Plant Leaf Diseases Detection Using Lightweight Convolutional Neural Networks

Mohammed Mahmud
Computer Science Department
Southern Connecticut State University
New Haven, Connecticut
mahmudm2@southernct.edu

***Abstract*—The greater use of technology in agriculture has enabled food production to fulfill the requirements of the world's 8 billion people. This was inadequate, however, since plant cultivation presents several challenges. For instance, climate change, pollution, plant disease, and many other variables all endanger the planet's food security. Plant disease is among these factors. Damages from plant diseases in Georgia were anticipated to exceed $820 million. Due to the variety of diseases and farmers' lack of knowledge, various unsuccessful efforts have been attempted to avoid crop loss due to conditions. Consequently, it is crucial to develop rapid and accurate procedures for the early identification and classification of plant diseases. This paper offers a lightweight deep learning model based on Plant Village data. The proposed model focuses on ten tomato diseases. The model was developed using a database of 18,160 photographs of tomato leaf surfaces. The trained model has an accuracy rating of 98.08 percent. Farmers may find the proposed practical approach for identifying and classifying damaged tomato crops.**

***Index Terms*—Classification, Plants Disease, Perceptron, Multilayer Perceptron, Support Vector Machine, Artificial Neural Network, Deep learning, Convolutional Neural Network.**

I. Introduction

Plant disease contributes significantly to human survival. We humans depend on plants that we are ignorant of most of the time. Every day, at every meal, a significant portion of our nourishment comes from plants. Diverse plant species produce fifty percent of the oxygen that humans breathe. [1], [2] Plant diseases have grown commonplace. Every year, regardless of the weather, there will be complex diseases. The illness may vary from year to year. The crops of farmers are harmed and diminished if these illnesses are not swiftly controlled or avoided. When farmers' crops are damaged, they incur monetary losses. Farmers seek to overprice their goods on the market, which has a detrimental impact on our economy and results in alterations. It connects each of these variables in a chain. Globally, plant disease has an economic impact and a detrimental effect on agricultural productivity. Many countries' economies rely heavily on agriculture. Several nations, like Liberia (76.9%), Somalia (60.2%), and Guinea-Bissau (55.8%), derive more than half of their gross domestic product from the agricultural output. [3], [4] Fungi and fungi-like organisms cause 85 % of plant illnesses. The other 15% of infections are caused by bacteria, viruses, viroids, or different types induced by certain nematodes. [5], [6] To avoid these illnesses manually, the farmer must spend a lot on a consultant. It takes time for a farmer and an expert to examine each plant in a field. A farmer is not required to be knowledgeable about every plant disease and its prevention. Diseases fluctuate annually and seasonally. Sometimes it is challenging to keep up with it since its symptoms and patterns vary.

Urgent need for a lightweight plant disease detection model that can operate on a smartphone, low-powered PC, or low-powered drone within its barrier restrictions. To manually prevent this disease, the farmer will need to spend a substantial amount of money on a consultant. Time is required for both the farmer and the expert to examine each plant in a field. A farmer is not needed to be aware of and prevent every plant disease. Every year and every season, the climate varies. Keeping up with it might be difficult since the symptoms and patterns vary. Urgently needed is a model for plant disease identification that can work on a smartphone, a low-powered computer, or a low-powered drone and perform seamlessly within their constraints. In the past, various promising investigations on this topic have been conducted. However, there are currently significant limitations in this study. Real-world scenario-captured and annotated images are excluded from the collection. As a consequence, the training occurs in a controlled environment. Two, deep learning strategies emphasize the majority of the research (such as CNN). Another fascinating study focuses on image processing, image segmentation, and soft computing image processing methodologies. Most CNN techniques are not cost-effective to compute and have complicated structures. It costs a great deal each day to keep it operating. In addition, other image processing methods lack a high degree of precision. In this paper, we conducted several experiments to demonstrate the practical limitations of current approaches and suggest a lightweight model to overcome these issues. Based on this notion, we offer a lightweight convolutional neural network model. A revolutionary design that can accurately diagnose plant diseases and is lightweight may help save a substantial amount of computer resources. Consequently, farmers may use their smartphones or drones to monitor conditions earlier, more efficiently, and for less money.

