An Optimized Dense Convolutional Neural Network Model for Disease Recognition and Classification in Corn Leaf

Abdul Waheed[1], Muskan Goyal[1], Deepak Gupta[1], Ashish Khanna[1], Aboul Ella Hassanien[2], Hari Mohan Pandey[3]

[1]Maharaja Agrasen Institute of Technology, New Delhi, India.
[1]e.abdul@protonmail.com, [1]goyalmuskan1508@gmail.com, [1]deepakgupta@mait.ac.in, [1]ashishkhanna@mait.ac.in

[2]Cairo University, Cairo, Egypt
[2]aboitcairo@gmail.com

[3]Edge Hill University, Lancashire, England
[3]Pandeyh@edgehill.ac.uk

Abstract: An optimized dense convolutional neural network (CNN) architecture (DenseNet) for corn leaf disease recognition and classification is proposed in this paper. Corn is one of the most cultivated grain throughout the world. Corn crops are highly susceptible to certain leaf diseases such as corn common rust, corn gray leaf spot, and northern corn leaf blight are very common. Symptoms of these leaf diseases are not differentiable in their nascent stages. Hence, the current research presents a solution through deep learning so that crop health can be monitored and, it will lead to an increase in the quantity as well as the quality of crop production. The proposed optimized DenseNet model has achieved an accuracy of 98.06%. Besides, it uses significantly lesser parameters as compared to the various existing CNN such as EfficientNet, VGG19Net, NASNet, and Xception Net. The performance of the optimized DenseNet model has been contrasted with the current CNN architectures by considering two (time and accuracy) quality measures. This study indicates that the performance of the optimized DenseNet model is close to that of the established CNN architectures with far fewer parameters and computation time.

Keywords: Artificial intelligence, Deep Learning, Corn leaf diseases recognition, Convolutional Neural Network.

1. INTRODUCTION

The agriculture sector is adopting artificial intelligence and machine learning in diverse areas including disease detection, soil monitoring, weed controlling, diagnosing pests, computer vision and drones for crop analysis, and weather predictions. Agriculture is the most important sector of the Indian economy, accounting for 18% of its GDP [1]. The main source

of income for a large population of India is agriculture. Hence, India is extremely dependent on crop productivity.

Corn is a very common crop in India. Corn farming is essential in India because it has a high export potential and a vast population of farmers is dependent on it [2, 3]. Corn is being used in various sectors like cattle feed, poultry industry and food and beverage industries. It has become a primary food in many parts of the world, with its overall production exceeding that of wheat and rice. But in India, the yield of corn is almost half of the overall global average [2]. One of the reasons for low corn yield is that it is subjected to many diseases which significantly reduce the overall crop yield. Certain leaf diseases remain a challenge for crop production, not only through scaling down the crop yield but also through the reduction in its nutritional value. Some of the prevalent leaf diseases include common rust, gray leaf spot, and leaf blight.

These leaf diseases may look very comparable in their nascent phases, hence making it very difficult to detect through the naked eye. Detection of leaf diseases through visual observation requires a team of specialists and constant crop monitoring [4]. Thus, making it very costly, time-consuming and less reliable. We can leverage deep learning techniques to conduct automatic, rapid and more precise leaf disease detection and classification. Several researchers have worked in this region to create a model that can predict the existence of leaf disease in corn plants. We examine some of these researches in the following paragraphs.

Zhang et al. [5] proposed an improved SVM called genetic algorithm support vector machine (GA-SVM). The author collected and classified six types of corn leaf diseases. The following steps were carried out to classify the diseases: For image processing, the JPEG images were transformed into BMP format. Furthermore, the images were converted from RGB template to HSI, to extract various features (average and standard deviations of R, G, B and shape features such as area, circumference, circularity, height, and width, etc). Then segmentation was implemented to get binary images. Finally, the technique of Orthogonal rotation was used to obtain appropriate parameters for the genetic algorithm. Twenty feature parameters were fed to the model. For classification comparisons between SVM and GA-SVM, four kernels were selected: linear, polynomial, RBF and sigmoid. The author concluded after comparison that GA-SVM has a higher classification rate (between 69.63% - 90.09% for SVM and between 88.72% -92.59% for GA-SVM).

Alehegn [6] classified three types of corn leaf diseases (common rust, leaf blight, leaf spot) using KNN (K-nearest neighbour) and ANN (artificial neural network) classification algorithms. Images of healthy and corn leaf diseases were taken from Ethiopia farming areas.

For training and testing, a minimum of 800 images were considered. Texture, morphology and color features were obtained from the images. Total 22 features were fed to the classification model of KNN and ANN. Lastly, the author concluded that ANN had a higher performance rate with an accuracy of 94.4% whereas KNN reached an accuracy of 82.5%.

An improved GoogLeNet and Cifar 10 model was proposed in [7] for classification of 8 corn leaf diseases (southern leaf blight, brown spot, rust, round spot, dwarf mosiac, curvularia leaf spot, gray leaf spot, northern leaf blight). Total 500 images were collected for 9 classes (8 classes of diseased corn leaves and one for healthy leaves). Data augmentation technique was deployed on the images. The proposed GoogLeNet architecture had 22 layers and lesser parameters than VGG and Alexnet model. Cifar 10 model was also optimized by adding more layers and using ReLU function. The performance of the models was evaluated on the corn leaf dataset. The precision of GoogLeNet and Cifar 10 was 98.9% and 98.8%, respectively.

Durga and Anuradha [8] used SVM and ANN algorithm for leaf disease classification in tomato and corn plants. Image dataset included 200 pictures with a collection of healthy leaves and diseased leaves like northern leaf blight, common rust, bacterial spot, tomato mosaic virus, etc. They used the following steps to identify the diseases: the RGB picture was converted to the grayscale picture and the image was then segmented by calculating the intensity gradient at each pixel. For feature extraction, HOG (histogram of oriented gradients) procedure was used. The extracted features were fed to the SVM and ANN classifier models. For corn crops, SVM gave an accuracy of 70-75% and ANN gave an accuracy of 55-65%.

Bhatt and Sarangai [9] developed a system for the identification of corn leaf diseases using CNN architectures (VGG16, inception-v2, ResNet50, mobileNet-v1) and applied a combination of adaptive boosting with decision tree-based classifier to distinguish between diseases that appeared to be similar. Four categories of image data included healthy leaves, common rust, leaf blight, and leaf spot. Pictures of each class were taken from PlantVillage dataset. The pictures were resized for image preprocessing according to the necessity of the CNN model used. Features derived from the CNN models were fed to the classifiers (softmax, support vector machine and random forest). It was observed that inception-v2 gave the highest precision with random forest. From the confusion matrix of each classifier, the author noted that leaf blight and leaf spot classes were difficult to differentiate. Therefore, to increase the classification accuracy between them, the Adaptive boosting technique was applied to the best performing model (based on Inception-v2 and RF). Finally, the model reached an accuracy of ~98%.

Lu C and Gao S, et al. [37] used a fuzzy least-square vector machine (FLSVM) to classify corn leaf diseases. To segment the diseased areas, YCbCr color space technology was used. Then, the spatial grey-level co-occurrence matrix was used to extract the texture features of the region and FLSVM was used to classify the diseases. A total of 12 pictures of four different diseases were taken. Further, 0 degrees, 45 degrees, 90 degrees, and 135 degrees matrix are computed for each disease. Five texture features extracted were used in training for the fuzzy least square algorithm. The results were approximately 98% correct, for identification and classification.

These studies have shown good results but on small image dataset with an increase in parameters or computation time, which has an adverse effect on recognition performance. Also, some methods defined above involve machine learning techniques for feature extraction and classification of corn leaf diseases with the help of SVM, ANN, random forest and so on. For image processing in machine learning, the researcher needs to extract the features manually in an image and feed them to the classification model. These workflows are very complicated, challenging and time-consuming. Therefore, the objective of this research is to improve the conventional methods of identification and to present an optimized model for corn leaf disease identification with lesser parameters and computation time.

