Mango Leaf Disease Detection Using Image Processing

1st Avizit Sarkar
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
avizit.sarkar@g.bracu.ac.bd

2nd Murshed Hasan
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
murshed.hasan@g.bracu.ac.bd

3rd Nirnoy Chandra Sarker
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
nirnoy.chandra.sarker@g.bracu.ac.bd

4th Moin Nadim Srabon
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
md.moin.nadim.srabon@g.bracu.ac.bd

5th Safwat Sufia
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
Safwat.sufia@g.bracu.ac.bd

6th Md. Ashraful Alam, PhD
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
ashraful.alam@bracu.ac.bd

7th Md. Tanzim Reza
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
tanzim.reza@bracu.ac.bd

***Abstract*—Bangladesh is an agricultural country and mango cultivation plays a significant role in the economy of Bangladesh. Mango trees are at risk of different kinds of leaf disease. As a result, it can be the reason for hindering food production and quality substantially. So, it is very much important for the farmers to timely detection of these diseases. As a result, farmers can ensure stable production and supply. So, in this thesis, we have provided a custom convolutional neural network (CNN) architecture that was designed especially for mango leaf disease detection in Bangladesh. Our dataset consists of over 7,535 images that show both affected and healthy mango leaves, exposing nine different leaf classifications. We have trained our custom CNN model through both healthy and sick images so that it can easily distinguish between affected and non-affected mango leaves. We have compared our custom CNN model with a few pre-trained models which are MobileNetV2, VGG16, DenseNet169, and InceptionV3 to evaluate our model's performance and accuracy. So, the main motive of our thesis is to overcome the limitations of the previous research. Therefore, our suggested work is very much determined to be very accurate and to solve critical issues earlier researchers might have faced.**

***Index Terms*—Convolutional Neural Networks (CNN), Mango Leaf, Disease Detection, Deep Learning, Bangladesh, MobileNetV2, VGG16, DenseNet169 and InceptionV3.**

I. INTRODUCTION

The agricultural sector plays a significant role in Bangladesh and mango cultivation is one of the great contributors. But a number of dangerous leaf diseases such as root rot, powdery mildew, bacterial black spot, and anthracnose can greatly hamper the quality and production of mangoes. So, early detection of these diseases is very much necessary for farmers to ensure the healthiness and supply of mangoes according to the demand. Leaf diseases can have a range of symptoms, and current detection techniques may not be fully decent enough to detect diseases unique to a given area. Deep learning is one of the most powerful tools for picture recognition and classification in recent years, which is a subsection of machine learning. Our main objective is to create a customized model that can easily and precisely distinguish between healthy and affected mango leaves by utilizing deep learning architecture and CNNs. Hopefully, our customized model will highly contribute to the distinct circumstances and disease detection and can completely contribute to the enhancement of our economy and ensure the food security as well.

The better production of mangoes in Bangladesh depends on the early detection of affected mango leaves. However, manual detection of mango leaf diseases is a time-consuming and laborious process and in rural and backdated areas, farmers don't have that luxury and knowledge about disease symptoms and also don't have easy access to disease detection instruments. Although conventional machine learning methods have been designed for the purpose of detecting leaf diseases. But they aren't specially designed for mango leaf diseases in Bangladesh. Thus, this thesis is looking forward to addressing the creation of a precise, reliable, and customized deep learning-based architecture for mango leaf disease detection in Bangladesh. In order to overcome the limitations of conventional methods, this thesis involves creating a comprehensive dataset that captures around 13200 images along with both healthy and affected leaves.

To reach the main goal, we need to:

- Deep learning architecture creation, cutting-edge, for identifying and classifying fresh mango leaves and affected leaves.
- Development of a Disease Detection Process and Management. This would open up the possibility for farmers to detect several mango production-impacting leaf diseases at an early stage.
- Improve Sustainability of Agriculture: This would mean that farmers would control the loss and make a regular supply of quality mangoes to both the local and international markets.
- Accommodate Regional Specificities by obtaining a large dataset of healthy and infested leaves, capturing regional differences, seasons, and mango tree species.
- Create an easy and user-friendly tool or interface that works for farmers of all technological backgrounds. By uploading leaf photographs and receiving real-time updates, farmers will be able to take proactive steps and raise yields.

Thereafter, the farmer shall be made aware, alongside other stakeholders who help make up the agricultural community at large, of the tremendous benefits that this deep learning-based disease detection brings them.

II. Related Work

Singh et al. [1] created two datasets composed of images in the number of 2200 to classify mango leaves infected by anthracnose and healthy ones. Two datasets consisted of diseased and healthy mango leaves; another contained leaves of other plants that are infected with mango diseases. The authors used the MCNN method and provided accuracy of 97.16%. This accuracy is larger in comparison to conventional methods like PSO and SVM, RBFNN.

Arivazhagan and Ligi, 2018, [2] provided a model of CNN for mango leaf disease identification that had an accuracy of 96.67%. The paper presented in the International Journal of Pure and Applied Mathematics was inclined to the timely diagnosis so that effective treatment is rendered to avoid the loss of yield in cultivation. The authors compared the performance of CNN against traditional methods and brought out the precision and robustness of the former. This research concludes that deep learning methods are likely to play a very important role in enhancing the accuracy and efficacy of disease identification in agriculture.

Patel, Kar, and Khan [3] proposed a method of color computer vision to identify surface irregularities in mangos. Comparative analysis with these different existing techniques has been performed where image processing methods like segmentation, feature extraction, and also classification algorithms are used. This approach resulted in an efficiency of 93.3% and accuracy of 88.6%.

Aditya et al. [4] proposed an automated mango leaf disease detection system using transfer learning coupled with CNNs, where DenseNet201, InceptionResNetV2, InceptionV3, ResNet50, ResNet152V2, and Xception architectures were compared. In their case, they dealt with the class imbalance by segmentation. DenseNet201 performed the best with an accuracy of 98.00 and an F1 score of 98.33, while its precision, sensitivity, and specificity were 99.58%. This was followed by ResNet50, which had accuracy of 97.00%. InceptionResNetV2 and InceptionV3 achieved an accuracy of 96.67%. DenseNet201 and ResNet50 have been found to perform better in this study.

Rahaman et al. [5] developed a smartphone-based application that applies deep learning to identify mango diseases and suggests pesticides. The app developed DenseNet 169, InceptionV3, and MobileNetV2 for detecting diseases from images of infected and healthy leaves and fruits with an accuracy rate of 97.81%.

Veling et al. [6] proposed a GLCM integration technique with SVM to identify diseases in mangoes. GLCM computes the essential texture features concerning disease identification; then, these diseases are classified using SVM. In this regard, this methodology serves with an accuracy rate of 90% in disease identification to prevent a decline in mango production.

Trongtorkid et al. [7] presented a robust model for the identification of mango diseases using leaf symptoms. In this research, they have used CLIPS and MATLAB software to check the system using 100 images of healthy and infected mango leaves. The model provided an accuracy of 89.92%, high above methods available for disease detection.

III. Methodology

Figure 1 depicts the overall system for detection of mango leaf diseases. It comprises different phases. In the first phase, raw images are collected from different resources, followed by pre-processing and data augmentation. Afterwards, this data will be divided into a training, test, and validation set. In the second phase, the training data will be used to train the selected models. For the evaluation of classier performance, confusion matrix, precision, accuracy, recall, and F1-score will be used.

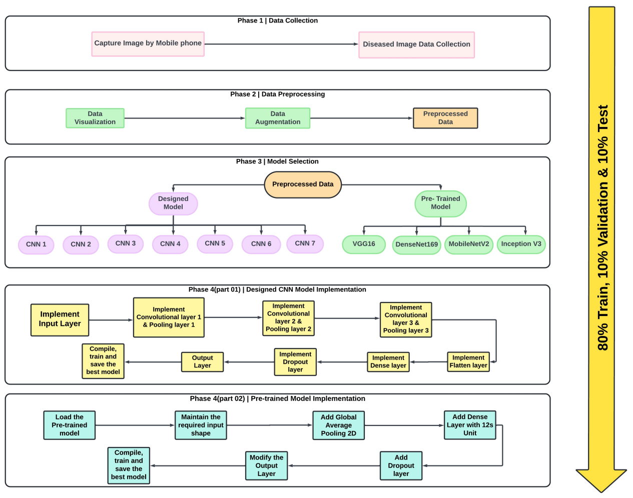

Phase 1 | Data Collection
Capture Image by Mobile phone
Diseased Image Data Collection
Phase 2 | Data Preprocessing
Data Visualization
Data Augmentation
Preprocessed Data
Phase 3 | Model Selection
Preprocessed Data
Designed Model
Pre- Trained Model
CNN 1
CNN 2
CNN 3
CNN 4
CNN 5
CNN 6
CNN 7
VGG16
DenseNet169
MobileNetV2
Inception V3
Phase 4(part 01) | Designed CNN Model Implementation
Implement Input Layer
Implement Convolutional layer 1 & Pooling layer 1
Implement Convolutional layer 2 & Pooling layer 2
Implement Convolutional layer 3 & Pooling layer 3
Implement Flatten layer
Implement Dense layer
Implement Dropout layer
Output Layer
Compile, train and save the best model
Phase 4(part 02) | Pre-trained Model Implementation
Load the Pre-trained model
Maintain the required input shape
Add Global Average Pooling 2D
Add Dense Layer with 12s Unit
Add Dropout layer
Modify the Output Layer
Compile, train and save the best model
80% Train, 10% Validation & 10% Test

Fig. 1. Top-level overview of the method

A. Dataset

Unless data standardization issues are addressed, Deep Learning has yet to see full deployment in agriculture. To this end, we created a comprehensive data set of mango leaves with images of 7,535 images of 1,907 unique leaves, representing eight different diseases from mango orchards in Bangladesh. Although the dataset contains leaves from farms in Bangladesh, the diseases represented relate to global contexts. This dataset shall be developed for facilitating deep learning research and applications in automated agriculture, probably drastically improving mango production in the world.