II. Literature Review & Related Work

Using deep learning, several complicated issues have been addressed. However, this does not imply that deep learning answers all problems. Deep learning is one of the most successful pattern recognition techniques, and it has transformed computer vision and artificial intelligence. Several studies have identified plant diseases using machine learning with respectable accuracy. We choose plant diseases based on their year-round availability. Significantly, they may acquire several advantageous characteristics from a single leaf, such as form, texture, pattern, and color.All of these traits might be extracted by a variety of methods. In conventional machine learning, these characteristics often do not provide adequate accuracy. For example, it cannot appropriately categorize illnesses.

S. Khirade et al. [7] addressed the subject of digital plant disease detection. Image processing methods and neural network backpropagation (BPNN). The authors have expanded on several leaf image-based approaches for identifying plant disease. First, to segment the contaminated portion of the leaf, they have applied Otsu's thresholding, [8] followed by border detection and spot detection algorithms. Then, for plant disease categorization, they retrieved parameters such as color, texture, morphology, edges, etc. Finally, BPNN is used to categorize or detect plant disease.

Sammy et al. [9] provided a CNN for identifying illness categories; the author used this classification system in this investigation. There are nine distinct tomatoes, grape, corn, apple, and sugarcane leaf diseases. In this investigation, training on the system was undertaken for around 50 epochs, and the batch size was 22. With categorical cross-entropy, an Adam optimizer is used in this model. The achieved accuracy is 96.5 percent.

The authors of [10] suggested a classification method using a support vector machine (SVM) to identify healthy rice seedlings affected with the banana disease. This approach is successful, but it cannot be used for other forms of plant diseases. This is due to its exclusive emphasis on a single condition. In this study, the pictures are preprocessed using a variety of image processing methods to attain precision.

Another research [11] identifies 44 plant species using the CNN approach. They used a dataset from Kew, England's Royal Botanic Gardens. They were extracting characteristics from the segmented picture using CNN. The retrieved characteristics were then categorized using a Multilayer Perceptron (MLP). Their neural network consists of five hidden layers. In their report, there is no timeline detailing how long training takes. They use a variety of activation functions. The whole procedure is computationally intensive. Using their abundant data resources and intensive computing, they reached a 97.13 percent accuracy rate.

[12] Ramcharan and Amanda presented CNN approaches for a mobile-based platform. Where it can identify numerous leaf diseases. The notion is remarkable, but again it is computationally intensive. Their data and images were captured using a professional camera. The lack of larger megapixels in the camera of the typical smartphone reduces its accuracy.

Modern research intends to offer deep learning as a classification method for plant diseases, concentrating on leaf pictures and resource-intensive models. However, what use is it if the model consumes a great deal of resources and forecasts the incorrect classification? This work makes two significant additions to the categorization of plant diseases: a strong, lightweight model and the identification of the location of symptoms on an infected leaf. Image preprocessing, feature extraction, and classification are the three hierarchies emphasized in this design. To examine cutting-edge works, we place a great deal of emphasis on architecture and picture preprocessing.

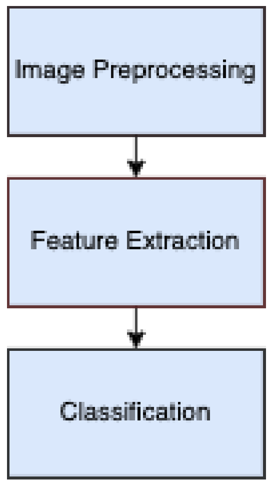


Fig. 1. Hierarchy Chart

We did not prioritize background removal during picture preprocessing and feature extraction. Numerous works attempt to eliminate the background, concentrating their investigation on the leaf. We used the picture as-is and moved on. On the basis of photos, we implemented transformations and normalized the RGB values, which assisted with feature extraction. Unfortunately, eliminating the background of a photo is difficult and often requires human intervention, which reduces the system's level of automation.

