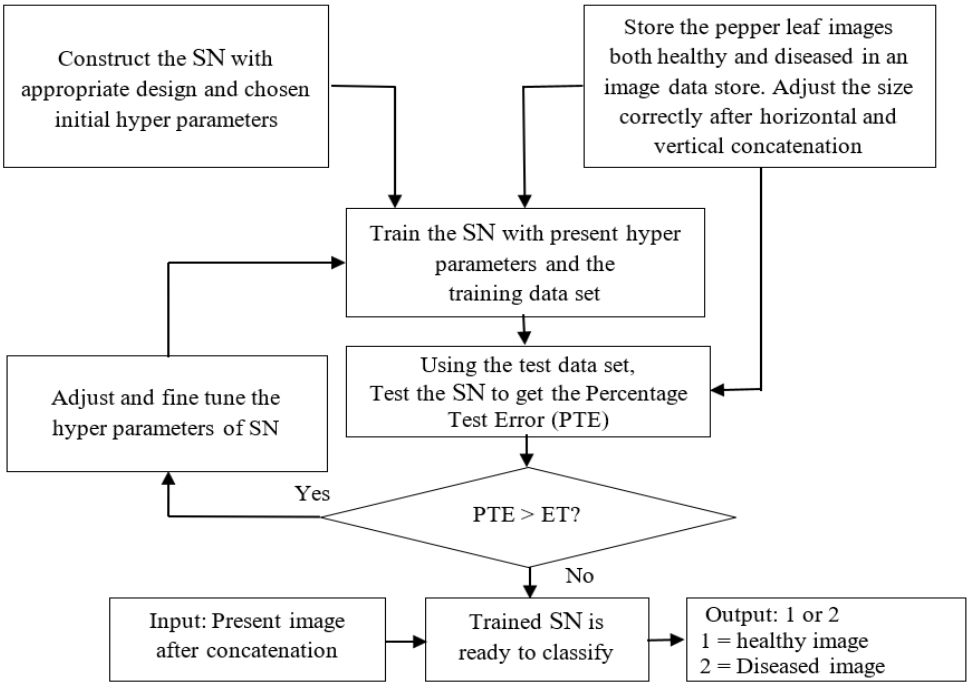

Figure 7.
Flow chart for fine tuning SqueezeNet

The test error vector is given by,

```
Error = Ylabel - Ypred                    (10)
```

With perfect training, Error vector should be all zeros. The non-zero terms of Error represent the Test Error (TE). Therefore, TE is given by,

TE = number of non-zero terms in Error(11)

Therefore, the Percentage Test Error (*PTE*) is given by,

$$PTE = \frac{TE*100}{length(Error)} \quad (12)$$

The PTE normalizes the error with respect to the length of the Error, so that the comparison of overall errors for different values of length(Error) remains fair. In PLC-SN, length(Error)= 150.

3.6.3.Application Phase

Once the training and testing are over with almost negligible PTE, the trained CNN is ready for the classification of a new leaf image in its composite form (flip and concatenate). The output of the CNN predicts the class (either 1 or 2) of this new image. The time taken for the classification task is very small compared to the training time.

4. Results and Discussion

Experiment 1: The SN is constructed as explained in Section 3. A. The training options are chosen as given by (6). The training process is executed using (7). The resulting training progress Figure are shown in Figure 8.

Training on single GPU.
Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:06	56.25%	74.83%	0.9955	0.5679	0.0010
1	10	00:00:08	87.50%	85.91%	0.4055	0.2983	0.0010
1	20	00:00:11	100.00%	97.99%	0.0689	0.0678	0.0010
1	30	00:00:13	93.75%	98.32%	1.3391	0.0583	0.0010
1	40	00:00:15	100.00%	98.66%	0.0587	0.0474	0.0010
2	50	00:00:17	100.00%	100.00%	0.0208	0.0211	0.0010
2	60	00:00:19	100.00%	100.00%	0.0018	0.0041	0.0010
2	70	00:00:21	100.00%	99.33%	0.0034	0.0153	0.0010
2	80	00:00:23	96.88%	100.00%	0.0578	0.0038	0.0010
3	90	00:00:24	100.00%	100.00%	0.0017	0.0039	0.0010
3	100	00:00:26	100.00%	100.00%	0.0022	0.0061	0.0010
3	110	00:00:28	100.00%	100.00%	0.0008	0.0036	0.0010
3	120	00:00:30	100.00%	100.00%	0.0120	0.0084	0.0010
4	130	00:00:32	100.00%	100.00%	0.0172	0.0010	0.0010
4	140	00:00:34	100.00%	100.00%	0.0005	0.0007	0.0010
4	150	00:00:36	100.00%	100.00%	0.0001	0.0007	0.0010
4	160	00:00:38	100.00%	100.00%	0.0005	0.0009	0.0010
4	170	00:00:40	100.00%	100.00%	0.0001	0.0009	0.0010
5	180	00:00:42	100.00%	100.00%	0.0024	0.0005	0.0010
5	190	00:00:44	100.00%	100.00%	6.0428e-05	0.0005	0.0010
5	200	00:00:46	100.00%	100.00%	3.8948e-05	0.0006	0.0010
5	210	00:00:48	100.00%	100.00%	0.0009	0.0005	0.0010
5	215	00:00:49	100.00%	100.00%	4.3858e-05	0.0005	0.0010