Deep learning image classifiers are used for corn leaf disease recognition and classification. In deep learning, Convolutional neural networks (CNNs) is arguably the most popular architecture. It was specially designed for working with images. Convolutional neural networks are significantly used in image recognition, video recognition, object detection, medical image analysis, traffic control [38] and flow prediction [39], anomaly detection [40][41] and recommendation systems for healthcare [42]. Deep convolution neural networks have accomplished unprecedented performance in the field of computer vision over recent years. For image recognition, models should have a lot of prior knowledge to make up for all the data that we don't have. Convolutional neural networks (CNNs) are one such class [10,11,12,13], but established CNN architectures give a high-performance rate with many parameters or high convergence time. By altering their depth and breadth, their capacity can be regulated, and mostly they make the right assumptions about the nature of images. CNN's, therefore, have much fewer links and parameters and are simpler to train as compared to normal neural networks [10].

We have been studying that the most efficient way of enhancing the performance of neural networks is by expanding their depth and width, but a recent study shows that once the network reaches a saturation point, scaling up the depth and width might cause an increase in error rate

[33]. Moreover, scaling up the depth and width has two other main challenges. The first challenge that occurs with bigger networks is many parameters that makes the network more likely to overfit [14]. Another challenge is that increasing the size of the network increases the use of computational resources. Since the computational budget is usually finite, we need a systematic distribution of computing resources [14].

While considering the above discussion, we have proposed a novel optimized DenseNet architecture for disease recognition and classification in corn leaf. The main contribution to this paper can be stated as follows.

- We propose an optimized DenseNet architecture for corn leaf disease recognition and classification. Further, we have trained four other established CNN models such as VGGNet, XceptionNet, EfficientNet, NASNet for corn leaf disease recognition and classification.
- The performance of the proposed model has been analyzed through rigorous simulations. It is noted that the proposed CNN architecture takes fewer parameters and is computationally cost effective. Simulation results reveal that the proposed network has 77,612 parameters and shows 98.06% accuracy as compared to the existing CNN models such as EfficientNet, VGG19Net, NASNet, and XceptionNet. Further, these results indicate that the accuracy of the proposed model is close to existing CNN architectures, but with significantly fewer parameters.

The remaining exposition is as follows. Background discussion is outlined in Section 2. Proposed model is elaborated in Section 3. Simulation model is discussed in Section 4. The conclusion and future avenues are presented in Section 5.

2. Background

CNNs are one of the most important parts of deep learning and are extensively used in computer vision. CNN's recognize visual patterns with minimal preprocessing straight from pixel images. The recent rise in popularity of CNN is attributed to its immense effectiveness. LeNet [15] architecture started the history of CNN. The interest in CNN began with AlexNet [10] and ZFNet [16] and it has grown exponentially ever since. Some popular architectures that have been considered in this study are discussed below.

a) VGGNet: Simonyan and Zisserman [17] created VGGNet. It is a deep architecture which extracts features at low spatial resolution. VGGNet has two versions, namely VGG16 (with 16 weight layers) and VGG19 (with 19 weight layers). Large receptive fields in VGGNet have been substituted with consecutive layers of 3x3 convolutions with ReLU in between. The

convolutional stride was fixed to 1 pixel. The padding of convolution layer input was maintained as 1 pixel and max-pooling was done with a stride of 2 over a 2 × 2-pixel window [17]. The pile of convolutional layers was joined with 3 fully connected layers and one softmax layer. VGGNet has 144 million parameters, of which approximately 124 million are used in the last 3 Fully Connected layers. Therefore, if the fully connected layers could be eliminated, the architecture efficiency could be enhanced. This step was taken in subsequent architectures replacing the first Fully Connected layer with a node layer using a method called average pooling [18]. The architecture of VGGNet (VGG16) is shown in Figure 1[36].

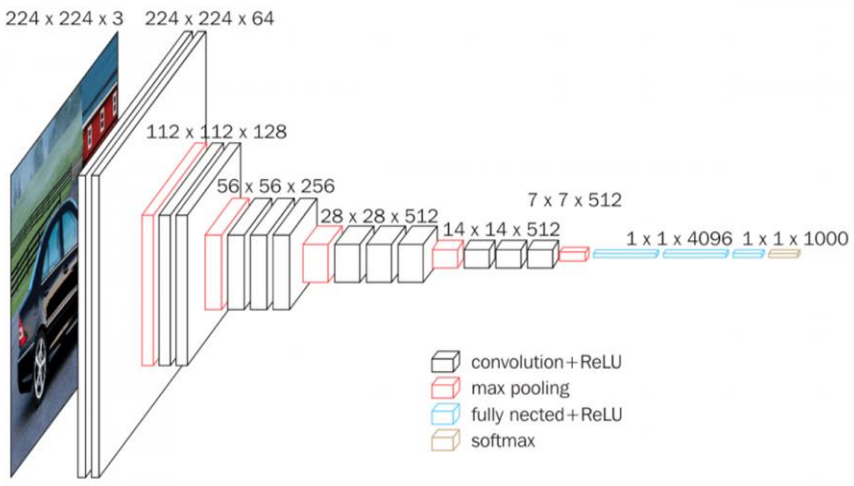


Figure 1: VGG Architecture

b) XceptionNet: Szegedy et al [14] introduced a new term referred to as an inception network. The main hallmark of the inception network was to utilize computing resources inside the network. It eliminates all the fully connected layers and uses average pooling, therefore, the system has only 5 million parameters [18]. Inception module is the building block of inception network, which captures parallel paths with distinct receptive field dimensions and operations in the stack of feature maps [14]. After the tremendous success of the inception network, GoogLeNet (Inception V1) was modified to InceptionV2, Inception V3, and Inception-ResNet. Inspired by inception network, Chollet [19] proposed a novel deep CNN architecture called XceptionNet. It has the same number of parameters (~ 23 million) as used in Inception V3, but it showed a higher performance rate in fact it significantly outperformed the Inception V3 on largest dataset. This model replaced the conventional inception modules with depth-wise separable convolutions (spatial convolution carried out separately over each input channel)

preceded by a point-specific convolution (1×1 convolution) [19]. The XceptionNet architecture depicted in Figure 2 [19].

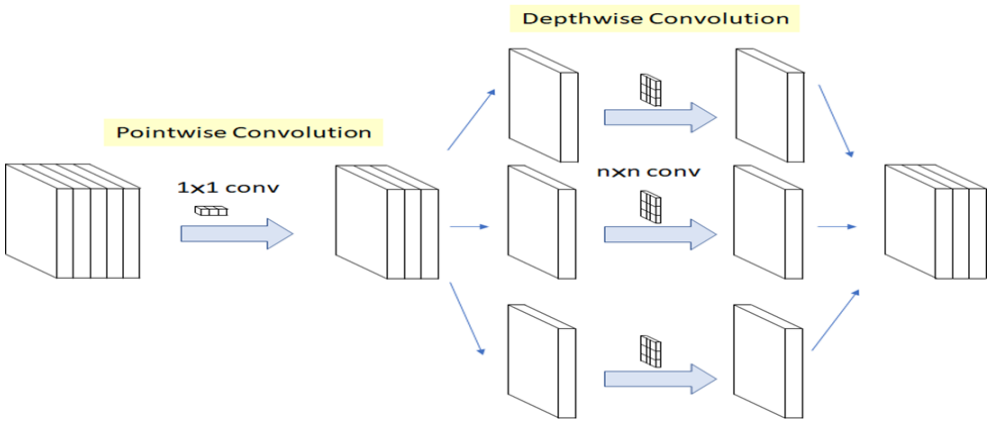


Figure 2: The Modified Depthwise Separable Convolution used in XceptionNet

c) NASNet: The need for computing resources grows as the dataset increases. Zoph and V. Le [22] proposed a method to search for an architectural building block on a small dataset and transferred the block to a larger dataset. NASNET achieved an 82.7 percent top-1 and 96.2 percent top-5 state-of-the-art accuracy on ImageNet. Two types of convolutional layers (or cells) are required to build NASNet: (1) convolutional layers that return same size feature maps (Normal Cells), and (2) convolutional layers which return a feature map with its height and width decreased by a factor of two (Reduction Cells). The overall architecture is predefined in [22] as shown in Figure 3.