B. Data Collection Methodology

1) Data Collection

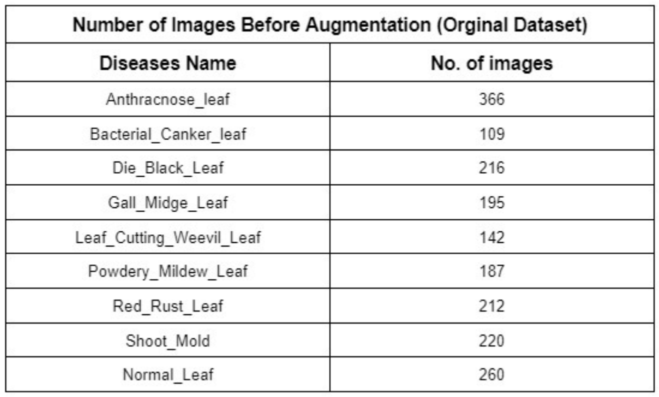
The dataset collection procedure involved several critical steps: first, we acquired background knowledge on common diseases affecting mango trees. We then selected suitable mango orchards for data collection with the help of agricultural experts. Images of both healthy and diseased mango leaves were captured directly from the trees, focusing on eight specific diseases. Additionally, we sourced images from various online platforms such as Krishi Batayon, Flickr, Mendeley, Google, and other relevant websites. [8] [9] [10] Krishi Batayon, a government agricultural website in Bangladesh, provided valuable insights into the symptoms of infected leaves, aiding in dataset construction, while the Plantix agricultural platform further enhanced our understanding of infected leaf images. Finally, we validated the dataset by manually labeling the images with the assistance of human experts, ensuring the accuracy and reliability of the collected data

C. Data Pre-processing

The data preprocessing has occurred in two stages, dataset visualization and data augmentation. These two techniques are described below.

1) Dataset Visualization

In our research, we curated a dataset comprising approximately 1907 images depicting both diseased and healthy mango leaves. In Figure 2, the number of images for each class is displayed in a table. Figure 3 presents a bar chart illustrating the distribution of images per class before augmentation. We observed that some images in our dataset were not sufficiently bright. To address this issue, we enhanced the brightness of these images, with the enhancement range set between 1.2 to 2. Additionally, we denoised some images to improve their quality.



Number of Images Before Augmentation (Orginal Dataset)	
Diseases Name	**No. of images**
Anthracnose_leaf	366
Bacterial_Canker_leaf	109
Die_Black_Leaf	216
Gall_Midge_Leaf	195
Leaf_Cutting_Weevil_Leaf	142
Powdery_Mildew_Leaf	187
Red_Rust_Leaf	212
Shoot_Mold	220
Normal_Leaf	260

Fig. 2. Number of Images Table Before Augmentation (Original Dataset)

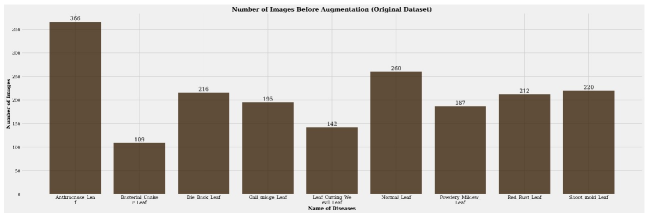
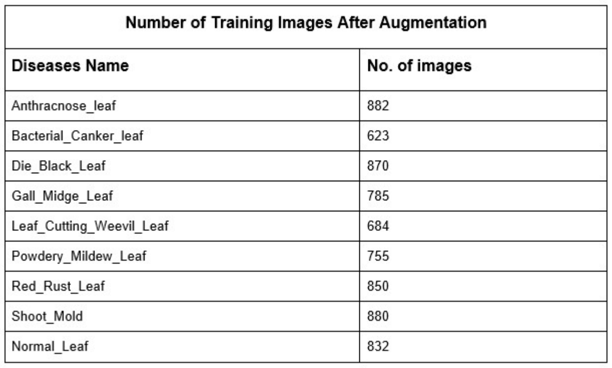


Fig. 3. Number of Images Before Augmentation (Original Dataset)

D. Data Augmentation

Data augmentation improves the performance of a machine learning model by creating varied training examples from existing data; this is done with domain-specific strategies. Extensive datasets improve model accuracy. In the image data augmentation, shifts, flips, and zooms will create modified duplicates, all belonging to the same class. Traditional methods are geometric transformations, color variations, rotation, reflection, and noise injection, all enhancing the performance of the model. We deployed augmented techniques using TensorFlow-based augmentation, whereas in our study, rotation and flipping were applied only on the training data. Figure 4 represents the number of images after augmentation per class. Figure 5 shows the distribution of the images after Augmentation.



Number of Training Images After Augmentation	
Diseases Name	**No. of images**
Anthracnose_leaf	882
Bacterial_Canker_leaf	623
Die_Black_Leaf	870
Gall_Midge_Leaf	785
Leaf_Cutting_Weevil_Leaf	684
Powdery_Mildew_Leaf	755
Red_Rust_Leaf	850
Shoot_Mold	880
Normal_Leaf	832

Fig. 4. Number of Traning Images Table After Augmentation

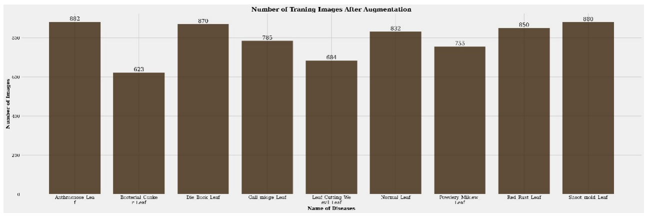


Fig. 5. Number of Traning Images After Augmentation

E. Pre-processed Data

We successfully pre-processed our data, preparing it for effective model training and reliable performance. Initially, our dataset comprised 1,907 original images. We then split the dataset, allocating 80% for training, 10% for testing, and 10% for validation. This resulted in 1,533 images for training, 187 for testing, and 187 for validation. As previously mentioned, we only augmented our training data. After augmentation, the total number of training images increased to 7,161. Figure 6 shows the distribution of the dataset after splitting.

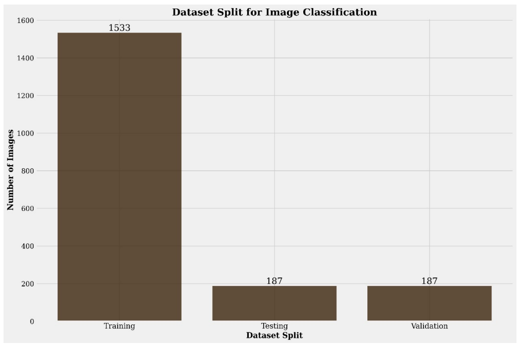


Fig. 6. Dataset Split for Image Classification

F. Model Specification

1) Convolutional Neural Network(CNN)

This is a very popular and most commonly used ML model that Rajbongshi and his colleagues [4] have used and described extensively in their paperwork.

2) VGG16

This is another very popular and robust model that Kaur, Gurjot, and their colleagues [11] have used and described extensively in their paperwork.

3) MobileNet V2

This is another widely used model that Ratha, Ashoka Kumar, and their colleagues [12] have used and described extensively in their research paper.

4) Inception V3

Mahato and his colleagues have a well-written description of this model in their recent study on tomato leaf detection. [13]

5) DenseNet169

Tri Wahyuningrum along with this colleagues has given an extensive description of this model in their paperwork. [14]

IV. Implementation

A. Workflow

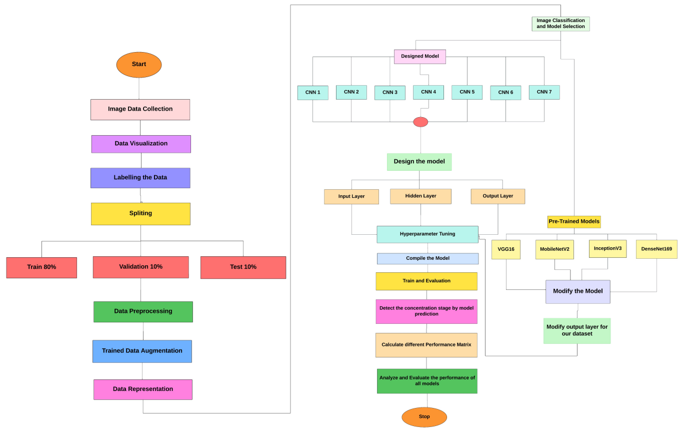


Fig. 7. Workflow Details for Implementation

Our research started with the development of a dataset designed for mango leaf disease detection. We constructed an extensive dataset by capturing images from several mango gardens, trees, and regions of Bangladesh. We have also followed the data collection procedure quite extensively to ensure adequate variations in the conditions of leaves and expressions of diseases.

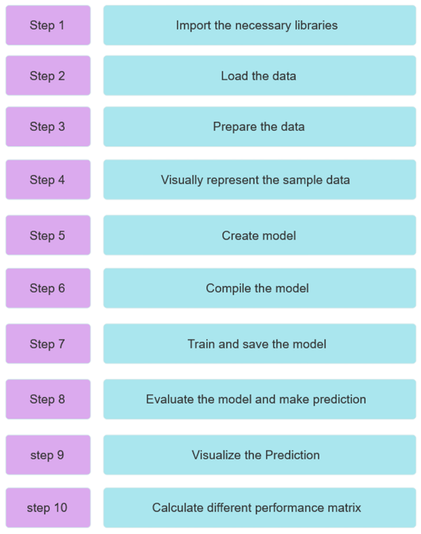
The dataset was divided systemically into segments for training, validation, and testing. A proportion of about 80% was set to be used for the training of deep learning models, 10% would be for validation of their performance, while the last 10% would be needed for final testing in order to bring out examples for models' accuracy and generalization capabilities.

This was a vital step in the process: model selection. It was that step in which we identified and evaluated appropriate models. We based our choice on two sets of models: custom models based on CNN architectures like ConvolutionNet-5, ConvolutionNet-3, ConvolutionNet-4, ConvolutionNet-4M1, ConvolutionNet-5M0, ConvolutionNet-5M1, and ConvolutionNet-6; pre-trained models such as VGG16, MobileNetV2, InceptionV3, and DenseNet169, with an inherently strong basis because of high performance themselves in many image classification tasks.

Setting up and tuning their hyper-parameters to optimize their performance followed the selection of the models. We implemented training and evaluation for both specially designed models and selected fine-tuned pre-trained models on our created dataset of mango leaves, which provided information about the most promising ones for detecting diseases and classifying them on mango leaves; hence, the results could suggest how feasible the solution was for farmers to upgrade the cultivation practice of mangoes.

Figure 7 describes the research plan for our project, outlining the design process of our specialized dataset and how we have consulted models that best fit our needs for this purpose. It shows, therefore, a thorough and entire cycle for the deep learning technique to be employed for the accurate detection of mango leaf diseases.