III. Methodology

To construct a robust model for plant leaf disease detection, we need a superior and novel technique. Our technique differs significantly from all other classic ML and DL approaches. To meet these goals, we have used

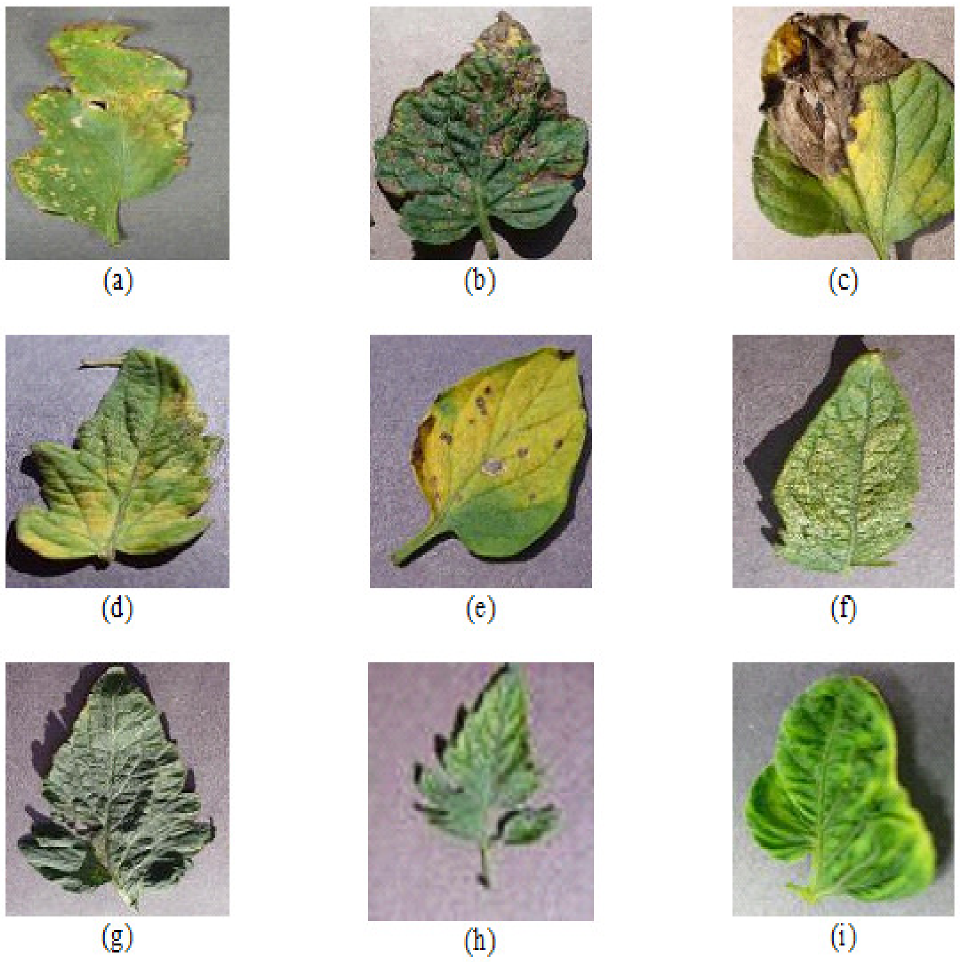


Fig. 2. Shown above are the samples of all the nine different diseases leaf. (Healthy leaf NOT included) (a) Bacterial Spot (b) Early Blight (c) Late Blight (d) Leaf Mold (e) Septoria leaf spot (f) Two-Spotted Spider Mite (g) Target Spot (h) Mosaic Virus (i) Yellow Leaf Curl Virus

a variety of ways. To identify plant diseases, an extensive collection of photographs of plant leaves is necessary. For this study, we had to narrow down our group's multiple plant illnesses to one unique plant leaf disease with ten distinct circumstances. Here, the tomato leaves are harvested. The images from the PlantVillage dataset were downloaded. This section describes the methodology used and [Fig. 3] provides a visualization of the images and a summary of the dataset and Proposed CNN Architecture

