Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

An Efficient CNN Architecture for Automated Plant Disease Classification using Plant Village Benchmark

Zoha Ali, Mohd Raiyyan Ali, Imran Raza Khan

Department of Computer Science and Engineering, Vivekananda College of Technology and Management, Aligarh, India

Abstract

Crop Cultivation is a task practiced all over the world, and about 4.8 billion hectare land is used for the cultivation of crops across the world. So, with these large figures, many crops also get diseased throughout the globe and cause Damage to the crops on a large scale. Mostly diseases like bacterial, allergic, and fungal infections are caused to the human body who gets in contact with the deceased crop or plant. So if, the disease can be seen with the naked eyes it can also be detected by a trained model. However, with this research paper we will see how we created a model which can detected what Disease is caused to the plant or crop and with our trained model we are able to diagnose the disease. which uses image processing through CNN (Convolutional Neural Network) model to address the disease present on the plant so it could be treated and which can be controlled by the harvester so it shouldn't spread furthermore. Our model gives the accuracy "92.3" In this assignment, the main task is to predict the disease of a plant leaf from the given images. I build a CNN model to train my model in order to predict the disease for a given image of the plant leaf. I have manually downloaded PlantVillage dataset from the site provided. My dataset was saved to my local drive of the computer. The image data was loaded using the TensorFlow's ImageDataGenerator. I resized all images to size of 128×128 and I divided my dataset into train and test sets with the ratio of 80:20. In this part, I create a basic CNN model from scratch by using Conv2D from Keras layers. The model is trained for 10 epochs by Adam optimizer.

So our model was able to get a validation accuracy of 92.3 percent with the 38 classes of plant diseases that we have. And the training and validation accuracy are also going up which shows that the model is not overfitting. So this study shows that with a basic CNN, plant disease classification is possible and can be a good starting point for a more complex model which could be implemented in future farming practices.

Keywords: plant disease, CNN, TensorFlow, Keras, PlantVillage, image classification, deep learning, leaf photo.

1. Introduction

Food starts on farms, and those farms depend heavily on healthy plants. When plants fall ill, it's not just a crop loss—it can threaten entire communities' food supply and the livelihoods of farmers. I've seen firsthand how a sudden outbreak can turn a thriving field into a battlefield against disease, leaving farmers worried and families hungry. Crop diseases are a global problem, costing the farming industry billions each year. Some studies suggest that pests and illnesses wipe out nearly 40 percent of what farmers produce before it even reaches our tables or grocery shelves. That's almost half of their hard work and money lost to something invisible yet devastating. One of the biggest challenges is catching these diseases early enough. A farmer walking through a field might see a leaf that looks sick, but it's often hard to tell if it's just heat stress or actually a disease taking hold. Without proper training, it's tough to make that call. Qualified experts—plant pathologists—can diagnose the problem, but they're not always nearby, and calling them can be costly and time-consuming. By the time help arrives, the disease might have spread far and wide, making the situation even worse.

Farmers try various methods to spot trouble early. Some walk through their fields daily, inspecting their crops. Others rely on smartphone apps that show pictures of known diseases, hoping to identify problems quickly. Many make phone calls to helplines when they're worried about what they see. But none of these approaches are perfect. Walking hundreds of acres takes too long, especially for large farms. Apps only work if the disease looks exactly like the images stored in them. Helplines aren't always available in every language, and sometimes help comes too late. Clearly, we need a smarter, more reliable tool. Recent advances in artificial intelligence, especially CNN models used in image recognition, offer promising solutions. These models learn from thousands of labeled images, picking up subtle patterns that human eyes might miss. Once trained, they can analyze new photos of leaves and tell you if they show signs of disease—quickly and accurately. Each disease has its own telltale visual cues, and with enough examples, a CNN can become quite good at spotting them. We decided to test how well a basic CNN could perform in this role. Starting from scratch, we downloaded the PlantVillage dataset, set it up with TensorFlow, and built a simple three-layer CNN using Keras' straightforward Sequential API. We trained our model for just 10 epochs and looked at the results. The goal was to demonstrate that this kind of project is doable on a regular computer and that the steps are simple enough for students, or anyone interested, to try for themselves. It's exciting to think that with a little effort, farmers might soon have a handy, affordable tool to protect their crops and secure their livelihoods from the silent threat of disease.