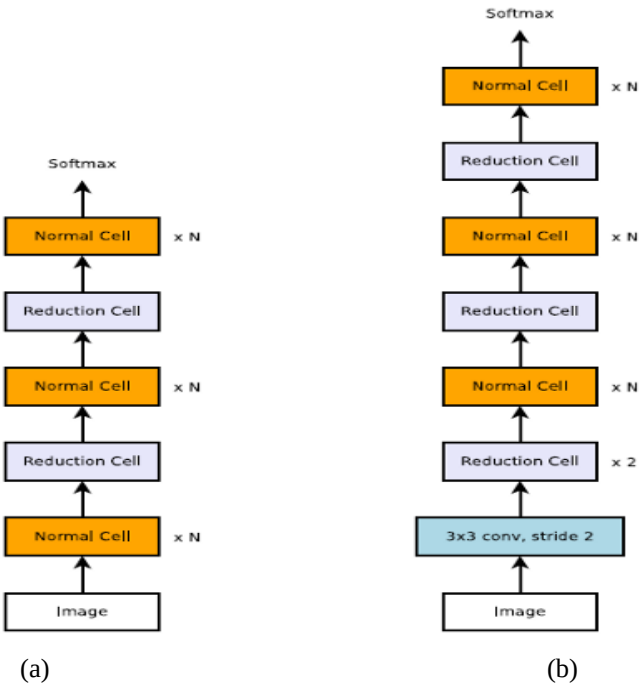


Figure 3: Scalable Architecture (a) CIFAR10 Architecture (b) ImageNet Architecture

d) EfficientNet: Tan and Le [20] proposed EfficientNet, which showed high precision value and, found more flexible as compared to ConvNets [21]. Authors [20] had addressed the challenges (developed at a fixed resource budget, and then scaled up to gain better accuracy) of the ConvNets by suggesting new scaling method which showed the tendency to uniformly scales all dimensions (depth/width/resolution) of the CNN. The effectiveness of the proposed scaling method was demonstrated by implementing it for scaling on MobileNets and ResNet. EfficientNet-B7 revealed impressive results as it achieved state-of-the-art result of 84.4% top-1/97-1% top-5 accuracy on ImageNet with 8.4x smaller and 6.1x quicker compared to finest current ConNets [21]. Figure 4 presents scaling model for EfficientNet where Figure 4 (a) and Figure 4 (b) – (d) respectively presents a baseline network and a traditional scaling technique whilst Figure 4 (e) depicts the compound scaling introduced in [20].

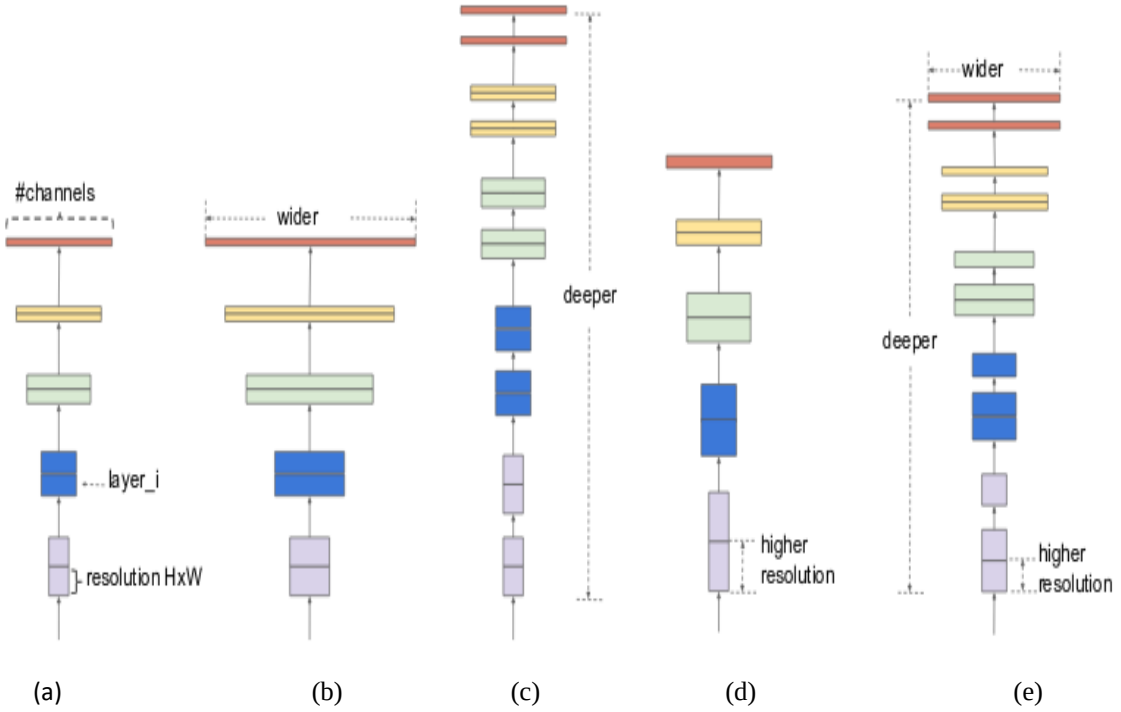


Figure 4: Model Scaling (a) baseline network; (b) and (d) is a traditional scaling technique that increases only one dimension of network width, depth, or resolution (e) compound scaling, was introduced in the [23].

e) DenseNet: DenseNet connects each layer with every other layer in a feed-forward manner. The feature-maps of all the previous layers are used as inputs for each layer, and their feature-maps are used in all subsequent layers as inputs. DenseNets solves the issue of vanishing gradient [34], strengthen feature propagation, encourages reuse of features, and significantly reduces the number of parameters [24]. Each *l*th layer has *l* inputs, composed of the feature-

maps of all preceding convolutional blocks. It passes its feature-maps to all $L-l$ subsequent layers. This introduced L(L+1)/2 connections in an L-layer network [24]. This architecture has a dense connectivity pattern, therefore called a Dense Convolutional neural network.

These established CNN architectures have shown high performance rate for leaf disease identification but with an increase in parameters and computation time. In this paper, we present an optimized custom DenseNet for leaf disease identification and classification which has fewer parameters and, hence, less computation time than the above discussed CNN architectures. An important point to note at this stage is – "recurrent neural network (RNN) or long short-term memory (LSTM) network) have been designed to work differently. The RNN is usually used to work with sequential data, in contrast, CNN is designed to exploit spatial correlation of data and works well on images. CNN was specially designed for working with images. The architecture of the proposed model is discussed in detail in the next section.

3. Proposed Model

This section presents the methodology adapted in the current research. We have presented a step-by-step description of the methodology to make the overall concept understandable.

3.1 Flow Diagram

Figure 5 is used to present the workflow diagram of the proposed model. Here, a deep convolutional neural network (DCNN) is used to construct the classification model. Initially, we have a '*Total Leaf Dataset*' that is divided into two sets, namely training set and testing set with a distribution of data items as 90% (contains 11097 images) and 10% (1235 images) respectively. Image preprocessing is performed on both training and testing sets images. An image is a matrix of pixels where each pixel is represented by three channels (RGB), thereby making it a three-dimensional matrix (mxnx3 where m is the number of pixels in a column and n is the number of pixels in a row). The pixel values in the images must be normalized using a preferred scaling technique just-in-time during the training or model evaluation process. We have implemented normalization by dividing each pixel with 255.0 to bring them in a range of 0 to 1. After normalization, the data augmentation technique is deployed on the images. Such multi-channeled images are then fed as an input to the CNN model (which has two parts namely feature extraction and classification). It can take hours, days or even weeks for deep learning models to train. If the run is suddenly halted, one can lose a lot of work. Therefore, the weights of the best fit model are stored in a file (weights.h5), so that they can be trained or used later. And finally, the performance of the model is assessed. The general approach model for corn leaf disease recognition and classification is shown below in Figure 5.