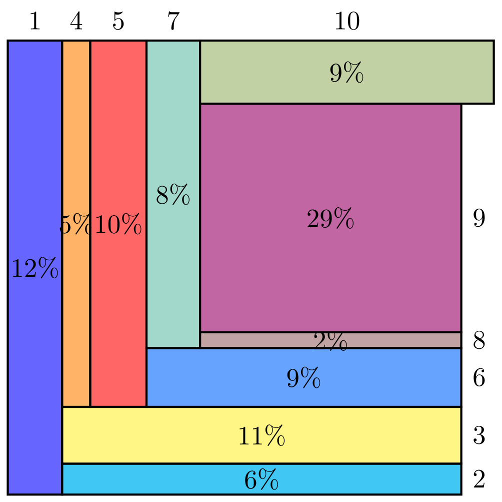
A. Dataset Description

A significant decline in classification ability is one of the significant drawbacks of the present research on plant disease detection. Another significant decline is data. Data is crucial in deep learning, CNN classification, or AI-related fields. On the this page, you will find a listing of photographs of all tomato leaves with their correct categorization. If you look carefully, you can see the disorders in the picture [Fig. 2]. These illnesses have distinct patterns. Below is a quick summary of the datasets and the number of images in each class. We have a total of 18160 images of tomato plant leaves.

TABLE I
Overview of our Dataset.

Diseases Name	Total Image
1. Bacterial Spot	2,127
2. Early Blight	1,000
3. Late Blight	1,909
4. Leaf Mold	952
5. Septoria Leaf Spot	1,771
6. Two-Spotted Spider Mite	1,676
7. Target Spot	1,404
8. Mosaic Virus	373
9. Yellow Leaf Curl Virus	5,357
10. Healthy Leaf	1,591

Here is a visualization percentage tile of the data where it represents the average percentage of the frequency of the data. Using this chart, we can determine the rate of pictures utilized by the CNN model for each categorization.



From this chart, as you can see, the lowest percentage of images is 2% (373) images. No. 8, respectfully, is classified as Mosaic Virus. Due to very low image collection, the training and classification accuracy might be affected, which turns into very low accuracy.

B. Image-Preprocessing & Data Augmentation

In this model, data sampling and data augmentation are also essential components. Data augmentation is one method for eliminating data diversification. Using data augmentation methods assists in cropping, padding, horizontally flipping and modifying the RGB value. Sometimes the data originates from the same environment rather than from many settings. Therefore, we must apply the method of data augmentation. After dividing the dataset into the train, test, and validation categories, we have just 15256 pictures to train and validate our model. We used 80% of the dataset for training purposes and 20% for testing. The validation set comprises 20 percent of the 80 percent training data set. Our training dataset consists of 11,624 photos, while our test and validation datasets include 3,632 and 2,908 images, respectively. To process this picture on our GPU, we use the Mini-batch technique. Our batch size was four. This implies that it will concurrently process four photos during training, testing, and validation. Mini-batch training dataset has 2,906 pictures, the test dataset contains 908 images, and the validation dataset contains 727 images. During picture pre-processing, we additionally randomized the horizontal flip of the data and also added 180-degree rotation. We include three mean and three standard deviation values for each RGB color channel to generate a new output for training and validation, which is also known as Normalizing the data set. We can represent the normalization function following showing below. NOTE: It's not a piecewise function.

$$Normalize(x) = \begin{cases} mean = & [0.485, 0.456, 0.406] \\ std = & [0.229, 0.224, 0.225] \end{cases}$$

After applying data augmentation, a photo [Fig 2.] set of training images is shown above as input to the proposed CNN model. The purpose of data augmentation in this data collection is to lessen the likelihood of overfitting. Another reason we used data augmentation is that we lack a sufficient dataset.

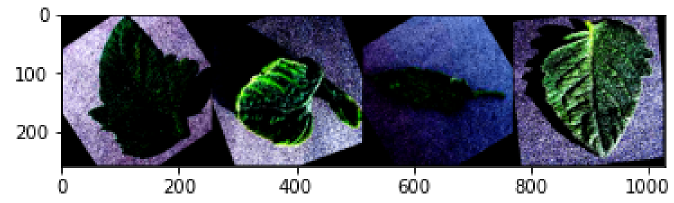


Fig. 3. Ground Truth: Leaf Mold. , Target Spot. , Leaf Mold. , Healthy.

As you can see above in [Fig 2] after applying data augmentation, you can see the patterns of the leaf. In the above Figs, as you can see, It has changed the image pixel value to the new value. so during the training of the model, the convolution functions can pick the pattern better.

Below table two displays the dataset split. this split has been used in this model.