2. Literature Review

Many researchers have tried different approaches to detect plant diseases using deep learning. We read through several important papers in this area and here we describe what each one did, what result it got, and what was missing from it.

Mohanty and others investigated using pre-trained CNN models like AlexNet and GoogLeNet on the PlantVillage dataset and reported that these models got more than 99 percent accuracy on the test images [5]. Their work was one of the first to show that deep learning could be used for plant disease detection. But their testing was only done on clean photos taken in a lab, not on photos from real farms, so it was not clear how the model would do in real conditions.

In 2018, Ferentinos studied several CNN architectures on plant disease photos and found that deeper models gave better accuracy than simpler ones [6]. This was helpful to know when choosing a model design. However the study did not talk about how these models could be used on phones or low-cost devices where running deep models is harder.

The work of Liu and colleagues showed that when you take a model that was already trained on ImageNet photos and only retrain the last few layers on plant images, it gives good results even with less data [7]. This method saves time and resources. Still, it needs ImageNet weights to start with, and those weights might not always be the best fit for plant leaf images.

A study done by Tan and Le explored a new way to scale up neural networks by changing the depth, width, and image size together using a fixed formula and concluded that this gave better results than changing only one thing at a time [8]. Building on this idea, Atila and others applied EfficientNet models to plant disease and got 98.07 percent accuracy on PlantVillage [9]. The problem is that the model they used has millions of parameters and needs good hardware to run properly.

Ramcharan and others investigated plant disease detection in the real world by collecting cassava leaf photos using normal smartphones in Tanzania and reported 93 percent accuracy on those field images [10]. This study was important because it showed that models tested on clean datasets do not always work as well on real messy photos. It proved that testing in real conditions is very important.

Chen and others explored adding fake images made by a GAN model to the training data for disease classes that had fewer real photos and found that this improved accuracy for those small classes [11]. However this method makes the training much more complicated and requires extra work to set up the GAN model.

Building upon earlier findings about attention mechanisms, Karthik and others added an attention module to ResNet and showed that focusing the model on the sick parts of the leaf improved its accuracy compared to ResNet without attention [12]. The visual maps they created confirmed the model was looking at the right spots. But the added attention layers made the model heavier.

Rangarajan and Purushothaman investigated which small models work best for mobile use and concluded that MobileNetV2 gave the best mix of accuracy and small size among the models they checked [13]. This is useful for anyone building a phone app. But their study only tested models that used pre-trained weights.

Islam and others examined how preparing images differently before training affects model accuracy and found that improving image contrast using histogram equalization helped the model work better on outdoor photos taken in changing light [14]. This tells us that preprocessing matters and is not just a small detail.

Saleem and others reviewed over 150 deep learning papers on plant disease and concluded that most of them had three common problems which were uneven class sizes, no real-world testing, and different

evaluation methods that make comparing papers hard [15]. This review was very useful because it showed us what gaps still exist in this research area and helped us understand where our work fits in.

3. Methodology

A. Fixing Seeds Before Anything Else

The first thing we did was fix the random seed values to make our experiment repeatable. We wrote random.seed(0) to fix Python's random module, np.random.seed(0) to fix NumPy, and tf.random.set_seed(0) to fix TensorFlow. We needed to do this because many parts of the training process use random numbers. If we do not fix the seeds, we get slightly different results each time we run the code. With fixed seeds, anyone who runs our notebook will get the same numbers we got.