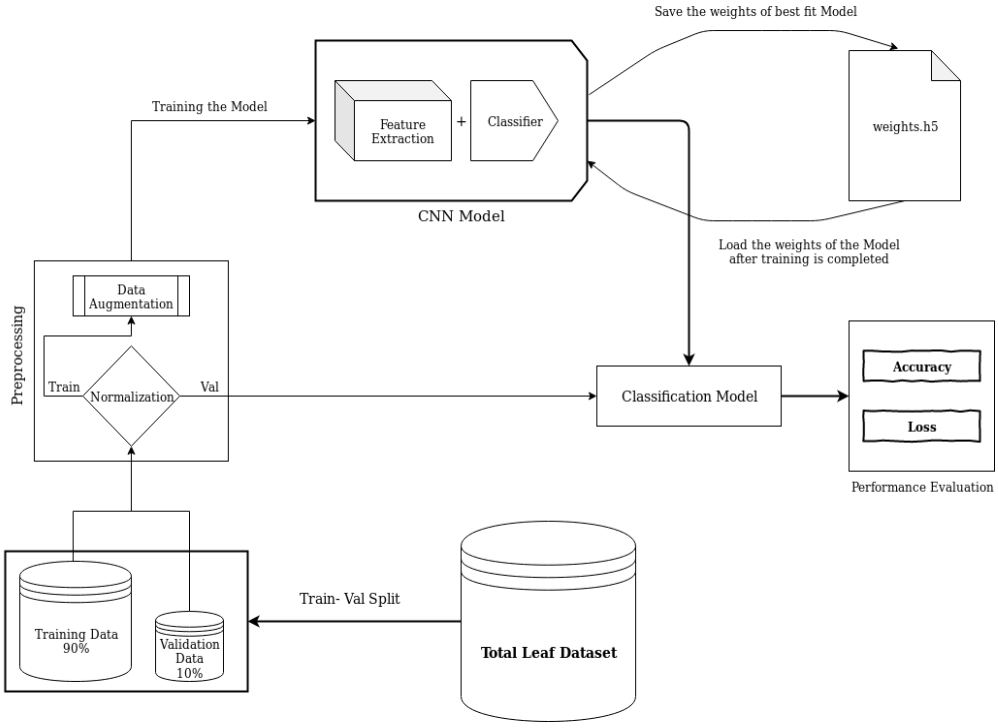


Figure 5: Flow diagram of the corn leaf disease classification model

3.2 Proposed Model

3.2.1 Why DenseNet ?

Problems arise with the CNNs as they get deeper because the route from the input layer to the output layer becomes so large that when it hits the other side of the network, the backpropagating gradient disappears. This vanishing gradient issue was resolved in ResNet [29] architecture by introducing skip connections between layers that add outputs from prior layers to stacked layers' outputs. ResNets shortened the stochastic depth [30] of the network by dropping some random layers during training which allowed better gradient and information flow. Thus, increasing the ability to train deeper networks. Unlike ResNets, DenseNets [24] links all layers to each other directly and ensures an optimal flow of information within the layers with a simple connectivity pattern. Rather than adding the residual, as in ResNet, all the feature maps are concatenated by the DenseNet. Therefore, they can be deeper than normal networks and can be readily optimized with this fresh residual usage. With very less parameters, this new CNN architecture has achieved state-of-the-art results on datasets (CIFAR, ImageNet). DenseNets have two main advantages: 1) Parameter Efficiency 2) Deep Supervision. This network has fewer parameters than standard networks because firstly it generates feature maps with a tiny growth rate (the total output feature maps of a layer is growth rate) and secondly it applies a 1×1 convolution as a bottleneck layer before each 3×3

convolution to reduce the number of input feature maps [24]. Lastly, extreme residual usage produces deep supervision as each layer in DenseNet receives more supervision from the loss function due to the shorter links.

3.2.2 Framework of the Optimized DenseNet

DenseNet is made by connecting multiple dense blocks. A dense block is a set of layers that are attached to all its former layers. One layer in the dense block is composed of 4 layers: 1) Batch Normalization 2) ReLU activation 3) 3x3 convolution 4) Dropout. The layer between two dense blocks i.e. the transition layer performs down-sampling. It is made of 1) Batch Normalization 2) ReLU activation 3) 1x1 convolution and 4) Average Pooling. Figure 6 shows the architecture of the proposed model.

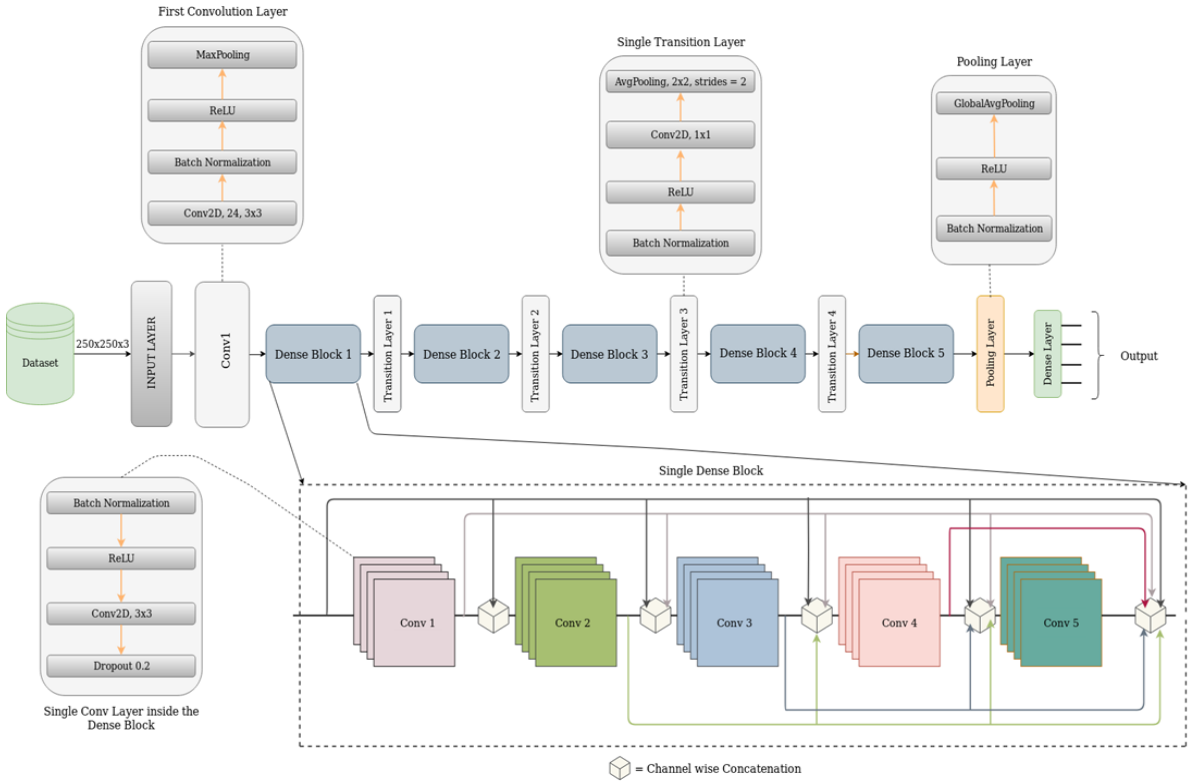


Figure 6: Proposed Architecture of the optimized DenseNet

An optimized DenseNet model is trained for leaf disease recognition and classification. The input given to the DenseNet is multi-channeled images (each of size 250x250x3) in a batch of 32. The input is then passed through the first convolution layer (Conv2D) that extracts features and outputs a feature map, which is then passed to the first dense block of 5 convolution layers. Each convolutional layer in a dense block is composed of a batch normalization layer, a ReLU [31] activation layer, a 3x3 Conv2D layer, and a dropout layer. The first convolution layer of

the dense block generates 4 feature maps (with growth rate = 4) that are further concatenated to the input. Then the second convolution layer is added to produce another 4 feature maps, which are again linked to the previous feature maps. This concatenation action is used in all the dense blocks. The convolution layers of a dense block must generate feature maps of the same dimension because feature maps with different sizes cannot be concatenated. The output of a single dense block is the concatenation of the output of 5 convolution layers and thus, it has 5*4 feature maps (20 feature maps per dense block). The layer between the two adjacent dense blocks i.e. the transition layer changes the size of the feature map by convolution and average pooling. We use a 1x1 convolution followed by a 2 x 2 average pooling in the transition layer. The proposed architecture has 5 dense blocks and 4 transition layers. After each dense block except for the last one, a transition layer is attached. The last dense block is followed by a global average pooling layer and softmax classifier. Table 1 shows the layer-wise description of the proposed model. Categorical cross entropy is used as model's loss function and Adaptive Moment Estimation (Adam) is used as the optimizer of the model. ReLU is used as an activation function because it is computationally efficient. Other functions like sigmoid and tanh are computationally expensive and suffer from vanishing gradient problem. The model was trained for 47 epochs with each epoch taking approximately 3.7 minutes, following which we get a validation accuracy of 98.06%.