TABLE II
DATASET SPLIT

Diseases Name	Total Image	Percentage
Training	11624	80%
Validation	2908	(20% form training data)
Test	3632	20%

C. Proposed Lightweight CNN Architecture

The purpose and the main goal of the lightweight model we proposed because there is not that much research based on lightweight models that detect plant diseases. So far, all the models we have seen have their complexity that takes a long time to process, and server-side computation is heavy-faced. In this model, we proposed three different convolutional layers. In the first layer of convolution, we take three input channels which are R. G. B values. We generate nine different feature Maps out of those three inputs. To generate all those nine different feature maps, our filter size is 5 X 5. For our feed forwarding network, it passes through a relu activation function. Our second layer of convolution takes 9 feature maps as input and generates 81 new feature maps. The filter size of this convolution is 5. X 5. In the second convolution layer, we implement stride. Our stride size is 3. When our stride is size 3, we move the filters to 3 pixels. We also have applied padding on this convolution layer. Basically, it's the number of pixels added to an image when it is being processed by the kernel of a CNN. Our padding size is 9 in this layer. For our third and last convolutional layer. It takes 81 feature maps from the second transformation and generates 6561 new feature maps. For the third convolutional layer, we still have the same filter size 5 X 5, and also for our feed forwarding network, it also passes through another relu activation function. In the third convolution layer, we also added stride and padding; stride size is 3, padding is 3. After each convolutional layer, there is a Max pooling happening which is also reducing our image size by 2 for itself. There's a total of 3 max pools that happened in this architecture. Using Max pulling significantly reduces the chance of over-fitting and computation power also. In this model, only in the second and third convolutional layer padding and stride are applied. In the end, our network is fully connected using fully connected linear layers. for our linear layers, we get 6561 input features coming in and from that, we classified 10. We also applied the softmax classifier layer here. After ten linear out features, our vector dimension one passes through a softmax classifier. Basically, this log softmax function normalized an input value into a vector of values that follows a probability distribution whose total sums up to 1. In our architecture, the dropout method applies randomly during the training process. Our Dropout percentage is 0.25. When Dropout is applied 25% neuron will be dropped during the training process which is also applied on a hidden layer. Our proposed model has 13,377,196 parameters that were trained. After training proposed model was 51.06 MB.

TABLE III
PROPOSED CNN ARCHITECTURE

Layer	Types	Kernel Size	Stride	Padding
C1	Convolution	5 x 5	0 x 0	0 x 0
S1	Max Pooling	3 x 3	0 x 0	0 x 0
C2	Convolution	5 x 5	3 x 3	9 x 9
S2	Max Pooling	3 x 3	3 x 3	0 x 0
C3	Convolution	5 x 5	3 x 3	3 x 3
S3	Max Pooling	3 x 3	3 x 3	0 x 0

Table III shows all the necessary kernel size, stride, and padding, and Table IV is displaying the proposed model parameter summary.

TABLE IV
MODEL PARAMETER SUMMARY

Layer (type:depth-idx)	Output Shape	Param
Conv2d: 1-1	[-1, 9, 252, 252]	684
MaxPool2d: 1-2	[-1, 9, 84, 84]	—
Conv2d: 1-3	[-1, 81, 33, 33]	18,306
MaxPool2d: 1-4	[-1, 81, 11, 11]	—
Conv2d: 1-5	[-1, 6561, 5, 5]	13,292,586
MaxPool2d: 1-6	[-1, 6561, 1, 1]	—
Linear: 1-7	[-1, 10]	65,620

TOTAL PARAMS: 13,377,196 | TRAINABLE PARAMS: 13,377,196 FORWARD/BACKWARD PASS SIZE (MB): 6.28 | PARAMS SIZE (MB): 51.03

During all three convolution layers. The generated output pass through the activation function. In this case, our activation function was relu. Relu's idea, the more positive the neuron more activated it is. The way we could represent relu using this piecewise equation function under below.

$$f(x) = max(0, X)$$

$$f(x) = \begin{cases} X & X \geq 0 \\ 0 & X < 0 \end{cases}$$

relu [13] function is nonlinear, which means we can use early backpropagate the error and have a multi-layer of neurons activated by the relu function.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Hardware, Software Environment

Despite the fact that the majority of deep learning models need high-performance computing and considerable time, there are a few exceptions. Our concept divides the old, conventional computing method from a more compact one. The workstation's specifications are summarized in Table 3.