B. Structural View of The Skeleton



Step 1	Import the necessary libraries
Step 2	Load the data
Step 3	Prepare the data
Step 4	Visually represent the sample data
Step 5	Create model
Step 6	Compile the model
Step 7	Train and save the model
Step 8	Evaluate the model and make prediction
step 9	Visualize the Prediction
step 10	Calculate different performance matrix

Fig. 8. Structural View of the Skeleton

C. Model Selection

We focus on selecting lightweight models that efficiently classify images in the task of concentration measurement; we will apply a CNN-based architecture in this context, including ConvolutionNet-5, ConvolutionNet-3, ConvolutionNet-4, ConvolutionNet-4M1, ConvolutionNet-5M0, ConvolutionNet-5M1, and ConvolutionNet-6. Our proposed customized model strikes a very efficient balance between accuracy and computational efficiency by being light, achieving very impressive accuracy.

For instance, in our project, four lightweight architecture models that are very well-known were chosen: VGG16, MobileNetV2, InceptionV3, and DenseNet169. Such models will offer a wide spectrum of performance-accuracy combinations.

Model selection was aimed at robustness and unique performance. Handling the transfer learning layer was critical to ensure that models fitted our measurement task despite being trained on nothing similar to our data before. These two stages of selection and fine-tuning have been dealt with cautiously, improving the performance and efficiency of our workflow. Figure 9 shows the diagram of the model selection.

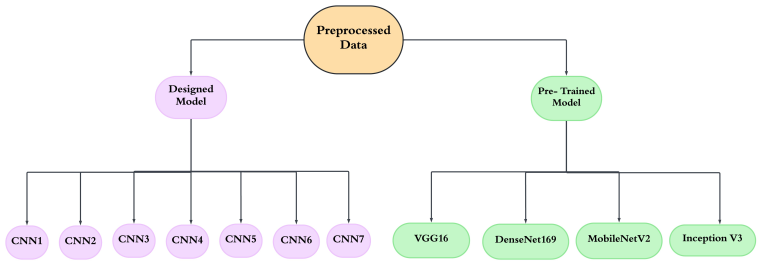


Fig. 9. Diagram of model selection

D. Hyperparameter Tuning

Hyperparameters are very important settings which are provided while calling a function or a class. These significantly impact the efficiency and performance of an algorithm. Proper choice of hyper-parameters is important in optimizing model performance.

For instance, during our training, Adam as the optimizer used an initial learning rate of 0.001 that turned out to show low accuracy on validation. To this effect, dropping the learning rate between 0.00001 and 0.000001 improved the validation accuracy drastically. Another case was that by setting 'steps per epoch' correctly to the number of images in the dataset used for training divided by the batch size, we made sure that the model would train based on all data, returning expected validation accuracy.

E. Design and Compile the Models

ConvolutionNet-5: This includes five convolutional layers with increasing filters, 16, 32, 64, 128, and 256, followed by max-pooling layers with ReLU activation. In addition, it has Dense layers instituted with ReLU activation, BatchNormalization layers, a dropout layer with 0.5, and finally, at the end, an output layer with softmax. Models use Adam optimizer with lr=0.00001, sparse categorical cross-entropy loss, and accuracy as a metric.

ConvolutionNet-5M0: It consists of five convolutional layers in series, filter of 32, 64, 128, 256, 512, with all being followed by max-pooling and arrogant ReLU. Then there are the Dense layers of 512, 1024, again followed by ReLU, BatchNormalization, and a Dropout layer of 0.5, followed by the final output layer of eleven neurons with softmax. It will be compiled using the Adam optimizer with a Learning Rate of 0.0001 and Sparse Categorical Cross-Entropy Loss with the accuracy metric.

ConvolutionNet-5M1: Five convolutional layers with max-pooling and ReLU: 16, 32, 64, 128, 256; Dense layers: 256, 512 – ReLU, BatchNormalization; Dropout Layer: 0.4; Output layer: eleven neurons, softmax. Optimizer: AdamW, lr=0.00001, sparse categorical cross-entropy loss, metric: accuracy.

ConvolutionNet-4: four convolutions with max-pooling and ReLU: 32, 64, 128, 256; more Dense layers of 512 and 256 with ReLU and BatchNormalization; a dropout layer of 0.5; an Output Layer with eleven neurons and softmax. Setup with the Adam optimizer (lr=0.00001), sparse categorical cross-entropy loss, and accuracy metric.

ConvolutionNet-4M1: It consists of four convolutional layers of max-pooling and ReLU. It has Dense layers of sizes 256, 512, with ReLU, BatchNormalization, a Dropout layer of 0.5, and eleven neurons in the output layer with softmax. Compiled with the Adam optimizer, lr=0.00001; sparse categorical cross entropy loss; and accuracy metric..

ConvolutionNet-6: six convolutional layers, 16, 32, 64, 128, 256, 512, with max-pooling and ReLU. It has Dense layers of 256, 256 with ReLU activation, BatchNormalization, a dropout layer of 0.5, and an output layer of eleven neurons upon softmax. Configured with Adam optimizer, learning rate of 0.00001; loss: sparse categorical cross-entropy measure; metrics: accuracy.

ConvolutionNet-3: It is based on the sequential model with three convolutional layers. Convolutional layers use 16, 32, and 64 filters, respectively; all of them are coupled with max-pooling and ReLU. Further, it consisted of a Dense layer of 256 neurons with ReLU, BatchNormalization, and a Dropout layer of 0.2, followed by an output layer of eleven neurons with softmax. It was also compiled with the Adam optimizer, sparse categorical cross-entropy loss as the loss function, and accuracy as the metric.

VGG16: VGG16 is fine-tuned by removing its original fully connected layers and adding a global average pooling layer. After that, dense layers of size 512 and 256 with the ReLU activation function and a dropout of 0.5 each are added. The output layer consists of three neurons with softmax activation. It freezes the top and base layers to retain only weights. This model is now compiled with the Adam optimizer with lr= 0.00001, sparse categorical cross-entropy loss, and accuracy as metric.

InceptionV3: This pre-trained model replaces fully connected layers of the following architecture: global average pooling, after which come dense layers of 128 and 512 neurons with ReLU and a dropout of 0.5. There will also be an added output layer with three neurons with softmax activation. The top and base layers are put in the frozen state to retain weights. It will be trained with the Adam optimizer, a learning rate of 0.00001, sparse categorical cross-entropy as the loss function, and accuracy as the metric.

MobileNetV2: This pretrained model is fine-tuned by discarding its fully connected layers and adding a global average pooling. Dense layers with 128 and 512 neurons, ReLU activation, and a dropout layer at a rate of 0.5 have been attached at the end. On the top, there is an output layer with three neurons and softmax activation. The top and base layers are frozen so that the weights get maintained. Here, the Adam optimizer is used with a learning rate of 0.00001, sparse categorical cross-entropy as the loss function, and accuracy as the performance metric.

DenseNet169: In this version, the DenseNet169 architecture will be followed; it provides a pre-trained model wherein the initially developed fully connected layers have been removed and replaced with a global average pooling layer. After that, dense layers of 128 neurons and 512 neurons with ReLU activation will be added; then, a dropout layer at a rate of 0.5 is appended. Finally, the output layer is set to have three neurons using the softmax activation function. It is also frozen at the top and base layers for preserving weights. Training in this case would be done using the Adam optimizer with a learning rate of 0.00001, sparse categorical cross-entropy as the loss function, and accuracy as a metric.

V. Results Analysis and Discussion

A. *Train and evaluate the Models*

1) *ConvolutionNet-5*

We trained the model for 50 epochs with a batch size of 8. Callbacks monitored and saved the best model based on validation accuracy. It gave test accuracy of 0.9411 and validation accuracy of 0.9130. Training history was visualized graphically, showing training versus validation accuracy and loss. Figures 10 and 11 illustrate training vs. validation accuracy and training vs. validation loss, respectively.

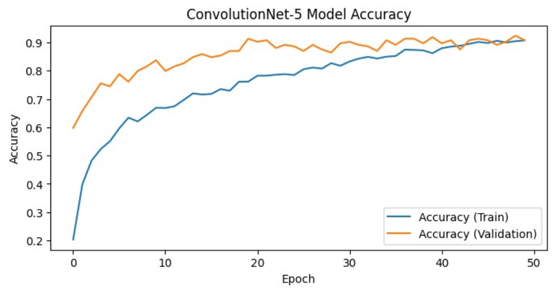


Fig. 10. Training_accuracy Vs Validation_accuracy of ConvolutionNet-5

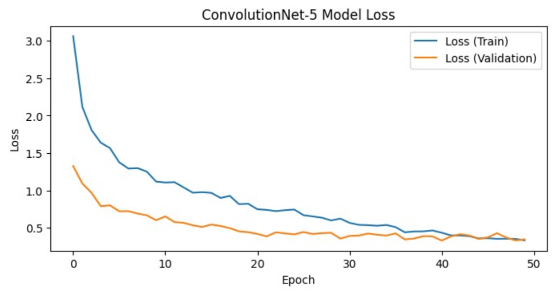


Fig. 11. Training_loss Vs Validation_loss of ConvolutionNet-5

2) *ConvolutionNet-3*

The model is trained with a batch size of 8 for 50 epochs. Callbacks monitor and save the model based on maximum validation accuracy. Test accuracy was 0.8877 and validation accuracy was 0.8478. Figures 12 and 13 show training versus validation accuracy and training versus validation loss, respectively.

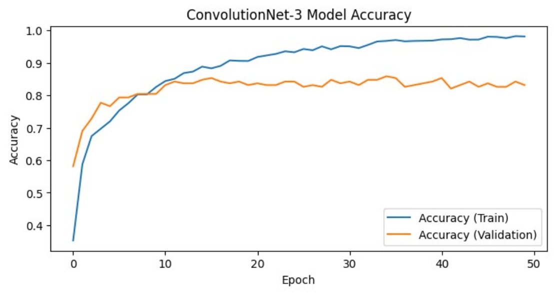


Fig. 12. Training_accuracy Vs Validation_accuracy of ConvolutionNet-3

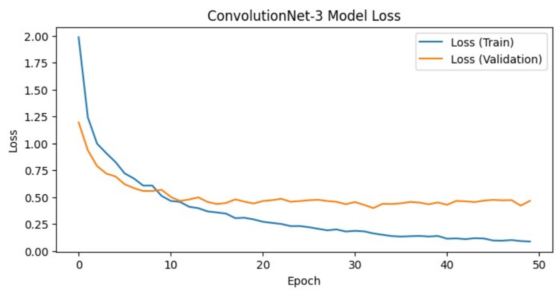


Fig. 13. Training_loss Vs Validation_loss of ConvolutionNet-3

3) *ConvolutionNet-4*

The model was trained on training and validation datasets for 50 epochs with a batch size of 8. Callbacks monitor and save the model based on maximum validation accuracy. Test accuracy is 0.9144 and validation accuracy is 0.8804. Figures 14 and 15 show training versus validation accuracy, and training versus validation loss.