Table 1: Layered description of the proposed model

S. No	Layer	Output	Parameters
1	Input layer	250x 250 x 3	0
2	Conv2D_1	125 x 125 x 24	920
3	DenseBlock_1	125 x 125 x 44	6400
4	TransitionBlock_1	62 x 62 x 28	1408
5	DenseBlock_2	62 x 62 x 48	7200
6	TransitionBlock_2	31 x 31 x 48	2496
7	DenseBlock_3	31 x 31 x 68	11200
8	TransitionBlock_3	15 x 15 x 68	4624
9	DenseBlock_4	15 x 15 x 88	15200
10	TransitionBlock_4	7 x 7 x 88	8096
11	DenseBlock_5	7 x 7 x 108	19200
12	GlobalAvgPooling	108	432
13	DenseLayer	4	436
	Total		77,612

3.3 Adam Optimizer

Adaptive Moment Estimation (Adam) [32] is a technique used for stochastic optimization. It evaluates adaptive learning rates for distinct parameters. This optimizer brings together the benefits of two latest optimizers, i.e. AdaGrad [35] and RMSProp to provide a technique for handling noisy issues with sparse gradients. It is easy to implement, works efficiently, has fewer memory requirements, invariant to the rescaling of gradient and works with sparse gradients. Adam uses the average of the second-order gradients rather than using the learning rates based on the average of the first moments. It specifically calculates the gradient and the squared gradient as an exponential moving average. Most of the features for the real time datasets are sparse due to which the corresponding gradient for most instances could be zero and therefore the parameters update is also zero hence we will not proceed the learning. In order to respond to this problem, the learning rate should be adaptive to relatively sparse data. The Adagrad, RMSProp, and Adam optimization algorithms are based on adaptive learning rates. In Adagrad the learning rate decays aggressively and RMSProp has high number of oscillations which might delay the convergence, so the best choice is Adam which overcomes both problems. Algorithm 1 presents the Adam optimizer used in this study.

Algorithm 1: Adam Optimizer

Input: current weights and bias

Output: updated weights and bias

1. Initialize $Vd^w = 0$, $Sd^w = 0$, $Vd^b = 0$, $Sd^b = 0$, $t = 0$
2. On iteration t:
3. Use current mini-batch to compute d^w, d^b
4. $t = t + 1$
5. $Vd^w = \beta1\ Vd^w + (1-\beta1)\ d^w$, $Vd^b = \beta1\ Vd^b + (1-\beta1)\ d^b$

   (momentum like update with hyper-parameter as β1)
6. $Sd^w = \beta2\ Sd^w + (1-\beta2)\ d^{w2}$, $Sd^b = \beta2\ Sd^b + (1-\beta2)\ d^b$

   (RMSProp like update with hyper parameter as β2)
7. Implement bias correction
8. $Vd^w_{corrected} = Vd^w/(1-\beta1^t)$, $Vd^b_{corrected} = Vd^b/(1-\beta1^t)$
9. $Sd^w_{corrected} = Sd^w/(1-\beta2^t)$, $Sd^b_{corrected} = Sd^b/(1-\beta2^t)$
10. Update parameters
11. $w_t = w_{t-1} - \alpha\ (Vd^w_{corrected} / \sqrt{Sd^w_{corrected}} + \varepsilon)$
12. $b_t = b_{t-1} - \alpha\ (Vd^b_{corrected} / \sqrt{Sd^b_{corrected}} + \varepsilon)$
13. Return w_t, b_t

Parameters required in Adam optimizer are:

- α : Step size(also called learning rate) parameter (0.001)

- d^w, d^b: weights and bias gradient
- β1 :The first moment exponential decay rate (0.9)
- β2 : The second moment exponential decay rate (0.999)
- ε : Fuzz factor; Very small number to avoid division from zero.
- t: Initial time step
- Vd^w : first moment vector for weights
- Vd^b: first moment vector for bias
- Sd^w: second moment vector for weights
- Sd^b : second moment vector for bias

4. Simulation model

4.1 Simulation Setup

Python language and Google Colaboratory is used to train and test the entire model on a laptop. Colaboratory is a free Jupyter notebook environment that does not require configuration and operates in the cloud completely [26]. It is pre-packaged with essential machine learning and artificial intelligence libraries, such as matplotlib, TensorFlow, and keras. The overall specifications of Colaboratory are shown below in Table 2 [27].

Table 2: Overall software and hardware specifications

NAME	**SPECIFICATIONS**
GPU	1xTesla K80, having 2496 CUDA cores, compute 3.7, 12GB (11.439GB Usable) GDDR5 VRAM
CPU	1xsingle core hyper threaded i.e. (1 core, 2 threads) Xeon Processors @2.3Ghz, 45MB Cache
RAM	~12.6 GB Available
DISK	~320 GB Available
LANGUAGE	Python
OS	MacOS version 10.14.5
Development Env.	Jupyter notebook environment on google colabs

Most of the computer vision applications use CNN. Two major difficulties with CNN are hardware for excellent performance and high-power consumption of this hardware [12]. Therefore, high-performance hardware like Colaboratory's GPU is required. Using the accelerated runtime of Colaboratory to train CNNs is 2.93 times faster on average than using all Linux server physical cores [25].

4.2 Hyperparameters

The proposed models' hyperparameters are shown in Table 3 that are used throughout the training of the model. The model's accuracy can be influenced by changing the initial learning rate. The model is optimized using Adam optimizer with an initial learning rate of 0.0005 which is further fine-tuned to 0.00001. Grid search is used to find the optimal hyperparameter values. The model is trained by grid search for all combinations by using different sets of hyperparameters. Kernel is initialized using glorot uniform and bias with zeros. The batch training method is to split the training and testing set into several batches with each batch comprising 32 images.

Table 3: Hyperparameters

Parameter	Value	Description
get_size	250x250x3	Size of the image we are feeding into the model
batch_size	32	Number of images in a batch
k	4	The growth rate of optimized DenseNet (see section 3.2.2)
min_delta	0.01	Required change in the monitored quantity to consider as improvement
patience	3	number of epochs without any improvement in the monitored quantity after which the training will be halted
alpha	0. 0005	Learning rate

4.3 Dataset

Total 12332 pictures of 250 x 250 pixel dimensions are manually gathered from different sources which are divided into 4 classes of crops: Common rust (3816 images), Healthy crop (3720 images), Cercospora leaf spot Gray leaf spot (1644 images), Northern leaf blight (3152 images). A picture of each class is shown below in Figure 7.