Training finished: Max epochs completed.

Figure 8.
Training and test accuracy have reached 100% with Training, validation and test set are of size 700, 150, 150

From Figure 8, it can be seen that both the training (mini-batch) accuracy and validation accuracies are 100% each. On testing, it is found that the PTE = 0%. The corresponding training progress plots are shown in Figure 9.

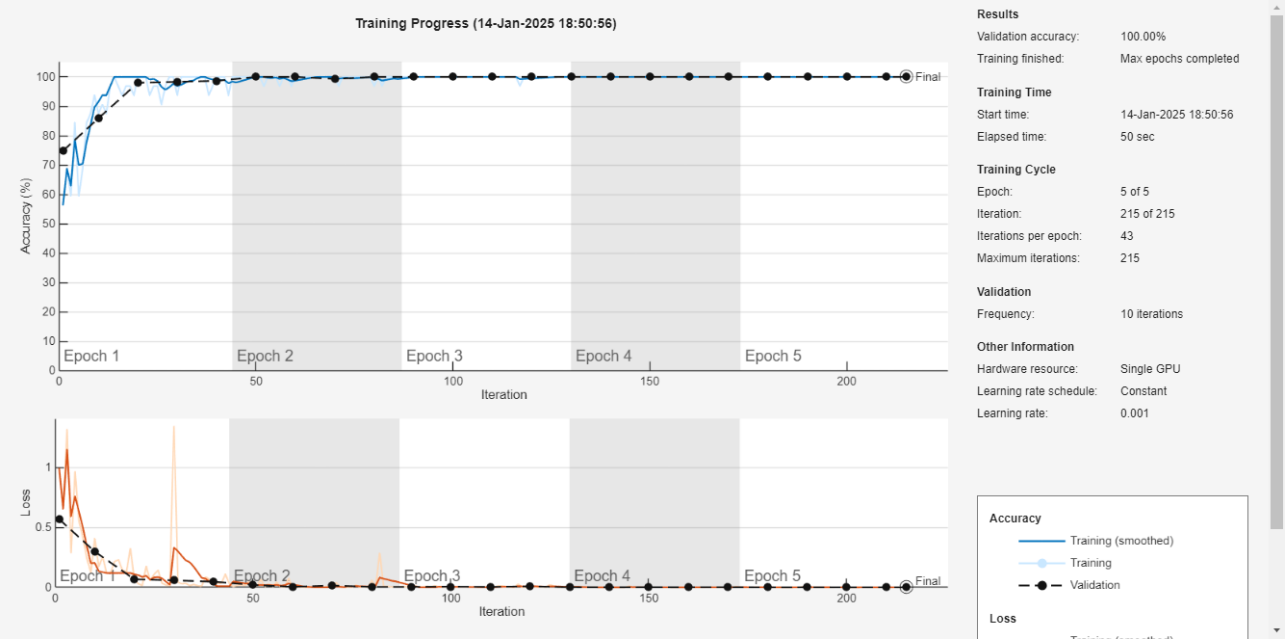

Figure 9.
Training Progress Plots with training and validation accuracy of 100%.

Experiment 2: When the number of epochs, denoted by *n_epoch*, is high, the error will be zero (as in Experiment 1). When *n_epoch* is decreased, the error gets increased. Let us denote the size of the training set as *R*. When *R* is changed the overall data size and the validation data size are also changed according to (5). It can also be observed that *R* decreases, and then also, the error increases. That means increases in *n_epoch* as well as R, cause a decrease in the error. In this experiment, Then_ephoc is increased from 10 to 24 in steps of 2 for each of the three values of R as, R = 400, 360, and 320. The resulting *PTE*'s are calculated. All other parameters are kept same as in Experiment 1. The resulting plots, *PTE* versus *n_epoch* for R = 400, 360, and 320 are shown in Figure 10.