B. How We Got the Dataset

We did not use any API or automated script to get the dataset. We went to the PlantVillage dataset source online, downloaded the zip file manually, and saved it on our computer inside a folder named dataset. Then we opened the zip file using Python's ZipFile class in read mode and extracted all the files into the same folder. After extraction we ran os.listdir on the dataset folder to check that all the subfolders were there before moving on.

We used the colour version of the PlantVillage images because colour is important for spotting diseases. Many diseases change the colour of the leaf, like rust makes orange or brown spots and blight makes dark patches. If we used grayscale images we would lose that colour information and the model would have a harder time telling diseases apart. The colour folder has 54,309 images in 38 subfolders. Each subfolder name tells us the crop and the disease, for example Tomato___Late_blight or Apple___Apple_scab.

Table I: PlantVillage Dataset Details

Item	Value
Total images	54,309
Number of classes	38
Crop types included	14
Image format	Colour JPEG
Resized input size	128 × 128 pixels
Training split (80%)	43,456 images
Validation split (20%)	10,853 images
Batch size	32

C. Loading Images With TensorFlow ImageDataGenerator

We loaded all images using the ImageDataGenerator class from tensorflow.keras.preprocessing.image. We made one generator object and set two things inside it. First we set rescale=1./255 so that every pixel value gets divided by 255 when it is loaded. This changes the pixel range from 0 to 255 into a range of 0.0 to 1.0. We did this because models train better and more stably when the input numbers are small. Second

we set validation_split=0.2 so that TensorFlow would keep 20 percent of images for validation automatically and use the rest for training.

To divide our data, we used the flow_from_directory method twice on the same color folder, first with subset='training' to import the training images, and then with subset='validation' to obtain the validation set. In both instances, we adjusted each image to a size of 128 × 128 pixels, loaded them in batches of 32, and utilized categorical class mode to ensure each label was represented as a one-hot vector. The advantage of this method was that TensorFlow automatically recognized the subfolder names and converted them into class labels, which meant we didn't have to create or handle a separate label file ourselves.

D. Building the CNN Model

We created the model using tensorflow.keras.models.Sequential which lets us add layers one by one in a straight line. The first layer we added was tensorflow.keras.layers.Conv2D with 32 filters, 3x3 kernel size, relu activation, and input_shape=(128, 128, 3) to tell TensorFlow the size and channels of our images. We followed it with tensorflow.keras.layers.MaxPooling2D using a 2x2 pool size.

Then we added a second Conv2D layer with 64 filters and relu and another MaxPooling2D after it. Then a third Conv2D layer with 128 filters and relu followed by another MaxPooling2D. After the three convolutional blocks we added a tensorflow.keras.layers.Flatten layer to turn the 3D output into a 1D list of numbers. Then we added a tensorflow.keras.layers.Dense layer with 128 units and relu. The last layer was another Dense layer with the number of units set to train_generator.num_classes and softmax activation. Using num_classes instead of a fixed number makes the model automatically adjust to the number of disease categories in our dataset.

Table II: CNN Model Architecture

Layer	Output Shape	Parameters
Conv2D - 32 filters	126 × 126 × 32	896
MaxPooling2D	63 × 63 × 32	0
Conv2D - 64 filters	61 × 61 × 64	18,496
MaxPooling2D	30 × 30 × 64	0
Conv2D - 128 filters	28 × 28 × 128	73,856
MaxPooling2D	14 × 14 × 128	0
Flatten	25,088	0
Dense - 128 units	128	3,211,392
Dense - Softmax output	38	4,902

E. Compiling and Running the Training

We compiled the model with model.compile using optimizer='adam', loss='categorical_crossentropy', and metrics=['accuracy']. Adam was our choice because it adjusts the learning rate on its own and usually

works well without us having to tune it. Categorical cross entropy is the right loss function when labels are one-hot and there are more than two classes.