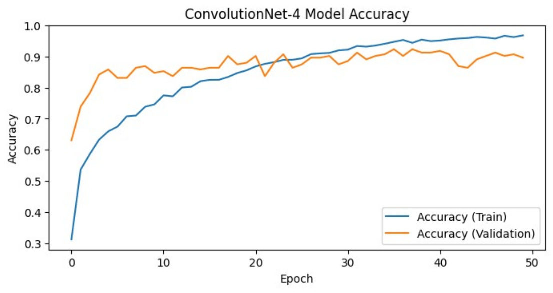


Fig. 14. Training_accuracy Vs Validation_accuracy of ConvolutionNet-4

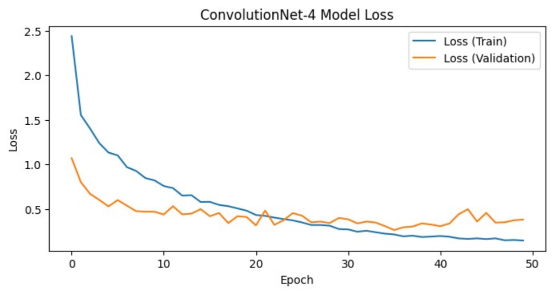


Fig. 15. Training_loss Vs Validation_loss of ConvolutionNet-4

4) *ConvolutionNet-4M1*

This model is trained with a batch size of 8 for the training and validation datasets for a total of 50 epochs. The creation of the model involves callbacks to monitor and save based on maximum validation accuracy. The test accuracy was 0.8983, and the validation accuracy is 0.8967. Figures 16 and 17 shows the Training vs. Validation Accuracy and Training vs. Validation Loss.

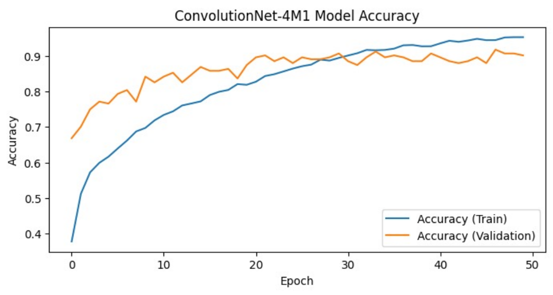


Fig. 16. Training_accuracy Vs Validation_accuracy of ConvolutionNet-4M1

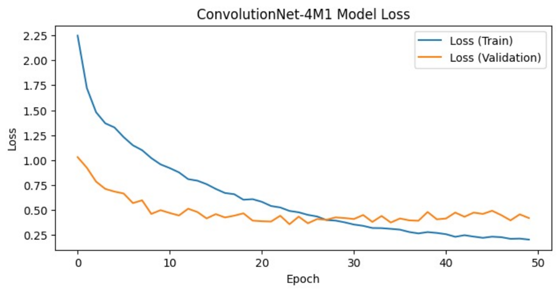


Fig. 17. Training_loss Vs Validation_loss of ConvolutionNet-4M1

5) *ConvolutionNet-5M0*

It trains the model with a batch size of 8 on both the training and validation sets for 50 epochs. Callbacks track and save the model based on the best validation accuracy. The test accuracy has a value of 0.8823, and validation accuracy equals 0.9022. Figures 18 and 19 depict the training vs. validation accuracy and the training vs. validation loss, respectively.

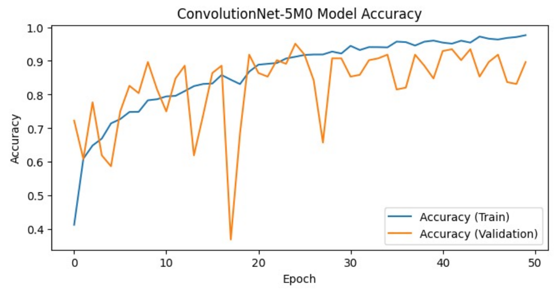


Fig. 18. Training_accuracy Vs Validation_accuracy of ConvolutionNet-5M0

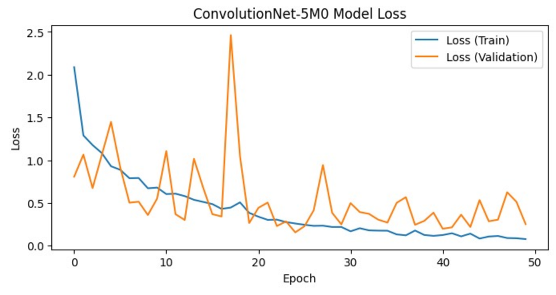


Fig. 19. Training_loss Vs Validation_loss of ConvolutionNet-5M0

6) *ConvolutionNet-5M1*

It is trained with a batch size of 16 on both train and validation datasets across 50 epochs. Callbacks are used to monitor and save the model on the basis of highest validation accuracy attained. Test Accuracy: 0.9144, Val Accuracy: 0.8920. Figures 20 and 21 show training vs. validation accuracy and training vs. validation loss respectively.

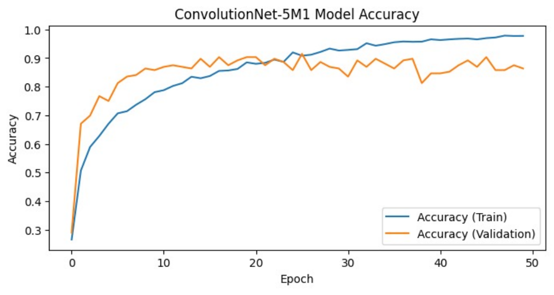


Fig. 20. Training_accuracy Vs Validation_accuracy of ConvolutionNet-5M1

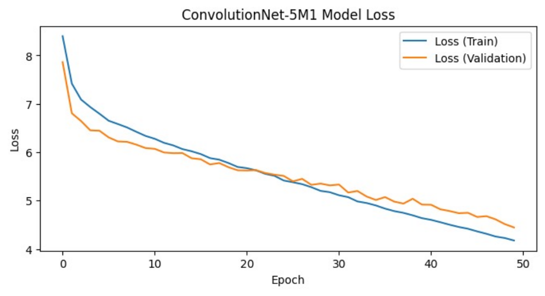


Fig. 21. Training_loss Vs Validation_loss of ConvolutionNet-5M1

7) *ConvolutionNet-6*

The model is trained with a batch size of 8 on both train and validation datasets across 50 epochs. Callback is used to track and save a model based on the highest validation accuracy obtained. Test accuracy is 0.8823, with validation accuracy equal to 0.8587. Figures 22 and 23 represent training vs. validation accuracy and training vs. validation loss, respectively.

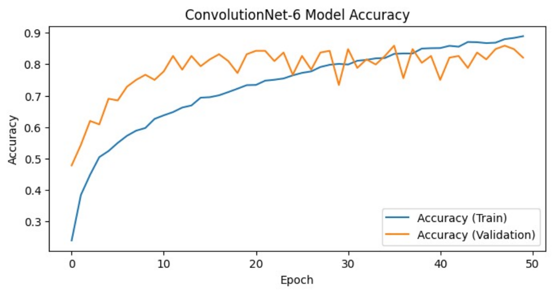


Fig. 22. Training_accuracy Vs Validation_accuracy of ConvolutionNet-6

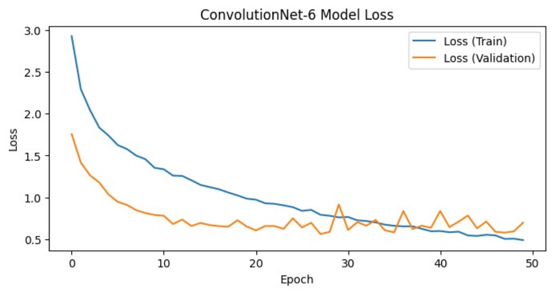


Fig. 23. Training_loss Vs Validation_loss of ConvolutionNet-6

8) *VGG16*

The model is trained with a batch size of 8, utilizing both train and validation datasets, run for 50 epochs. Callbacks are used to monitor and save the model based on the maximum validation accuracy attained. Test Accuracy: 0.8770, Validation accuracy 0.8261. Figures 24 and 25 show training vs validation accuracy and training vs validation loss, respectively.

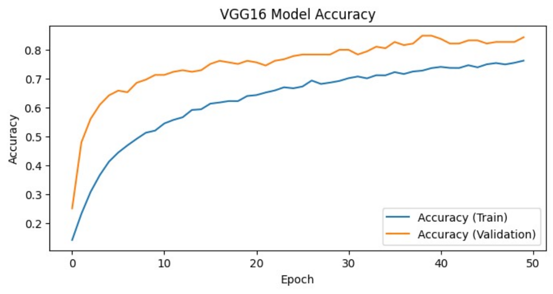


Fig. 24. Training_accuracy Vs Validation_accuracy of VGG16

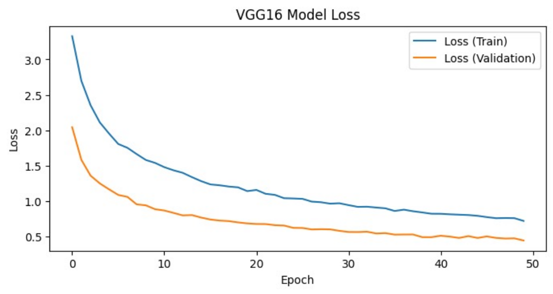


Fig. 25. Training_loss Vs Validation_loss of VGG16

9) *MobileNetV2*

It is trained with a batch size of 8 on both train and validation datasets for epochs of 50. Callbacks monitor and save the model based on the highest validation accuracy achieved. Test accuracy is 0.8021, validation accuracy is 0.8152. Figures 26 and 27 compare training vs. validation accuracy, and training vs. validation loss, respectively.