TABLE V
WORKSTATION SPECIFICATION

Machine Types	Desktop Workstation
CPU	Intel(R) Xeon(R) CPU @ 2.00GHz
RAM	16 GB
GPU	Tesla P100-PCIE-16GB
OS	GNU/Linux

For the deep learning framework, we used PyTorch. [14] The PyTorch is an open-source machine learning library. Since the study was done in python programming language. We use the open-source jupyter notebook [15] system which was implemented in Google colab [16]. We also used aws [17] python jupyter notebook cloud bass system.

B. Proposed Model Parameters

The used models parameters are:

- Base Optimizer: Rectified Adam [18]
- Optimizer: Lookahead Optimizer [19], [20]
    1) K = 5
    2) Alpha = 0.5
- Learning rate: 1e-3
- Scheduler: Reduce learning rate [21]
    1) Factor=0.2
    2) Patience=2
    3) Cooldown=2
- Loss function: Cross Entropy Loss
- Momentum: 0.0009
- Batch size: 4
- Num workers: 0
- Number of epochs: 13

V. Results and Analysis

To evaluate the performance of our methodology, we used the accuracy and loss of the model. In the first fold, we achieved Training accuracy of 98.21% validation accuracy of 97.21%, and Test Accuracy of 97.08% with a loss of 0.120998. This loss was performed in the 13Th epoch. Our model has 13.3 million parameters, which is very low compared to other studies. The proposed train model Params size was (MB): 51.03, which can be reduced to half using archive technology to achieve end-user delivery also. Our model is not just lightweight; It also outperforms the two best Image processing CNN architectures. The loss curves vs. epoch of our designed model are displayed in Figure 4, and Figure 5 shows the Learning rate change during training.

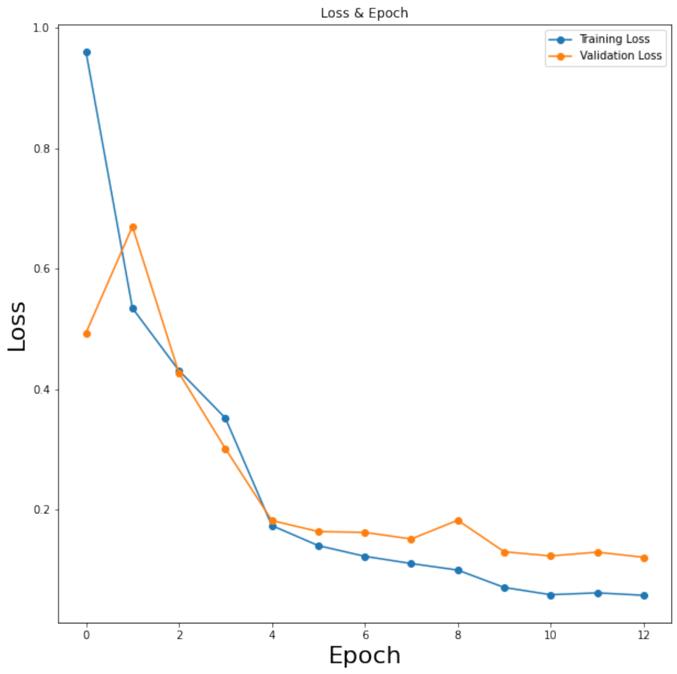


Fig. 4. Loss VS Epoch, Decreasing

In [Fig 4.], our model starts underfitting at the 6Th epoch. Then it started going up at the 8Th epoch, and loss started increasing. Due to our intelligent learning rate scheduler, our optimizer realized that loss was growing, not decreasing. So it shifts to a lower learning rate, and as a result, its start loss drops. At the 10Th and 11Th epochs, it's still underfitting. The learning rate scheduler recognizes the pattern and stops the training process. If your compare the learning rate graph[Fig 4] and the loss and epoch graph [Fig 5] side by side, you will see a linear trend. In [Fig 5.], as you can see the learning rate has been decreasing every other epoch. This model converges in 13 epochs and achieves an accuracy of 97.08% (testing dataset), outperforming classical algorithms by a huge margin by time and accuracy.