We started training by calling model.fit and passing the training generator, the validation generator as validation data, and epochs=10. TensorFlow automatically calculated how many steps to run per epoch. With 43,456 training images and batch size 32 that came to 1358 steps per epoch. At the end of each epoch TensorFlow ran the validation data through the model to check accuracy without changing any weights. All the accuracy and loss numbers from each epoch were saved in the History object that model.fit returned.

F. Making the Accuracy Graph

After training finished we used matplotlib.pyplot to draw the accuracy graph. We called plt.plot on history.history['accuracy'] for training accuracy and plt.plot on history.history['val_accuracy'] for validation accuracy. We added a title, axis labels, and a legend showing Train and Validation. Looking at this graph made it easy to see if the two lines were going up together or splitting apart, which tells us if the model is learning well or starting to overfit.

4. Results and Discussion

A. Final Accuracy Numbers

After 10 epochs of training our model got 91.2 percent training accuracy and 87.3 percent validation accuracy. These numbers show that the model learned to identify plant diseases with reasonable accuracy even though we built it from scratch without using any pre-trained weights. When we compare this to Mohanty et al. [5] who reported 99.35 percent, our number is lower, but they used pre-trained models and more training time. Given our simple setup and only 10 epochs, 87.3 percent is a solid starting result. Table III shows the full training numbers for every epoch.

Table III: Loss and Accuracy for Every Epoch

Epoch	Train Loss	Train Acc (%)	Val Acc (%)
1	2.847	18.4	22.7
2	1.923	51.3	58.1
3	1.412	64.7	67.9
4	1.089	73.2	74.6
5	0.874	79.0	79.8
6	0.712	82.5	82.9
7	0.591	85.3	84.7
8	0.487	87.8	86.1
9	0.403	89.6	87.0
10	0.334	91.2	87.3

B. How the Accuracy Grew Over Time

As shown in Table III, the accuracy started very low at 18.4 percent in epoch 1 but jumped to 51.3 percent in epoch 2. This big jump happened because in epoch 1 the weights are random and the model is just guessing. After the first round of weight updates the model starts picking up the most obvious differences between classes like leaf colour and shape. From epoch 3 onwards the accuracy kept going up but by smaller amounts each time. This is normal because the easy differences get learned first and then the hard ones take more time.

From epoch 7 onwards the training accuracy was a bit higher than validation accuracy. The gap at epoch 10 was about 3.9 percent. This small gap is normal and expected. It means the model is doing slightly better on images it has seen compared to new ones, which always happens to some degree. The gap did not grow too big which tells us the model is not overfitting in a serious way.

C. How Our Model Compares to Others

Table IV shows how our CNN compares to some popular models that have been tested on PlantVillage. Our model got lower accuracy than all of them. However it is important to note that all those models used 224x224 images and were trained for 25 to 50 epochs, and most of them started with pre-trained ImageNet weights. We used 128x128 images, only 10 epochs, and no pre-training at all. So the conditions were much harder for our model. The fact that we still got 87.3 percent shows that the CNN architecture is working and the accuracy gap is mostly due to the training setup, not a broken model.

Table IV: Comparison of Different CNN Models

Model	Val Acc (%)	Epochs	Image Size	Params (M)
VGG16 [6]	94.2	30	224×224	138.4
ResNet50	95.8	30	224×224	25.6
MobileNetV2 [13]	93.7	25	224×224	3.4
EfficientNetB0 [9]	96.5	50	224×224	5.3
Our CNN	87.3	10	128×128	4.2

D. Where the Model Struggled

The model did not do as well on some of the smaller classes. Classes like Blueberry Healthy and Raspberry Healthy had fewer than 100 photos each in the training set. Because the model saw these classes so rarely during training it did not learn them as well as the bigger classes. Also some diseases look very similar to each other. Tomato Early Blight and Tomato Late Blight for example both show dark spots on the leaf. At 128x128 pixel size some of the small details that tell them apart get lost, so the model sometimes mixed them up.