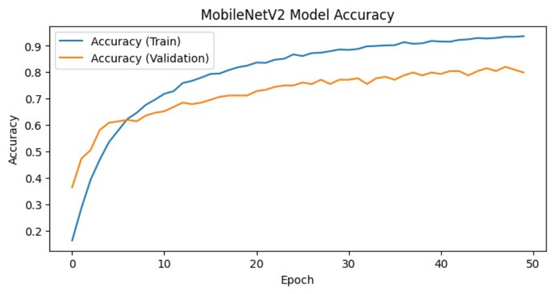


Fig. 26. Training_accuracy Vs Validation_accuracy of MobileNetV2

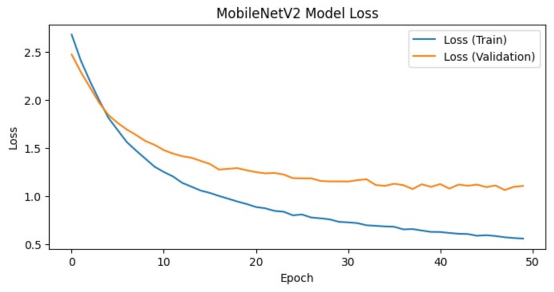


Fig. 27. Training_loss Vs Validation_loss of MobileNetV2

10) *InceptionV3*

The model is trained on a batch size of 8, using both the train and validation datasets for 50 epochs. Callbacks monitor and save the model maximum validation accuracy. The results in test accuracy were 0.8823, while that in validation accuracy is 0.8533. Figures 28 and 29 drawing train accuracy versus validation accuracy and training loss versus validation loss, respectively.

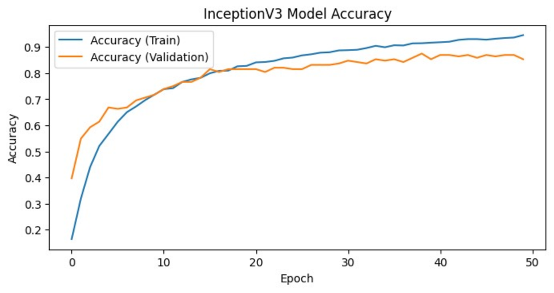


Fig. 28. Training_accuracy Vs Validation_accuracy of InceptionV3

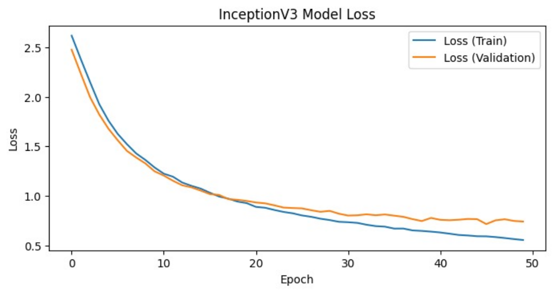


Fig. 29. Training_loss Vs Validation_loss of InceptionV3

11) *DenseNet169*

It was trained with a batch size of 8, both train and validation datasets, for 50 epochs. Callbacks were used, monitoring validation accuracy and saving the model producing the highest validation accuracy. It achieved an accuracy of 0.8342 on the test set and an accuracy of 0.8696 on the validation set. Figures 30 and 31 plot training accuracy versus validation accuracy and training loss versus validation loss, respectively.

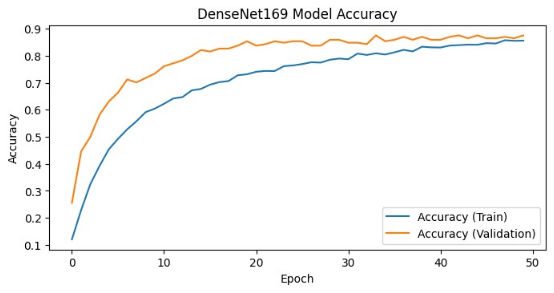


Fig. 30. Training_accuracy Vs Validation_accuracy of DenseNet169

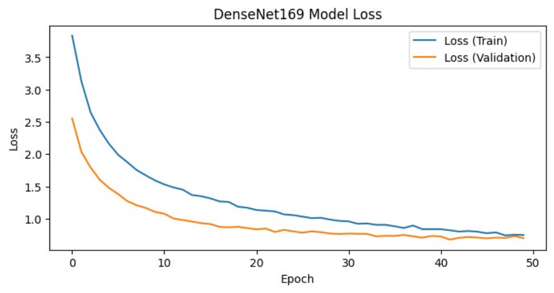


Fig. 31. Training_loss Vs Validation_loss of DenseNet169

B. Ablation Study of Custom Convolutional Neural Networks for Mango Leaf Disease Classification

In this section, we have observed and compared the performance and accuracy of seven custom (CNNs) architecture designed for the variation or classification of various mango leaf diseases. These models are different in their architectures and characteristics, including the number of convolutional layers, dense layers, filter sizes, optimizers, learning rates, batch sizes, and dropout rates. The main goal of this study is to critically analyze the performance of these architectures on model performance in terms of accuracy, loss, precision, recall, and F1-score.

1) Model Architectures and Performance Metrics

ConvolutionNet-5: It has 5 Conv2D layers with 16, 32, 64, 128, and 256 filters, along with 2 dense layers with 256 and 512 neurons. Here the architecture is using Adam optimizer with a learning rate of 0.00001. Along with a batch size and a dropout rate of 8 and 0.5 after each dense layer. This model has Test Accuracy: 0.9411 and Validation Accuracy: 0.9130 with a loss of 0.4533 and a validation loss of 0.3590. The following classification report indicates high precisions, recalls, and F1-scores across most classes, except for small and higher averages of 0.94.

ConvolutionNet-3: This is a Conv2D layer having 16, 32, and 64 filters with a dense layer composed of 512 neurons. This setup was made with the learning rate of the Adam optimizer equal to 0.000001; the batch size is 8, and the use of a 0.2 dropout rate. Test accuracy from this model was 0.8877 and validation accuracy was 0.8478, and the loss and validation loss are equal to 0.1629 and 0.3984. Furthermore, contrary to ConvolutionNet-5, for this architecture, precision, recall, and F1-score are lower, with averages of 0.88 and 0.89, respectively.

ConvolutionNet-4: It consists of 4 Conv2D layers with 32, 64, 128, and 256 filters, and 2 dense layers with 512 and 256 neurons. It uses the Adam optimizer, with a learning rate set to 0.00001, along with a batch size and a dropout rate set to 8 and 0.5, respectively, after each dense layer. This model has given a test accuracy of 0.9144 and validation accuracy of 0.8804 with a loss of 0.4024 and validation loss of 0.3213. The model has turned in really nice performance, with precisions, recalls, and F1-scores smaller and weighted averages of 0.91.
ConvolutionNet-4M1: It features 4 Conv2D layers with 16, 32, 64, and 256 filters, and 2 dense layers with 256 and 512 neurons. The Adam optimizer has been used with an effective learning rate set to 0.00001, and batch size is 8 with a dropout of 0.5 after every dense layer. This architecture showed a test accuracy of 0.8983 and a validation one of 0.8967, with a loss and validation loss including 0.2676 and 0.3974, which indicates only a bit better performance with ConvolutionNet-4 than ConvolutionNet-4 in terms of validation accuracy but with the smaller weighted average precision-recall F1-score of 0.90.

ConvolutionNet-5M1: It comprises 5 Conv2D layers with 16, 32, 64, 128, and 256 filters, and 2 dense layers with 256 and 512 neurons. The Adam optimizer is used in this architecture, assuming the learning rate of 0.00001, and with a batch size of 16 and a dropout rate of 0.4 after each dense layer. Obviously, this creates the test accuracy of 0.9144 and validation accuracy of 0.8920, but with a loss and validation loss of 5.0719 and 5.1669, respectively. Even though the accuracy is high in the benchmark, the loss values are at a high, showing problems with potential overfitting. The smaller weighted average precision, recall, and F1 score are all at 0.91 and 0.92, respectively.

ConvolutionNet-5M0: It includes 5 Conv2D layers with 32, 64, 128, 256, and 512 filters, and 2 dense layers with 512 and 1024 neurons. The model has applied the Adam optimizer as a learning rate of 0.0001 and including a batch size and dropout rate of 8, and 0.5 after each dense layer. This model has achieved d a test and validation accuracy of 0.8823 and 0.9022, with a loss of 0.1101 and a loss of validation which is 0.3636. Because of high variance this model's performance is hindered, as indicated by its spiky loss graphs, with macro and higher averages for precision, recall, and F1-score of 0.89 and 0.88, respectively.

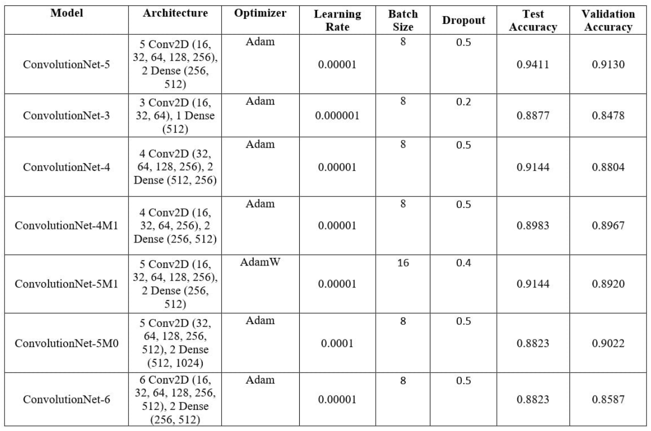
ConvolutionNet-6: It has 6 Conv2D layers with 16, 32, 64, 128, 256, and 512 filters, and 2 dense layers with 256 and 512 neurons. It is using a learning rate of 0.00001 as Adam optimizer. This model also has a batch size of 8, and a dropout rate of 0.5 after each dense layer. This architecture has shown a test and validation accuracy of 0.8823 and 0.8587, with a loss of 0.6757 and a validation loss of 0.5820. This model is suffering from higher variance and instability just similar to ConvolutionNet-5M0, with precision, recall, and F1-score averages of 0.88 for both smaller and weighted metrics.

2) *Analysis*

First and foremost, comparison from the architectures nails ConvolutionNet-5 at the top with high test and validation accuracies while keeping the loss at a minimum. It shows moderate to strong robustness, as the precision, recall, and F1-scores are almost completely identical for all the classes. On the other hand, ConvolutionNet-3, with fewer layers and lower dropout rates, works quite decently but returns extremely lame precisions and recalls among some features, like the Powdery Mildew Leaf, which makes it weak in feature capture.