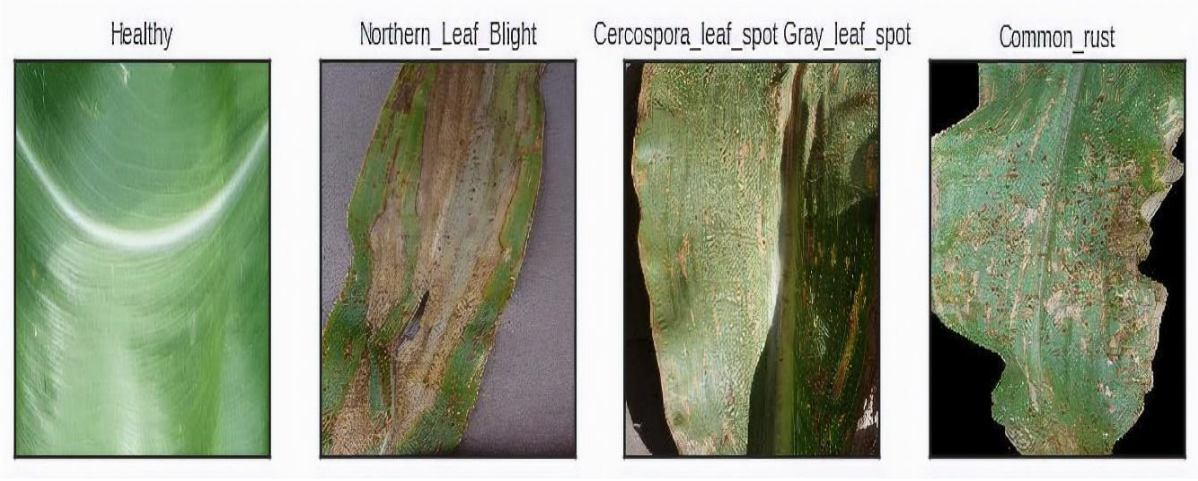


Figure 7: A Sample of each class of corn leaf diseases

The collection comprises of 12332 images out of which 11097 (90%) and 1235 (10%) images are divided as train and test set. After the dataset split, the data augmentation technique is applied to the images. The model used batch training with a batch size of 32, in which the images are processed in batches of the given size. Deep learning models are robust when trained on large dataset and training is also stable on large dataset. We have used data augmentation technique to increase the number of samples by creating artificial samples from existing samples which is discussed in the next section. The dataset for the corn leaf disease images is presented by the graph in Figure 8.

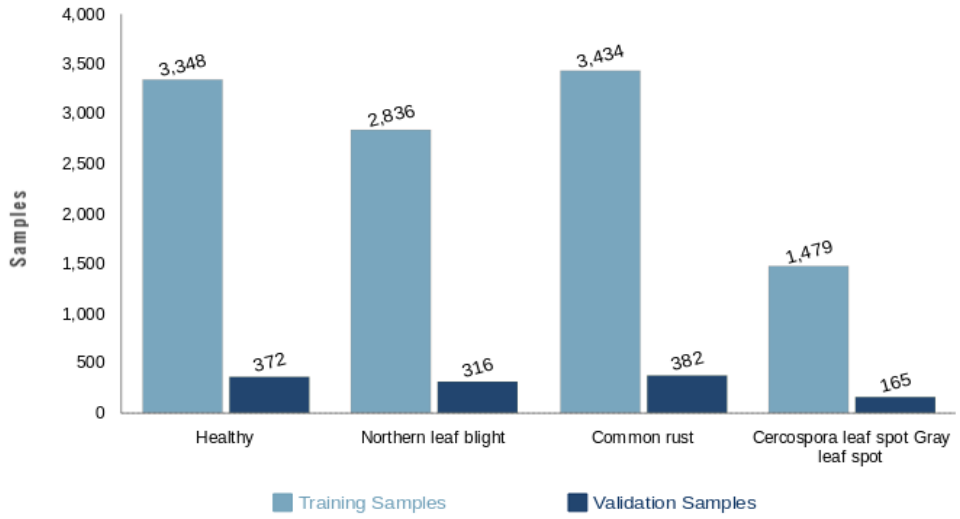


Figure 8: Dataset partitioning of corn leaf disease images

4.4 Data Augmentation

Deep learning models perform better when the size of the dataset is large hence to achieve this, we expanded the dataset by creating artificial samples from existing samples. The technique of creating artificial samples from existence samples by varying orientations of original samples is called data augmentation. Data augmentation allows programmers to considerably enhance the diversity and eventually the size of data available to train the models [28]. It is a known fact that CNN can handle variations in image and classify items even when they are positioned in distinct orientations [43] [44]. This is fundamentally the premise of data augmentation. To train CNN, substantial data is required so that it can learn and extract more features. The data collection process is associated with a cost that can be in terms of money, human effort, computational resources, and time consumption. Therefore, one may need to augment the existing data by making some minor alterations to images like scaling, rotating, translating, etc. The advantage of generating a greater number of training examples is that it

makes the model more robust and generalized; thereby improving its accuracy. All images in the dataset are subject to several steps of data augmentation.

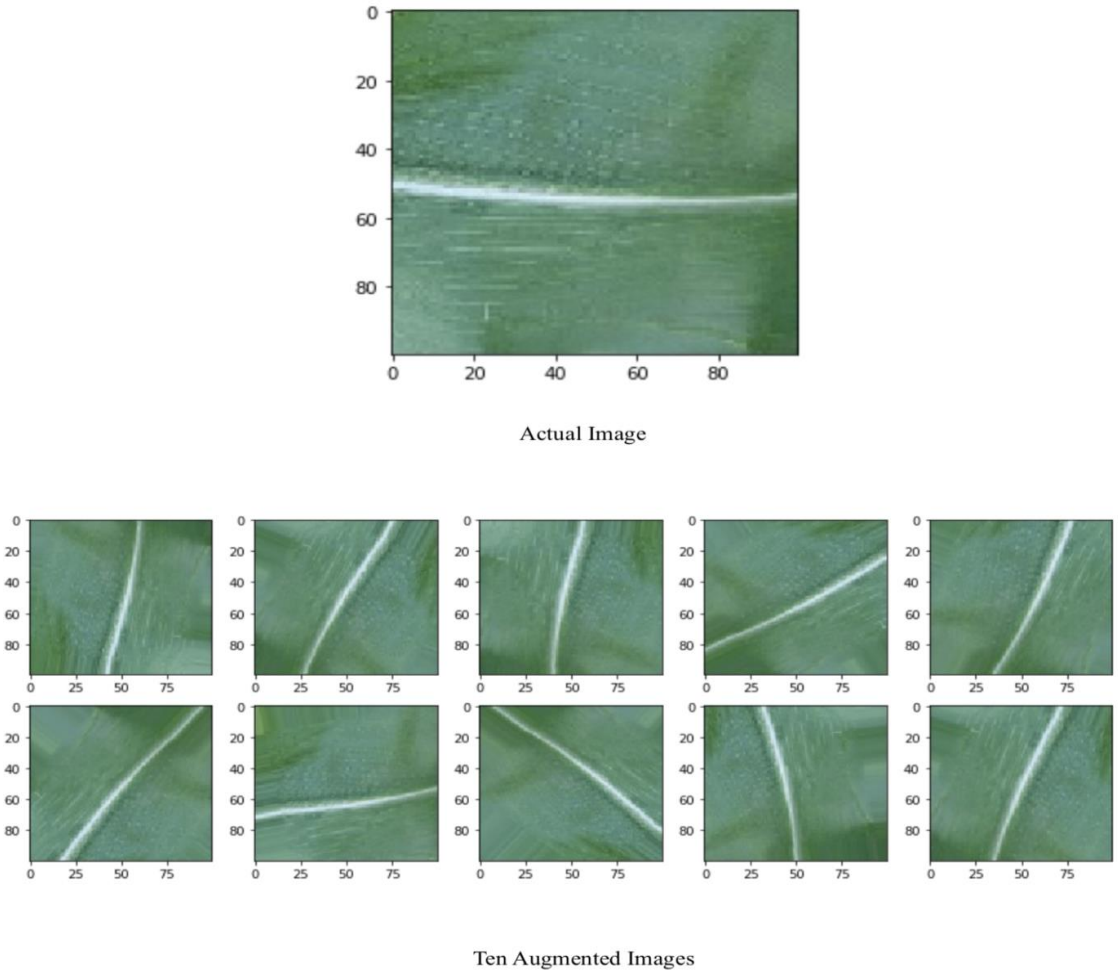


Figure 9: Samples after the data augmentation process

We implemented data augmentation techniques such as horizontal flipping, rotation, scaling, random zooming and translation with the ranges given below in Table 4.