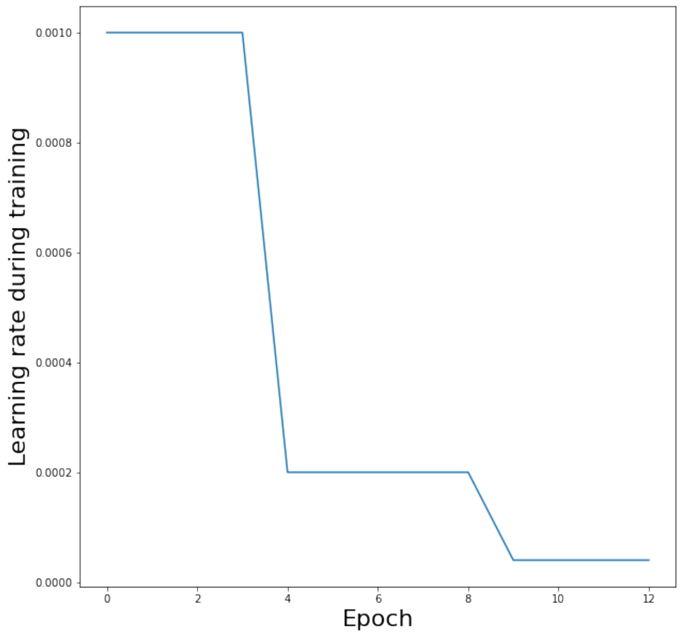


Fig. 5. Learning rate VS Epoch; During training, Decreasing.

TABLE VI
Table of Confusion Matrix
0. Bacterial Spot, 1. Early Blight , 2. Late Blight
3. Leaf Mold, 4. Septoria Leaf Spot, 5.Two-Spot Spider Mite
6. Target Spot, 7. Mosaic Virus, 8. Yellow Leaf Curl Virus
9. Healthy Leaf

	0	1	2	3	4	5	6	7	8	9
0	421	3	0	4	0	0	0	1	3	0
1	7	181	0	10	0	1	2	2	0	0
2	0	0	328	0	0	1	1	3	0	0
3	1	9	1	366	1	4	1	0	1	0
4	0	1	0	3	189	3	3	1	0	0
5	0	4	0	3	3	337	0	0	0	1
6	0	0	0	0	0	0	323	5	3	0
7	3	2	0	1	1	2	8	276	0	0
8	5	1	0	2	1	0	0	1	1031	0
9	0	0	0	0	0	0	1	0	0	66

As you could see the learning curve has been converging when the loss goes down. keep in mind that we use mini-batch techniques so that way we load images parallelly simultaneously. In our confusion Matrix, as you can see, testing set classification. To be clear, the number of classifications numbers are different because we took 20% of the data from the main dataset. We did not take an even distribution of data for our testing set; there are some classification category has a higher number of images than the training set. Using this technique introduces non-linearity into the dataset

In our model, we propose a lightweight CNN. We have achieved various types of accuracy based on the Train and test data size. In TABLE VII below as you can see, the three different sizes of training Accuracy. In our analysis,

TABLE VIII
CLASSIFICATION ACCURACY OF VARIOUS MODEL

Classes	Classification Accuracy(%) Proposed Model	Classification Accuracy(%) VGG-16	Classification Accuracy(%) AlexNet
Bacterial Spot	97.47	98.79	96.83
Early Blight	89.16	82.20	79.70
Late Blight	95.31	93.60	95.70
Leaf Mold	94.50	86.96	92.86
Septoria Leaf Spot	97.58	95.51	93.28
Two-Spotted Spider Mite	94.16	94.74	96.41
Target Spot	99.63	90.91	92.06
Mosaic Virus	92.65	92.65	95.83
Yellow Leaf Curl Virus	98.90	98.92	99.43
Healthy Leaf	98.49	99.33	99.34

TABLE VII
COMPARISON OF OUR PROPOSED METHOD WITH OTHER METHODS

Architecture name	Training Time (%)	Accuracy(%)
VGG-16 (Pretrained=False)	00:52:27	93.12
AlexNet (Pretrained=False)	00:27:54	94.34
Proposed Model	**00:13:94**	**97.08**

we concluded that this model would not increase its training accuracy with more data. It's at its optimal point, which is 98.21%. Therefore, it may increase between 0.5 up-to 1.5%, not a significant amount. It may need to use different augmentation processes and optimizers described in the future work section[VI]. In Table[VIII] above it shows how our model did overall; compare with out other models (PreTrained=False) for classification cases. Overall Testing accuracy

VI. Proposed Future Work

For our proposed future work, we can try to get an even distribution set of training data and test the proposed model. We can also try different activation functions such as Tanh and Leaky ReLU. For optimizers, we can try to apply SGD or adam.