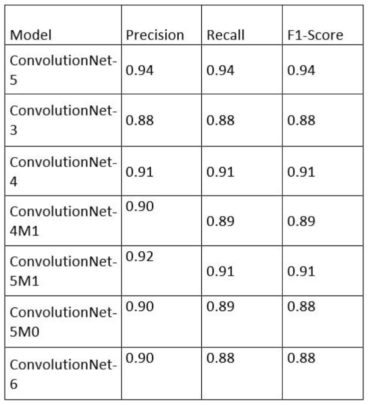
Additional layers and increased dropout rates make ConvolutionNet-4 and ConvolutionNet-4M1 perform better than ConvolutionNet-3. While many would hope that mixed filter sizes increase performance significantly due to ConvolutionNet-4M1, it does not perform much better than ConvolutionNet-4. ConvolutionNet-5M1 achieves high accuracies but has higher loss values, which may indicate some problems, either with overfitting or the optimizer. Other models, including ConvolutionNet-5M0 and ConvolutionNet-6, are complex, have high variances, and can be highly unstable. This essentially explains that a too complicated model is not a guarantee that good results must be achieved all the time; it might result in overfitting.

The overall message of this study is that model complexity needs to be modulated using the ability of generalization. Although performance can be improved by increasing the number of layers and dropout rates, in some cases unavoidable overfitting and instability result from careless over-tuning of hyper-parameters. ConvolutionNet-5 also manifests this balance between them and enables high efficiency with optimal performance by architectural choices.



Model	Architecture	Optimizer	Learning Rate	Batch Size	Dropout	Test Accuracy	Validation Accuracy
ConvolutionNet-5	5 Conv2D (16, 32, 64, 128, 256), 2 Dense (256, 512)	Adam	0.00001	8	0.5	0.9411	0.9130
ConvolutionNet-3	3 Conv2D (16, 32, 64), 1 Dense (512)	Adam	0.000001	8	0.2	0.8877	0.8478
ConvolutionNet-4	4 Conv2D (32, 64, 128, 256), 2 Dense (512, 256)	Adam	0.00001	8	0.5	0.9144	0.8804
ConvolutionNet-4M1	4 Conv2D (16, 32, 64, 256), 2 Dense (256, 512)	Adam	0.00001	8	0.5	0.8983	0.8967
ConvolutionNet-5M1	5 Conv2D (16, 32, 64, 128, 256), 2 Dense (256, 512)	AdamW	0.00001	16	0.4	0.9144	0.8920
ConvolutionNet-5M0	5 Conv2D (32, 64, 128, 256, 512), 2 Dense (512, 1024)	Adam	0.0001	8	0.5	0.8823	0.9022
ConvolutionNet-6	6 Conv2D (16, 32, 64, 128, 256, 512), 2 Dense (256, 512)	Adam	0.00001	8	0.5	0.8823	0.8587

Fig. 32. Ablation Study Summary Table 1

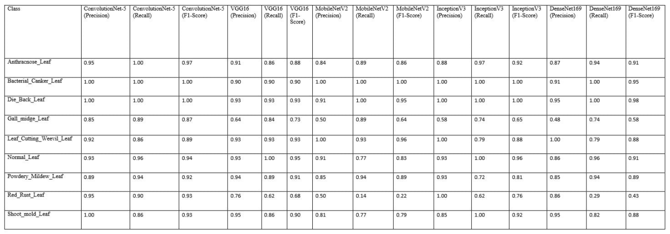


Model	Precision	Recall	F1-Score
ConvolutionNet-5	0.94	0.94	0.94
ConvolutionNet-3	0.88	0.88	0.88
ConvolutionNet-4	0.91	0.91	0.91
ConvolutionNet-4M1	0.90	0.89	0.89
ConvolutionNet-5M1	0.92	0.91	0.91
ConvolutionNet-5M0	0.90	0.89	0.88
ConvolutionNet-6	0.90	0.88	0.88

Fig. 33. Ablation Study Summary Table 2

C. *Comparison between our custom model and the pre-trained models*

Using precision, accuracy, recall, and f1-score, we evaluate our proposed CNN model and the pre-trained model named VGG16, InceptionV3, DenseNet169, and MobileNetV2. The results are shown in the figure 34.

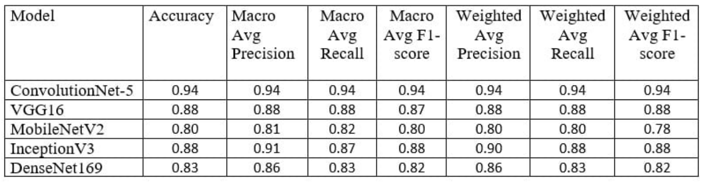


Class	ConvolutionNet-5 (Precision)	ConvolutionNet-5 (Recall)	ConvolutionNet-5 (F1-Score)	VGG16 (Precision)	VGG16 (Recall)	VGG16 (F1-Score)	MobileNetV2 (Precision)	MobileNetV2 (Recall)	MobileNetV2 (F1-Score)	InceptionV3 (Precision)	InceptionV3 (Recall)	InceptionV3 (F1-Score)	DenseNet169 (Precision)	DenseNet169 (Recall)	DenseNet169 (F1-Score)
Anthracnose_Leaf	0.95	1.00	0.97	0.91	0.86	0.88	0.84	0.89	0.86	0.88	0.97	0.92	0.87	0.94	0.91
Bacterial_Canker_Leaf	1.00	1.00	1.00	0.90	0.90	0.90	1.00	1.00	1.00	1.00	1.00	1.00	0.91	1.00	0.95
Die_Back_Leaf	1.00	1.00	1.00	0.93	0.93	0.93	0.91	1.00	0.95	1.00	1.00	1.00	0.95	1.00	0.98
Gall_midge_Leaf	0.85	0.89	0.87	0.64	0.84	0.73	0.50	0.89	0.64	0.58	0.74	0.65	0.48	0.74	0.58
Leaf_Cutting_Weevil_Leaf	0.92	0.86	0.89	0.93	0.93	0.93	1.00	0.93	0.96	1.00	0.79	0.88	1.00	0.79	0.88
Normal_Leaf	0.93	0.96	0.94	0.93	1.00	0.95	0.91	0.77	0.83	0.93	1.00	0.96	0.86	0.96	0.91
Powdery_Mildew_Leaf	0.89	0.94	0.92	0.94	0.89	0.91	0.85	0.94	0.89	0.93	0.72	0.81	0.85	0.94	0.89
Red_Rust_Leaf	0.95	0.90	0.93	0.76	0.62	0.68	0.50	0.14	0.22	1.00	0.62	0.76	0.86	0.29	0.43
Shoot_mold_Leaf	1.00	0.86	0.93	0.95	0.86	0.90	0.81	0.77	0.79	0.85	1.00	0.92	0.95	0.82	0.88

Fig. 34. Classification Report of our custom CNN model

This figure 35 summarizes the performance metrics including accuracy, macro-average, and weighted-average precision, recall, and F1-score for ConvolutionNet-5, VGG16, MobileNetV2, InceptionV3, and DenseNet169 models

- Macro-average treats all classes equally, regardless of their distribution in the dataset.
- Weighted-average takes into account the imbalance in the dataset by weighting each class's contribution by its support.



Model	Accuracy	Macro Avg Precision	Macro Avg Recall	Macro Avg F1-score	Weighted Avg Precision	Weighted Avg Recall	Weighted Avg F1-score
ConvolutionNet-5	0.94	0.94	0.94	0.94	0.94	0.94	0.94
VGG16	0.88	0.88	0.88	0.87	0.88	0.88	0.88
MobileNetV2	0.80	0.81	0.82	0.80	0.80	0.80	0.78
InceptionV3	0.88	0.91	0.87	0.88	0.90	0.88	0.88
DenseNet169	0.83	0.86	0.83	0.82	0.86	0.83	0.82

Fig. 35. Classification Report of our custom CNN model

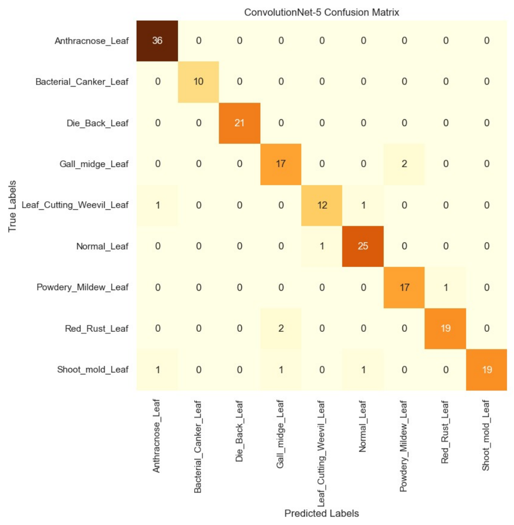
Based on this Figure 35,

- ConvolutionNet-5 shows the highest performance across all metrics.

- VGG16 and InceptionV3 also perform well, with similar macro and weighted average scores.
- MobileNetV2 and DenseNet169 have slightly lower overall performance compared to the other models.

D. *Class-wise study for the confusion matrix*

In our research, we categorize the confusion matrix outcomes into two main groups: True and False values. Let's delve into understanding the confusion matrices below.
ConvolutionNet-5:



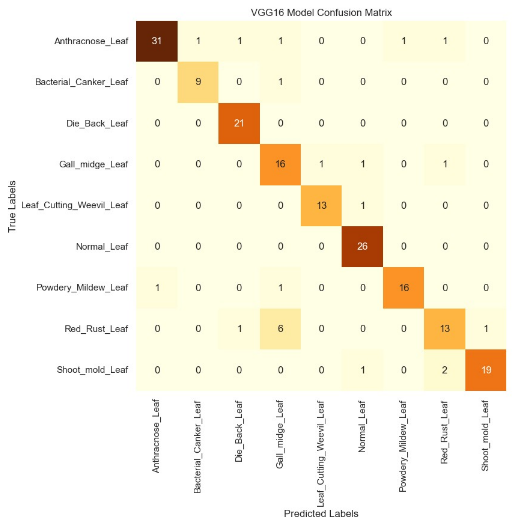
	Anthracnose_Leaf	Bacterial_Canker_Leaf	Die_Back_Leaf	Gall_midge_Leaf	Leaf_Cutting_Weevil_Leaf	Normal_Leaf	Powdery_Mildew_Leaf	Red_Rust_Leaf	Shoot_mold_Leaf
Anthracnose_Leaf	36	0	0	0	0	0	0	0	0
Bacterial_Canker_Leaf	0	10	0	0	0	0	0	0	0
Die_Back_Leaf	0	0	21	0	0	0	0	0	0
Gall_midge_Leaf	0	0	0	17	0	0	2	0	0
Leaf_Cutting_Weevil_Leaf	1	0	0	0	12	1	0	0	0
Normal_Leaf	0	0	0	0	1	25	0	0	0
Powdery_Mildew_Leaf	0	0	0	0	0	0	17	1	0
Red_Rust_Leaf	0	0	0	2	0	0	0	19	0
Shoot_mold_Leaf	1	0	0	1	0	1	0	0	19

Fig. 36. Confusion Matrix of ConvolutionNet-5

Let's explain the values in the confusion matrix. Vertically, we have 'Actual Class', and horizontally, 'Predicted Class'. in this confusion matrix (Figure 36), all the diagonal values are True predictions and others are False predictions of the model.