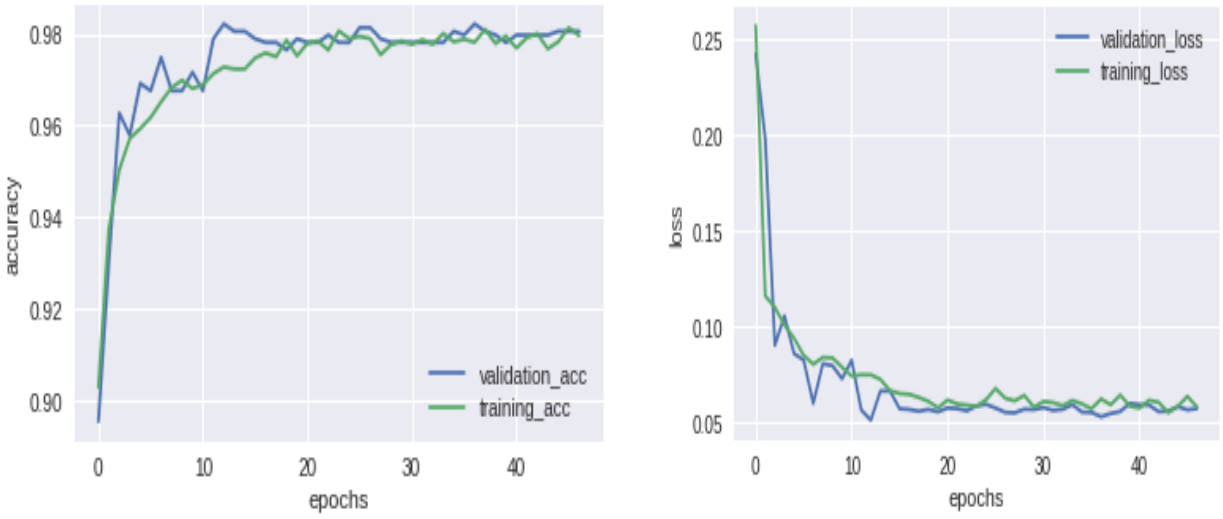
Table 4: Augmentation techniques with their range

TECHNIQUE	RANGE
Rescale	1/255.0
Rotation	45
Width Shift	0.2
Height Shift	0.2
Shear	0.4
Zoom	0.3

4.5 Results and Discussion

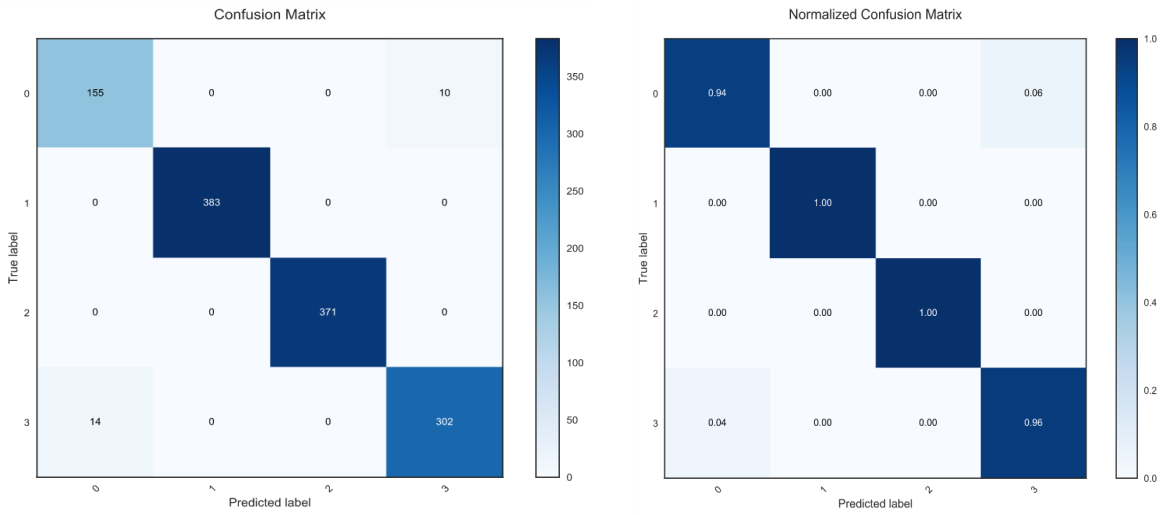
Initially, the model is trained with a learning rate of 0.001(default learning rate of Adam). After training the model for few epochs (monitor = 'val_acc', min_delta = 0.01, patience = 3), 96% training accuracy is achieved. Even though Adam uses exponential decay for the attenuation of the learning rate, a frequent fluctuation in the accuracy is witnessed due to high learning rate hence the model could not be considered as stable. Therefore, the training of the model started with 0.0005 learning rate with the same Early Stopping configuration, and after a few epochs we achieved ~98% training and validation accuracy. This time sporadic fluctuation is witnessed, so to make it stable we fine-tuned the learning rate up to 0.00001 and achieved 97.37% training accuracy with 2.53% error rate and 98.06% validation accuracy with 1.94% error rate and as we can see on the plot (Figure 10), the model was quite stable with negligible fluctuation in validation accuracy. While training the model, it is observed that a very low learning rate never progresses, and a high learning rate can cause instability hence never converges. Therefore, it is important to choose the right learning rate for the Adam, else the network will either fail to train or take much longer to converge.

The total number of parameters generated by the model are 77,612. The weights of the best fit model using ModelCheckpoint and used EarlyStopping to prevent our model from overfitting. In conclusion, after training the model for 47 epochs the overall validation(testing) accuracy achieved by the proposed architecture is 98.06%.



(a) (b)

Figure 10: (a) Accuracy plot and (b) Loss plot.



(a) (b)

{'Cercospora_leaf_spot Gray_leaf_spot': 0, 'Common_rust': 1, 'Healthy': 2, 'Northern_Leaf_Blight': 3}

Figure 11: Confusion matrix (a) Without Normalization (b) Normalized

The confusion matrixes are used to summarize the performance of the proposed classification model. In the blue colored diagonal of the matrix are the correct classifications, whereas all other entries are misclassifications. From the validation matrix, it can be seen that the classifier incorrectly recognized 16 images (10%) of class 0 as class 3. Images from Class 1 and 2 were classified accurately. Further, six images (2%) of class 3 are misclassified as class 0. High prediction scores in the diagonal indicate that the classifier is returning accurate results. Besides, it can be seen that the classes 0 and 3 are overlapping the most. Therefore, we can say that it is difficult to distinguish between Cercospora_leaf_spot Gray_leaf_spot and Northern_Leaf_Blight classes. A comprehensive classification reported has been prepared and drafted and in the Table 5.

Table 5: Classification report.

	Precision	**Recall**	**F1-score**	**Support**
0	0.92	0.94	0.93	165
1	1.00	1.00	1.00	383
2	1.00	1.00	1.00	371
3	0.97	0.96	0.96	316
Macro avg	0.97	0.97	0.97	1235
Weighted avg	0.98	0.98	0.98	1235
Accuracy		0.98		1235

Four other classification models are trained for corn leaf disease recognition and classification using existing CNN architectures (XceptionNet, EfficientNet, VGG19Net, and NASNet). Table 6 shows the comparative analysis of the proposed model with these models. As we can see in the table the validation accuracy of the proposed model is close to the state-of-the-art EfficientNet model and significantly higher than the other models. Also, the proposed model has only 0.0776 million parameters compare to the 4.4109 million parameters of the state-of-the-art EfficientNet model and 21.4194, 20.1891 and 4.5737 million parameters of the XceptionNet, VGGNet and NASNet respectively which results in lesser training time. All the mentioned CNN models were trained on the same dataset (with same hardware configuration) that we've used in our research. It can be seen clearly that the proposed optimized DenseNet with an overall accuracy of 98.06% is better than the established CNN models in terms of parameters and computation time.