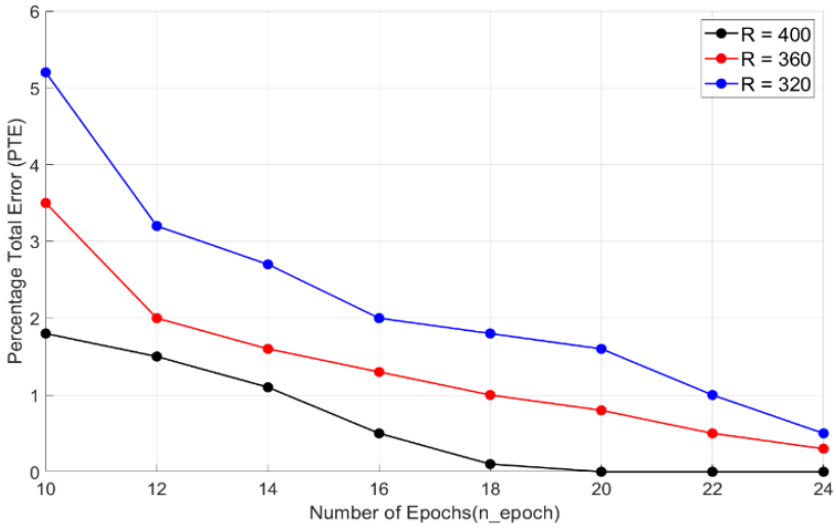

Figure 10.
PTE versus number of epochs.

From Figure 10, it can be seen that as *n-epoch* increases, the error values converge to low error levels for all three values of R, 400, 360, and 320.

Experiment 3: The distribution TE, the total number of classification errors as given by (11), over individual classes are pictorially represented by the Confusion Matrix chart. In this experiment,n_epoch is set low at 5 to increase the error to see the distribution of errors over a wider range. With R = 400 (with validation and test sizes adjusted according to (5), the TE value is found to be 18. The Confusion Matrix chart for this experiment is shown in Figure 11.

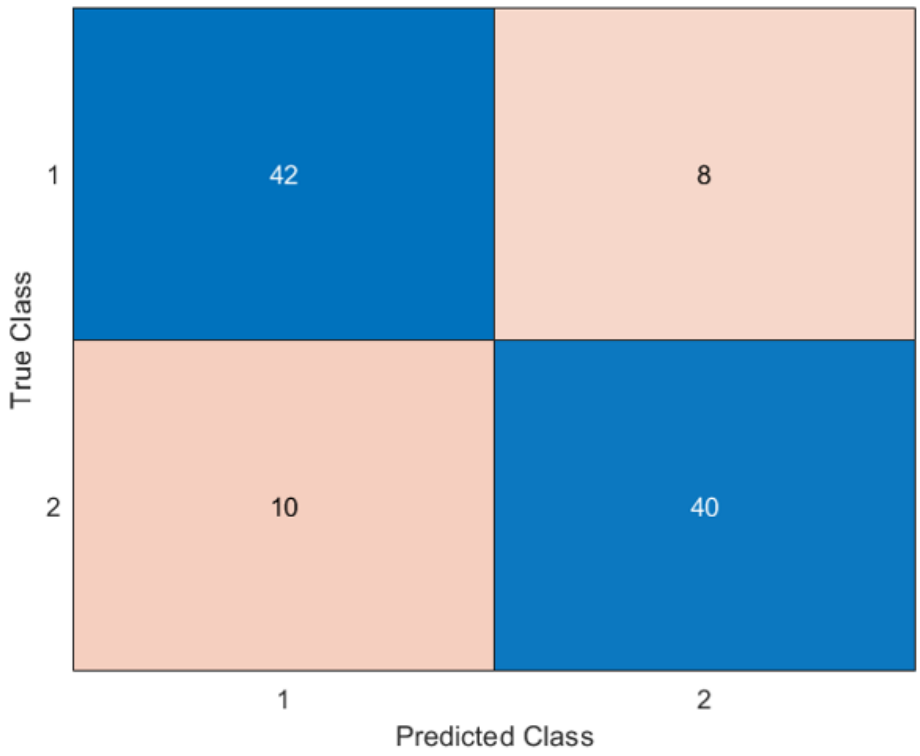

Figure 11.
Confusion Matrix chart for Experiment 3.

Here, the true positive (TP) is found to be 42, true negative (TN) = 40, false negative (FN) = 8 and false positive (FP) = 10. From the confusion matrix, we see that the trend to detect the false positive is slightly higher than to detect the false negative.