- True predictions: 36,10,21,17,12,25,17,19,19(Diagonal Dark light boxes)
- False predictions: 2,1,1,1,1,2,1,1,1(Row-wise-All the bottle white boxes)

VGG16:



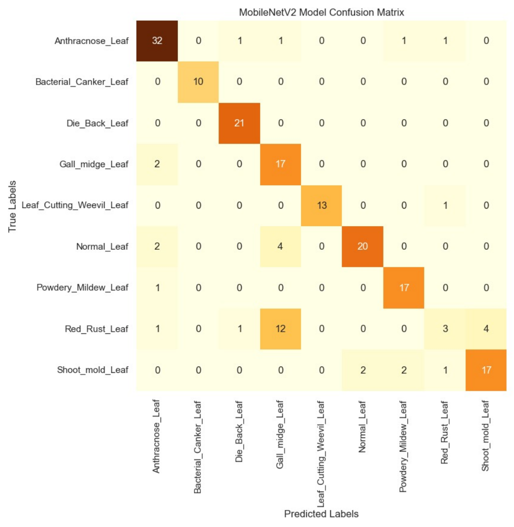
	Anthracnose_Leaf	Bacterial_Canker_Leaf	Die_Back_Leaf	Gall_midge_Leaf	Leaf_Cutting_Weevil_Leaf	Normal_Leaf	Powdery_Mildew_Leaf	Red_Rust_Leaf	Shoot_mold_Leaf
Anthracnose_Leaf	31	1	1	1	0	0	1	1	0
Bacterial_Canker_Leaf	0	9	0	1	0	0	0	0	0
Die_Back_Leaf	0	0	21	0	0	0	0	0	0
Gall_midge_Leaf	0	0	0	16	1	1	0	1	0
Leaf_Cutting_Weevil_Leaf	0	0	0	0	13	1	0	0	0
Normal_Leaf	0	0	0	0	0	26	0	0	0
Powdery_Mildew_Leaf	1	0	0	1	0	0	16	0	0
Red_Rust_Leaf	0	0	1	6	0	0	0	13	1
Shoot_mold_Leaf	0	0	0	0	0	1	0	2	19

Fig. 37. Confusion Matrix of VGG16

Likewise, in this confusion matrix (Figure 37), all the diagonal values are True predictions and others are False predictions of the model.

- True predictions: 36,9,21,16,13,26,16,13,19(Diagonal Dark light boxes)
- False predictions: 1,1,1,1,1,1,1,1,1,1,1,1,1,1,6,1,1,2 (Row-wise-All the bottle white boxes)

MobileNetV2:



	Anthracnose_Leaf	Bacterial_Canker_Leaf	Die_Back_Leaf	Gall_midge_Leaf	Leaf_Cutting_Weevil_Leaf	Normal_Leaf	Powdery_Mildew_Leaf	Red_Rust_Leaf	Shoot_mold_Leaf
Anthracnose_Leaf	32	0	1	1	0	0	1	1	0
Bacterial_Canker_Leaf	0	10	0	0	0	0	0	0	0
Die_Back_Leaf	0	0	21	0	0	0	0	0	0
Gall_midge_Leaf	2	0	0	17	0	0	0	0	0
Leaf_Cutting_Weevil_Leaf	0	0	0	0	13	0	0	1	0
Normal_Leaf	2	0	0	4	0	20	0	0	0
Powdery_Mildew_Leaf	1	0	0	0	0	0	17	0	0
Red_Rust_Leaf	1	0	1	12	0	0	0	3	4
Shoot_mold_Leaf	0	0	0	0	0	2	2	1	17

Fig. 38. Confusion Matrix of MobileNetV2

Similarly, in this confusion matrix (Figure 38), all the diagonal values are True predictions and others are False predictions of the model.

- True predictions: 32,10,21,17,13,20,17,3,17(Diagonal Dark light boxes)
- False predictions: 1,1,1,1,2,1,2,4,1,1,1,12,4,2,2,1 (Row-wise-All the bottle white boxes)

InceptionV3

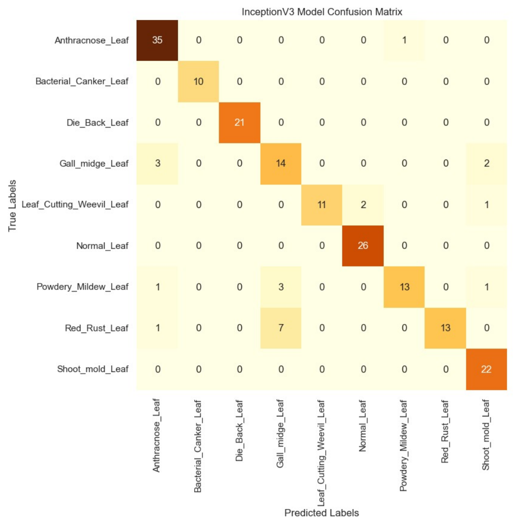


Fig. 39. Confusion Matrix of InceptionV3

Moreover, in this confusion matrix (Figure 39), all the diagonal values are True predictions and others are False predictions of the model.

- True predictions: 36,10,21,14,11,26,13,13,22(Diagonal Dark light boxes)
- False predictions: 1,3,2,2,1,1,3,1,1,7 (Row-wise-All the bottle white boxes)

DenseNet169

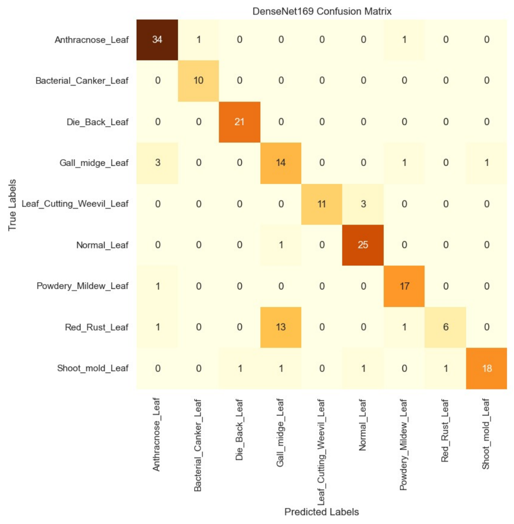


Fig. 40. Confusion Matrix of DenseNet169

However, in this confusion matrix (Figure 40), all the diagonal values are True predictions and others are False predictions of the model.

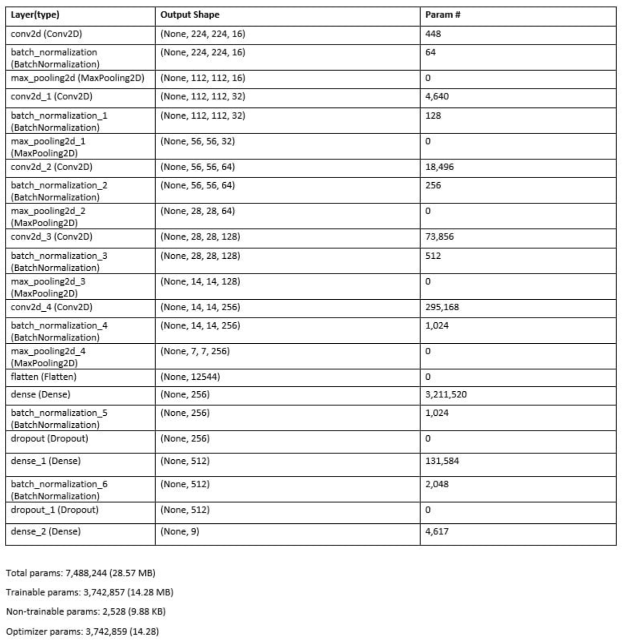
- True predictions: 34,10,21,14,11,25,17,6,18(Diagonal Dark light boxes)
- False predictions: 1,1,3,1,1,3,1,1,1,13,1,1,1,1,1 (Row-wise-All the bottle white boxes)

E. Output

1) Analysing Our Design Model (ConvolutionNet-5) and the Pre-Trained Model

Pre-trained models: In this research paper, we utilized VGG16, InceptionV3, DenseNet169, and MobileNetV2 as pre-trained models. Compared to our model (**ConvolutionNet-5**), these pre-trained models have lower accuracy. However, their Precision, Recall, F1 score values are decent. The downside is their high computational cost due to the large number of neurons and dense layers.

Our Proposed CNN model: We created a custom model for our research purpose. This model is based on the CNN-architecture and is lightweight and the computational cost is also low. Because it has fewer neurons and a smaller number of dense layers. This model has higher accuracy rate than pre-trained models. It indicates that our model's performance is very good. So, the model is very preferable. Figure 41 shows the summary of our proposed model ConvolutionNet-5.