Table 6: Comparison of the proposed model with other models

Model	**Accuracy**	**Parameters** (in millions)	**Training Time** (avg minutes per epoch)
EfficientNet-B0	99.84	4.41	4.88
VGG19Net	96.36	20.18	7.41
XceptionNet	93.52	21.41	10.00
NasNet	91.9	4.57	6.22
Optimized DenseNet	98.06	0.07	3.00

5. Conclusions and Future work

In this paper, we have presented an optimized dense CNN architecture (DenseNet) for corn leaf disease recognition and classification. We showed existing neural network architectures in the background and comprehensively discussed their strengths and weaknesses. We presented a step-by-step discussion of the proposed model so that it can be re-implemented for future purposes by other researchers. Extensive computer simulations have been performed to evaluate the performance of the proposed network. A robust experimental environment has been developed to conduct the simulations. Table 2 and 3 respectively show the hardware and software used with hyperparameters utilized to train the proposed network. We used a massive dataset of total 12332 pictures of 250 x 250 pixel dimensions for experimentations. The results achieved by the proposed DenseNet is impressive. It has achieved 98.06% accuracy in identifying three types of corn leaf diseases. The data augmentation method has been used to

increase the amount of relevant data. Hence, it helped in improving the generalization of the proposed model. The performance of the proposed DenseNet model has been tested by comparing its results with existing CNN architectures. Table 6 shows the comparative results based on performance metrics such as accuracy, parameters and training time. Based on the comparative analysis, we could conclude that the proposed DenseNet has outperformed over the existing CNN models implemented in this paper. Also, the approach presented in this research directly takes an image as input to CNN and learns itself to reach an effective recognition rate. Therefore, this approach is better than conventional machine learning techniques. The current results are encouraging; hence, this work can be extended by identifying more types of corn leaf diseases and creating a more optimized model with much fewer parameters and computation time. In future, we intend to develop a mobile app for corn leaf disease identification.

REFERENCES

[1]PRSLegislative Research. [Online]Available:https://www.prsindia.org/policy/discussion-papers/state-agriculture-India

[2] Kleffmann group press releases. [Online] Available:https://www.kleffmann.com/en/kleffmann-group/news--press/press-releases/india---maize-productivity-and-crop-potection/

[3] Farmers' portal. [Online] Available: https://farmer.gov.in/M_cropstaticsmaize.aspx

[4] Vijai Singh, A.K. Mishra, "Detection of plant leaf diseases using image segmentation and soft computing techniques", I*nformation Processing in Agriculture*, vol. 4, Issue 1, March 2017, pp. 41-49.

[5] Zhang Z., He X., Sun X., Guo L., Wang J., Wang F., "Image Recognition of Maize Leaf Disease Based on GA-SVM", *Chemical Engineering Transactions*, vol. 46, pp. 199-204, 2015.

[6] Enquhone Alehegn, "Maize Leaf Diseases Recognition and Classification Based on Imaging and Machine Learning Techniques.", *International Journal of Innovative Research in Computer and Communication Engineering*, Vol 5, Issue 12, Dec. 2017.

[7] Xihai Zhang, Yue Qiao, Fanfeng Meng, Chengguo Fan and Mingming Zhang, "Identification of Maize Leaf Diseases Using Improved Deep Convolutional Neural Networks.", *IEEE Access, China*, vol. 6, pp. 30370- 30377, June 2018.

[8] N. Kanaka Durga, G. Anuradha, "Plant Disease Identification Using SVM and ANN Algorithms", *International Journal of Recent Technology and Engineering*, Volume-7, Issue-5S4, Feb. 2019.

[9] Prakruti Bhatt, Sanat Sarangai, Anshul Shivhare, Dineshkumar Singh, and Srinivasu Pappula , "Identification of Diseases in Corn Leaves using Convolutional Neural Networks and

Boosting", *International Conference on pattern recognition applications and methods,* Feb. 2019.

[10] Alex Krizhevsky ,Ilya Sutskever and Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Commun. ACM,* vol. 60, pp.84-90, 2012.

[11] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun., "What is the best multi-stage architecture for object recognition?" , *12th International Conference on Computer Vision,* pp. 2146–2153, 2009.

[12] Y. LeCun, F.J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting", *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* vol. 2, pp. II-104, 2004.

[13] LeCun, Yann et al. "Convolutional networks and applications in vision." *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 253-256, 2010.

[14] Szegedy, Christian et al. "Going deeper with convolutions." *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1-9, 2014.

[15] LeCun, Yann et al. "Learning algorithms for classification: A comparison on handwritten digit recognition.", 1995.

[16] Zeiler, M.D., & Fergus, R., "Visualizing and Understanding Convolutional Networks" , *ArXiv preprint* ,vol. abs/1311.2901, 2013.

[17] Simonyan, K., & Zisserman, A.,"Very Deep Convolutional Networks for Large-Scale Image Recognition", *CoRR,* vol. abs/1409.1556, 2014.

[18] Subir Varma and Sanjiv Das, Deep Learning Book, 2018. [E-book] Available: https://srdas.github.io/DLBook/ ConvNets.html#convnet-architectures

[19] Chollet, François. "Xception: Deep Learning with Depthwise Separable Convolutions." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1800-1807, 2016.

[20] Mingxing Tan, Quoc V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", *ICML 2019.*

[21] Google AI Blog. [Online] Available:https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

[22] Zoph, Barret et al. "Learning Transferable Architectures for Scalable Image Recognition." *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition,* pp. 8697-8710, 2017.

[23] Zoph, Barret and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning.", *ArXiv preprint,* vol.abs/1611.01578, 2016.

[24] Huang, Gao et al. "Densely Connected Convolutional Networks.", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 2261-2269, 2016.

[25] Carneiro, Tiago et al., "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications.", *IEEE Access,* vol. 6, pp. 61677-61685, 2018.

[26] Google Colaboratory: Frequently Asked Questions. [Online] Accessed: Jun. 21, 2018. [Online]. Available:https://research.google.com/colaboratory/faq.html

[27] Google Colab specifications. [Online] Available: https://colab.research.google.com/drive/151805XTDg--dgHb3-AXJCpnWaqRhop_2

[28] Berkeley Artificial Intelligence Research. [Online] Available:https://bair.berkeley.edu/blog/2019/06/07/data_aug/

[29] He, Kaiming et al., "Deep Residual Learning for Image Recognition.", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, 2015.

[30] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth", *ECCV*, 2016.

[31] Agarap, Abien Fred, "Deep Learning using Rectified Linear Units (ReLU).", ArXiv preprint,vol.abs/1803.08375, 2018.

[32] Kingma, Diederik P. and Jimmy Ba.,"Adam: A Method for Stochastic Optimization.", *CoRR*, vol.abs/1412.6980, 2014.

[33] Zagoruyko, Sergey and Nikos Komodakis, "Wide Residual Networks.", *ArXiv*, vol. abs/1605.07146, 2016.

[34] Bengio, Yoshua et al.,"Learning long-term dependencies with gradient descent is difficult." , *IEEE transactions on neural networks*, vol.5 2, pp.157-66, 1994 .

[35] Duchi, John C. et al., "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.", J. Mach. Learn. Res., vol.12, pp. 2121-2159, 2012.

[36] VGG- Convolutional Network for Classification and Detection. [Online]Available:https://neurohive.io/en/popular -networks/vgg16/

[37] L Cunlou, G Shangbing, et al., "Maize Disease Recognition via Fuzzy Least Square Support Vector Machine.", Journal of Information and Computing Science; vol. 8: 316-320, 2013.

[38] A. Jindal, G. S. Aujla, N. Kumar, R. Chaudhary, M. S. Obaidat and I. You, "SeDaTiVe: SDN-Enabled Deep Learning Architecture for Network Traffic Control in Vehicular Cyber-Physical Systems," *in IEEE Network*, vol. 32, no. 6, pp. 66-73, November/December 2018.

[39] Miglani, Arzoo and Neeraj Kumar. "Deep learning models for traffic flow prediction in autonomous vehicles: A review, solutions, and challenges." *Veh. Commun.* 20 (2019): n. pag.

[40] S. Garg, K. Kaur, N. Kumar and J. J. P. C. Rodrigues, "Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective," *in IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 566-578, March 2019.

[41] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya and R. Ranjan, "A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networks," *in IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 924-935, Sept. 2019.

[42] G. S. Aujla et al., "DLRS: Deep Learning-Based Recommender System for Smart Healthcare Ecosystem," *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, 2019, pp. 1-6.

[43] Kang, C., Yu, X., Wang, S. H., Guttery, D., Pandey, H., Tian, Y., & Zhang, Y. A Heuristic Neural Network Structure Relying on Fuzzy Logic for Images Scoring. IEEE Transactions on Fuzzy Systems, 2020.

[44]. Pandey, Hari Mohan, and David Windridge. "A comprehensive classification of deep learning libraries." Third International Congress on Information and Communication Technology. Springer, Singapore, 2019.