VII. Conclusion

Convolutional neural networks (CNNs) have done remarkable feats in various disciplines, including a growing interest in agriculture. However, deep learning is not a miracle cure, despite its dominance in several challenging tasks such as picture classification and object recognition. Experimenting with other algorithms and enhancing the approach's characteristics, which are most likely the approach's limiting factor, may be used to improve the classical method's precision further. Therefore, it is essential to comprehend CNN's key concepts and advantages and the limitations of deep learning to employ CNN in plant disease detection research and improve plant disease detection performance.

References

[1] D. Nelson. (2015) Save the plankton, breathe freely. [Online]. Available: www.nationalgeographic.org/activity/save-the-plankton-breathe-freely/

[2] D. C. S. Baird. (2013) How do trees give earth all its oxygen? [Online]. Available: https://wtamu.edu/ cbaird/sq/2013/01/05/how-do-trees-give-earth-all-its-oxygen/

[3] (2018) Ag and food sectors and the economy. [Online]. Available: www.ers.usda.gov/data-products/ag-and-food-statistics-charting-anr.the-essentials/ag-and-food-sectors-and-the-economy/

[4] (2018) The world factbook. [Online]. Available: www.cia.gov/library/publications/the-world-factbook/fields/214.html.

[5] A. P. Ken Pernezny, Monica Elliott and N. Havranek, "Guidelines for identification and management of plant disease problems: Part ii. diagnosing plant diseases caused by fungi, bacteria and viruses," *IFAS extension*, 2008. [Online]. Available: https://edis.ifas.ufl.edu/mg442

[6] J. Isleib. (2015) Signs and symptoms of plant disease: Is it fungal, viral or bacterial? [Online]. Available: https://www.canr.msu.edu

[7] S. D. Khirade and A. Patil, "Plant disease detection using image processing," in *2015 International Conference on Computing Communication Control and Automation*, 2015, pp. 768–771.

[8] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[9] S. V. Militante, B. D. Gerardo, and N. V. Dionisio, "Plant leaf detection and disease recognition using deep learning," in *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, 2019, pp. 579–582.

[10] C.-L. Chung, K.-J. Huang, S.-Y. Chen, M.-H. Lai, Y.-C. Chen, and Y.-F. Kuo, "Detecting bakanae disease in rice seedlings by machine vision," *Computers and Electronics in Agriculture*, vol. 121, pp. 404–411, 02 2016.

[11] S. H. Lee, C. S. Chan, P. Wilkin, and P. Remagnino, "Deep-plant: Plant identification with convolutional neural networks," 09 2015, pp. 452–456.

[12] A. Ramcharan, P. McCloskey, K. Baranowski, N. Mbilinyi, L. Mrisho, M. Ndalahwa, J. Legg, and D. Hughes, "A mobile-based deep learning model for cassava disease diagnosis," *Frontiersin*, 03 2018.

[13] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2018. [Online]. Available: https://arxiv.org/abs/1803.08375

[14] From research to production. [Online]. Available: https://pytorch.org/

[15] Project jupyter exists. [Online]. Available: https://jupyter.org/

[16] Welcome to colaboratory. [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynbrecent=true

[17] Jupyter notebook server. [Online]. Available: https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter.html

[18] A. Yaguchi, T. Suzuki, W. Asano, S. Nitta, Y. Sakata, and A. Tanizawa, "Adam induces implicit weight sparsity in rectifier neural networks," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 318–325.

[19] Lookahead optimizer: k steps forward, 1 step back. [Online]. Available: https://paperswithcode.com/paper/lookahead-optimizer-k-steps-forward-1-step

[20] G. H. J. B. Michael R. Zhang, James Lucas, "Lookahead optimizer: k steps forward, 1 step back," *Creative Commons Attribution Non Commercial Share Alike 4.0 Internationa*, 2019.

[21] Reducelronplateau. [Online]. Available: https://pytorch.org/docs/stable/generated /torch.optim.lr$_{s}$*cheduler.ReduceLROnPlateau.html*