Layer(type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 16)	448
batch_normalization (BatchNormalization)	(None, 224, 224, 16)	64
max_pooling2d (MaxPooling2D)	(None, 112, 112, 16)	0
conv2d_1 (Conv2D)	(None, 112, 112, 32)	4,640
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 32)	0
conv2d_2 (Conv2D)	(None, 56, 56, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 64)	0
conv2d_3 (Conv2D)	(None, 28, 28, 128)	73,856
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_4 (Conv2D)	(None, 14, 14, 256)	295,168
batch_normalization_4 (BatchNormalization)	(None, 14, 14, 256)	1,024
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 256)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 256)	3,211,520
batch_normalization_5 (BatchNormalization)	(None, 256)	1,024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131,584
batch_normalization_6 (BatchNormalization)	(None, 512)	2,048
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 9)	4,617

Total params: 7,488,244 (28.57 MB)

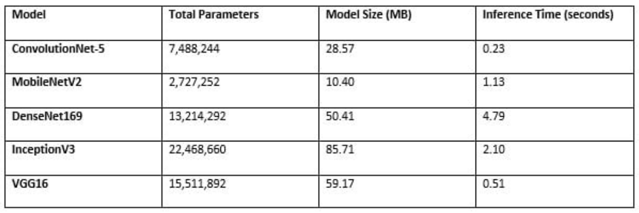
Trainable params: 3,742,857 (14.28 MB)

Non-trainable params: 2,528 (9.88 KB)

Optimizer params: 3,742,859 (14.28)

Fig. 41. The Summary Of Our Proposed Model ConvolutionNet-5

2) *Key Metrics Comparison for five different neural network models*



Model	Total Parameters	Model Size (MB)	Inference Time (seconds)
ConvolutionNet-5	7,488,244	28.57	0.23
MobileNetV2	2,727,252	10.40	1.13
DenseNet169	13,214,292	50.41	4.79
InceptionV3	22,468,660	85.71	2.10
VGG16	15,511,892	59.17	0.51

Fig. 42. Key Metrics Comparison of five different models

The comparison Figure 42 highlights the key metrics for five different neural network models. ConvolutionNet-5, MobileNetV2, DenseNet169, InceptionV3, and VGG16. ConvolutionNet-5 has 7,488,244 parameters, a model size of 28.57 MB, and an inference time of 0.23 seconds, making it moderately sized with a fast inference time and superior accuracy compared to the other models. MobileNetV2, with 2,727,252 parameters and a model size of 10.40 MB, offers the smallest storage footprint but has a relatively slower inference time of 1.13 seconds. DenseNet169, having 13,214,292 parameters and a model size of 50.41 MB, results in the longest inference time of 4.79 seconds due to its large parameter count and size. InceptionV3 stands out with the highest number of parameters (22,468,660) and the largest model size (85.71 MB), yet it has a faster inference time of 2.10 seconds compared to DenseNet169. VGG16, with 15,511,892 parameters and a model size of 59.17 MB, balances a significant parameter count and size with a relatively quick inference time of 0.51 seconds.

3) *Visual Representation and Analysis of the Result Implementation:*

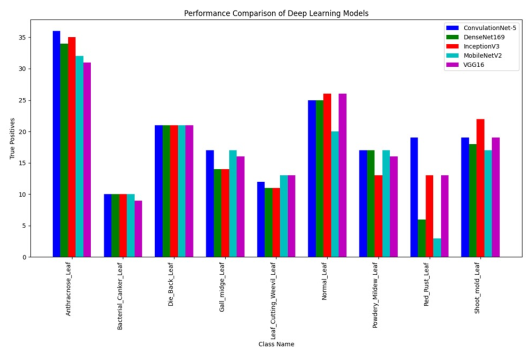
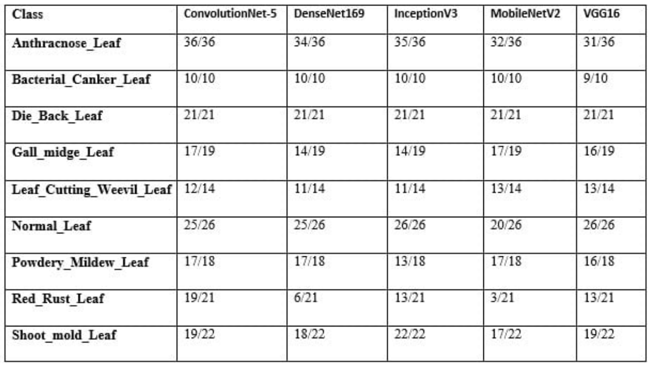


Fig. 43. Performance Comparison of Deep Learning Models



Class	ConvolutionNet-5	DenseNet169	InceptionV3	MobileNetV2	VGG16
Anthracnose_Leaf	36/36	34/36	35/36	32/36	31/36
Bacterial_Canker_Leaf	10/10	10/10	10/10	10/10	9/10
Die_Back_Leaf	21/21	21/21	21/21	21/21	21/21
Gall_midge_Leaf	17/19	14/19	14/19	17/19	16/19
Leaf_Cutting_Weevil_Leaf	12/14	11/14	11/14	13/14	13/14
Normal_Leaf	25/26	25/26	26/26	20/26	26/26
Powdery_Mildew_Leaf	17/18	17/18	13/18	17/18	16/18
Red_Rust_Leaf	19/21	6/21	13/21	3/21	13/21
Shoot_mold_Leaf	19/22	18/22	22/22	17/22	19/22

Fig. 44. Number of True Predition of Deep Learning Models

The Figures 43 and 44 provides a comparative analysis of the performance of five deep learning models (ConvulationNet-5, DenseNet169, InceptionV3, MobileNetV2, VGG16) across nine different leaf categories used in classification tasks. Here's a description of the table:

1. Performance Consistency Across Models:

- Anthracnose_Leaf: All models perform reasonably well, with ConvulationNet-5, DenseNet169, and InceptionV3 achieving very high accuracy (around 94-97%), while MobileNetV2 and VGG16 are slightly lower (around 86-89%).
- Bacterial_Canker_Leaf: All models achieve perfect accuracy (100%) except for VGG16, which is slightly lower (90%).
- Die_Back_Leaf: Consistent perfect accuracy (100%) across all models.
- Gall_midge_Leaf: ConvulationNet-5 and MobileNetV2 perform better (89-94%), while DenseNet169, InceptionV3, and VGG16 are slightly lower (74-84%).
- Leaf_Cutting_Weevil_Leaf: ConvulationNet-5 performs the best (86%), followed closely by DenseNet169, InceptionV3, MobileNetV2, and VGG16 (79-93%).
- Normal_Leaf: All models achieve near-perfect accuracy (96-100%).
- Powdery_Mildew_Leaf: InceptionV3 performs the best (72%), followed by ConvulationNet-5, DenseNet169, MobileNetV2, and VGG16 (61-94%).
- Red_Rust_Leaf: ConvulationNet-5 and InceptionV3 achieve similar performance (90-100%), while DenseNet169, MobileNetV2, and VGG16 are notably lower (14-62%).
- Shoot_mold_Leaf: InceptionV3 achieves the highest accuracy (100%), followed by DenseNet169, VGG16, ConvulationNet-5, and MobileNetV2 (77-86%).

2. Model-specific Observations:

- ConvulationNet-5: Generally, shows competitive performance across most leaf categories, often achieving high accuracy.

- DenseNet169: Performs consistently well, especially in categories with higher sample sizes, but shows variability in categories with fewer instances.
- InceptionV3: Often excels in categories where others struggle, demonstrating robustness in varied leaf conditions.
- MobileNetV2: Shows a mixed performance, sometimes matching the top performers but occasionally falling short in certain categories.
- VGG16: Performs adequately but tends to lag behind in categories requiring finer distinctions or with smaller sample sizes.

VI. Conclusion

So, in this paper, we developed Convolutional Neural Network based models for the task of mango leaf disease detection. Understanding the critical compliance requirement to detect diseases as quickly and accurately as possible to ensure the best quality and quantity of mangoes we have trained a few CNN models especially for this task. The results are pretty close to multiple famous pre-trained models such as VGG16, InceptionV3, Densenet169, and MobileNetV2. Finally, our findings have shown that the custom CNN models could outperform the pre-trained models in some aspects and also showed significant potential for practical application in real-world agricultural settings because of their better-optimized architecture and parameters when utilized for identifying mango leaf disease.

References

[1] U. P. Singh, S. S. Chouhan, S. Jain, and S. Jain, "Multilayer convolution neural network for the classification of mango leaves infected by anthracnose disease," *IEEE Access*, vol. 7, pp. 43 721–43 729, 2019.

[2] S. Arivazhagan and S. V. Ligi, "Mango leaf diseases identification using convolutional neural network," *International Journal of Pure and Applied Mathematics*, vol. 120, no. 6, pp. 11 067–11 079, 2018.

[3] K. K. Patel, A. Kar, and M. Khan, "Common external defect detection of mangoes using color computer vision," *Journal of the Institution of Engineers (India): Series A*, vol. 100, pp. 559–568, 2019.

[4] A. Rajbongshi, T. Khan, M. M. R. A. Pramanik, S. M. Tanvir, and N. R. C. Siddiquee, "Recognition of mango leaf disease using convolutional neural network models: a transfer learning approach," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 23, no. 3, pp. 1681–1688, 2021.

[5] M. Rahaman, M. Chowdhury, and e. a. M. A. Rahman, "A deep learning based smartphone application for detecting mango diseases and pesticide suggestions," *International Journal of Computing and Digital Systems*, vol. 13, no. 1, pp. 1–1, 2023.

[6] S. Veling, "Mango disease detection by using image processing," *International Journal for Research in Applied Science and Engineering Technology*, vol. 7, pp. 3717–3726, 04 2019.

[7] C. Trongtorkid and P. Pramokchon, "Expert system for diagnosis mango diseases using leaf symptoms analysis," in *2018 international conference on digital arts, media and technology (ICDAMT)*. IEEE, 2018, pp. 59–64.

[8] S. C. Nelson, "Mango powdery mildew," 2008.

[9] S. Nelson, "Mango powdery mildew caused by oidium mangiferae," Sep 2013, retrieved from: https://www.flickr.com/photos/scotnelson/9808512885.

[10] S. I. Ahmed, M. Ibrahim, M. Nadim, M. M. Rahman, M. M. Shejunti, T. Jabid, and M. S. Ali, "Mangoleafbd: A comprehensive image dataset to classify diseased and healthy mango leaves," *Data in Brief*, vol. 47, p. 108941, 2023.

[11] G. Kaur, N. Sharma, S. Malhotra, S. Devliyal, and R. Gupta, "Mango leaf disease detection using vgg16 convolutional neural network model," in *2024 3rd International Conference for Innovation in Technology (INOCON)*. IEEE, 2024, pp. 1–6.

[12] A. K. Ratha, N. K. Barpanda, P. K. Sethy, and S. K. Behera, "Automated classification of indian mango varieties using machine learning and mobilenet-v2 deep features." *Traitement du Signal*, vol. 41, no. 2, 2024.

[13] S. Mahato and A. Sharma, "Tomato leaf disease detection using inception v3," *NEU Journal for Artificial Intelligence and Internet of Things*, vol. 2, no. 2, pp. 1–7, 2023.

[14] R. Tri Wahyuningrum, A. Kusumaningsih, W. Putra Rajeb, and I. K. Eddy Purnama, "Classification of corn leaf disease using the optimized densenet-169 model," in *Proceedings of the 2021 9th International Conference on Information Technology: IoT and Smart City*, 2021, pp. 67–73.