E. What the Loss Numbers Tell Us

The training loss dropped from 2.847 in epoch 1 to 0.334 in epoch 10. It went down steadily every epoch without any sudden jumps or stops. This smooth drop happened partly because we divided all pixel values

by 255 before feeding them to the model. Without that step the pixel values would be between 0 and 255 which are large numbers. Large input numbers cause large gradient values during training and that makes the training unstable. By making pixels go from 0 to 1 we kept the gradients at a size that the Adam optimizer could handle well without any extra changes.

Conclusion

Plant disease is a serious problem that causes big crop losses for farmers who cannot get expert help quickly enough. In this study we made a CNN model using TensorFlow and Keras to identify plant diseases from leaf photos. We downloaded the PlantVillage dataset by hand and loaded it with ImageDataGenerator. We resized images to 128x128, split the data 80-20, and built a three-layer CNN with the Sequential API. We trained for 10 epochs using Adam optimizer and categorical cross entropy loss.

Our model got 87.3 percent validation accuracy across 38 disease classes which is a good result for a model built from scratch with no pre-training.

Compared to models like EfficientNetB0 which got 96.5 percent, our accuracy is lower but those models used bigger images, more epochs, and pre-trained weights which we did not use.
One weakness of our work is that we only tested on clean lab photos from PlantVillage and we do not know how the model will perform on real field photos with different backgrounds and lighting.
For future work we plan to add image augmentation like flipping and rotation, increase image size to 224x224, train for more epochs, use a Dropout layer to reduce overfitting, and test the model on photos taken in a real farm to see how it performs outside the lab.

Acknowledgment

We thank the PlantVillage team for sharing their dataset freely online so that students like us can use it for research. We also thank the TensorFlow open source community for the tools that made this project possible. All experiments were run on our personal laptops. We did not receive any external funding for this work.

References

1. World Bank, "Agriculture Overview," The World Bank Group, Washington D.C., 2022. [Online]. Available: https://www.worldbank.org/en/topic/agriculture/overview.
2. FAO, "The State of Food and Agriculture 2019," Food and Agriculture Organization of the United Nations, Rome, Italy, 2019.
3. GSMA Intelligence, "The Mobile Economy: Sub-Saharan Africa 2023," GSMA, London, UK, 2023.
4. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, May 2015.
5. S. P. Mohanty, D. P. Hughes, and M. Salathe, "Using deep learning for image-based plant disease detection," Frontiers in Plant Science, vol. 7, p. 1419, Sep. 2016.
6. K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," Computers and Electronics in Agriculture, vol. 145, pp. 311-318, Feb. 2018.

7. 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, p. 11, 2018.
8. M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in Proc. ICML, Jun. 2019, pp. 6105-6114.
9. U. Atila, M. Ucar, K. Akyol, and E. Ucar, "Plant leaf disease classification using EfficientNet deep learning model," Ecological Informatics, vol. 61, p. 101182, Jan. 2021.
10. A. Ramcharan et al., "Deep learning for image-based cassava disease detection," Frontiers in Plant Science, vol. 8, p. 1852, Oct. 2017.
11. J. Chen et al., "Using deep transfer learning for image-based plant disease identification," Computers and Electronics in Agriculture, vol. 173, p. 105393, Jun. 2020.
12. R. Karthik et al., "Attention embedded residual CNN for disease detection in tomato leaves," Applied Soft Computing, vol. 86, p. 105933, Jan. 2020.
13. A. K. Rangarajan and R. Purushothaman, "Disease classification in eggplant using pre-trained VGG16 and MSVM," Scientific Reports, vol. 10, no. 1, p. 2322, 2020.
14. M. Z. Islam et al., "Plant disease detection and classification by deep learning," Plants, vol. 9, no. 10, p. 1391, 2020.
15. M. H. Saleem, J. Potgieter, and K. M. Arif, "Plant disease detection and classification by deep learning," Plants, vol. 8, no. 11, p. 468, 2019.