5. Conclusion

A new method of black pepper creeper disease detection based on the deep learning convolutional neural network is presented. In this method, the standard SqueezeNet is slightly modified and trained to classify the pepper leaf images as diseased or healthy. The images in the dataset are concatenated horizontally as well as vertically with the corresponding left-right flipped and up-down flipped counterparts. The use of these concatenated images for training is found to increase the image classification efficiency of the SqueezeNet to 100%. This method of using doubly concatenated images to enhance the training accuracy, especially when the availability of the training dataset is scarce, is our original contribution.

References

[1] OpenGenus, "Different types of CNN models," Retrieved: https://iq.opengenus.org/different-types-of-cnn-models/. [Accessed January 13, 2024], 2025.

[2] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016. https://arxiv.org/abs/1602.07360

[3] G. Bhadur and R. Rani, "Agricultural crops disease identification and classification through leaf images using machine learning and deep learning technique: A review," presented at the Proceedings of the International Conference on Innovative Computing & Communications (ICICC), 2020.

[4] D. Taterwal, *Detection of pepper leaves diseases using CNN and Machine learning algorithms*. United States: Graduate Project, California State University San Marcos, 2021.

[5] B. Liu, Y. Zhang, D. He, and Y. Li, "Identification of apple leaf diseases based on deep convolutional neural networks," *Symmetry*, vol. 10, no. 1, pp. 1-16, 2017. https://doi.org/10.3390/sym10010011

[6] Z. Lin *et al.*, "A unified matrix-based convolutional neural network for fine-grained image classification of wheat leaf diseases," *IEEE Access*, vol. 7, pp. 11570-11590, 2019.

[7] J. Shijie, J. Peiyi, H. Siping, and S. Haibo, "Automatic detection of tomato diseases and pests based on leaf images," presented at the 2017 Chinese Automation Congress (CAC), 2017.

[8] T. M. Prajwala, A. Pranathi, K. SaiAshritha, N. B. Chittaragi, and S. G. Koolagudi, "Tomato leaf disease detection using convolutional neural networks," presented at the 2018 Eleventh International Conference on Contemporary Computing (IC3), 2018.

[9] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, "Tomato crop disease classification using pre-trained deep learning algorithm," *Procedia Computer Science*, vol. 133, pp. 1040-1047, 2018. https://doi.org/10.1016/j.procs.2018.07.070

[10] J. Ma, K. Du, F. Zheng, L. Zhang, Z. Gong, and Z. Sun, "A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network," *Computers and Electronics in Agriculture*, vol. 154, pp. 18-24, 2018. https://doi.org/10.1016/j.compag.2018.08.048

[11] M. Dai *et al.*, "Pepper leaf disease recognition based on enhanced lightweight convolutional neural networks," *Frontiers in Plant Science*, vol. 14, p. 1230886, 2023. https://doi.org/10.3389/fpls.2023.1230886

[12] M. P. Mathew and T. Y. Mahesh, "Leaf-based disease detection in bell pepper plant using YOLO v5," *Signal, Image and Video Processing*, pp. 1-7, 2022. https://doi.org/10.1007/s11760-021-02024-y

[13] A. A. Alatawi, S. M. Alomani, N. I. Alhawiti, and M. Ayaz, "Plant disease detection using AI based VGG-16 model," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, pp. 1-11, 2022. http://dx.doi.org/10.14569/IJACSA.2022.0130484

[14] Ü. Atila, M. Uçar, K. Akyol, and E. Uçar, "Plant leaf disease classification using EfficientNet deep learning model," *Ecological Informatics*, vol. 61, p. 101182, 2021. https://doi.org/10.1016/j.ecoinf.2020.101182

[15] Y. A. Bezabh, A. O. Salau, B. M. Abuhayi, A. A. Mussa, and A. M. Ayalew, "CPD-CCNN: Classification of pepper disease using a concatenation of convolutional neural network models," *Scientific Reports*, vol. 13, no. 1, p. 15581, 2023. https://doi.org/10.1038/s41598-023-42843-2

[16] A. S. Kini, K. Prema, and S. N. Pai, "Early stage black pepper leaf disease prediction based on transfer learning using ConvNets," *Scientific Reports*, vol. 14, no. 1, p. 1404, 2024. https://doi.org/10.1038/s41598-024-51884-0

[17] MathWorks, "Matlab.io.datastore.ImageDatastore," Retrieved: https://in.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html. [Accessed January 11, 2025], 2025.

[18] B. Jason, "What is the difference between test and validation Datasets?," Retrieved: https://machine-learningmastery.com/difference-test-valid -ation-datasets /. [Accessed 11-01-2025], 2025.