

Mango leaf disease detection using image processing

by

Avizit Sarkar

19201113

Murshed Hasan

23141066

Nirnoy Chandra Sarker

24141132

Moin Nadim Srabon

20101140

Safwat Sufia

24141297

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
Spring 2024

© 2024. Brac University
All rights reserved.

Declaration

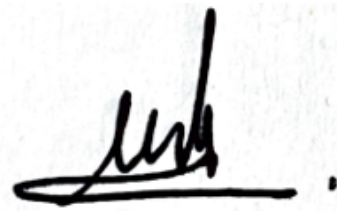
It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

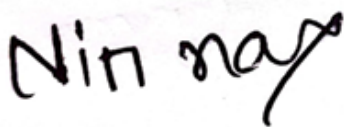
Student's Full Name & Signature:



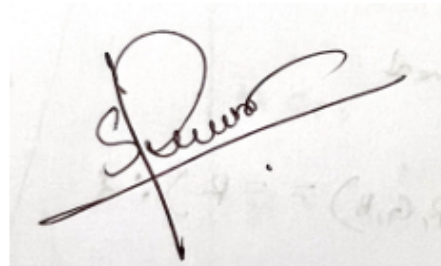
Avizit Sarkar
19201113




Murshed Hasan
23141066



Nirnoy Chandra Sarker
24141132



Moin Nadim Srabon
20101140



Safwat Sufia
19101293

Approval

The thesis titled “Mango leaf disease detection using image processing” submitted by

1. Avizit Sarkar(19201113)
2. Murshed Hasan(23141066)
3. Nirnoy Chandra Sarker(24141132)
4. Moin Nadim Srabon(20101140)
5. Safwat Sufia(24141297)

of Spring, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on July 05, 2024.

Examining Committee:

Supervisor: (Member)



Dr. Md. Ashraful Alam
Associate Professor
Department of Computer Science and Engineering
Brac University

Co-Supervisor: (Member)



Md Tanzim Reza
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator: (Member)

Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department: (Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

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.

Keywords: Convolutional Neural Networks (CNN), Mango Leaf, Disease Detection, Deep Learning, Bangladesh, MobileNetV2, VGG16, DenseNet169 and InceptionV3.

Acknowledgement

We would like to acknowledge and give our warmest thanks to our Supervisor, Dr. Md. Ashraful Alam, and Co-supervisor, Md. Tanzim Reza, whose guidance and advice were crucial throughout all stages of my thesis. Their expertise and support have been invaluable in shaping this work. We also wish to express our heartfelt gratitude to Horticulturist Sujit Mandal for his assistance in the creation of the dataset and for doing the verification of it. His contributions significantly enhanced the quality of this research.

Table of Contents

Declaration	i
Approval	ii
Abstract	iv
Acknowledgement	v
Table of Contents	vi
Nomenclature	vii
1 Introduction	1
1.1 Motivation	1
1.2 Research Problem	2
1.3 Research Objective	2
2 Literature Review	5
3 Methodology	11
3.1 Proposed Methodology	11
3.2 Dataset	13
3.3 Data Collection Methodology	13
3.3.1 Data Collection	13
3.4 Data Pre-processing	13
3.4.1 Dataset Visualization	13
3.4.2 Data Augmentation	14
3.4.3 Pre-processed Data	16
3.5 Model Specification	17
3.5.1 Convolutional Neural Network(CNN)	17
3.5.2 MobileNet V2	20
3.5.3 VGG 16	21
3.5.4 DenseNet169	22
3.5.5 InceptionV3	23
4 Implementation	25
4.1 Workflow	25
4.2 Setup for Experiment	26
4.2.1 Training hardware and Software	26
4.2.2 Library List	27

4.2.3	Structural view of the code skeleton	27
4.3	Model Selection	27
4.4	Hyperparameter Tuning	28
4.5	Design and Compile the Models	29
4.5.1	ConvolutionNet-5	29
4.5.2	ConvolutionNet-5M0	29
4.5.3	ConvolutionNet-5M1	30
4.5.4	ConvolutionNet-4	30
4.5.5	ConvolutionNet-4M1	30
4.5.6	ConvolutionNet-6	31
4.5.7	ConvolutionNet-3	31
4.5.8	VGG16	32
4.5.9	InceptionV3	32
4.5.10	MobileNetV2	33
4.5.11	DenseNet169	33
5	Result Analysis	34
5.1	Train and evaluate the Models	34
5.1.1	ConvolutionNet-5	34
5.1.2	ConvolutionNet-3	35
5.1.3	ConvolutionNet-4	36
5.1.4	ConvolutionNet-4M1	36
5.1.5	ConvolutionNet-5M0	38
5.1.6	ConvolutionNet-5M1	39
5.1.7	ConvolutionNet-6	40
5.1.8	VGG16	41
5.1.9	MobileNetV2	42
5.1.10	InceptionV3	44
5.1.11	DenseNet169	44
5.2	Ablation Study of Custom Convolutional Neural Networks for Mango Leaf Disease Classification	45
5.2.1	Model Architectures and Performance Metrics	46
5.2.2	Analysis	47
5.3	Comparison between our custom model and the pre-trained models .	49
5.4	Class-wise study for the confusion matrix	50
5.5	Output	56
5.5.1	Analysing Our Design Model (ConvolutionNet-5) and the Pre- Trained Model:	56
5.5.2	Key Metrics Comparison for five different neural network models	59
5.6	Visual Representation and Analysis of the Result Implementation: .	60
6	Conclusion	62
6.1	Future Work	62
	Bibliography	66

Chapter 1

Introduction

1.1 Motivation

In Bangladesh, the agricultural sector plays a significant role and mango cultivation is one of the great contributors. However, the quality and production of mangoes can be greatly hampered by a number of dangerous leaf diseases such as root rot, powdery mildew, bacterial black spot, and anthracnose. These diseases not only degrade the trees' visual appeal but also damage their capacity for effective photosynthetic processes, which in turn decreases fruit production and quality [9]. Hence, early detection of these diseases is very much necessary for farmers to ensure the healthiness and supply of mangoes according to the demand.

The condition of the environment in Bangladesh, such as the climate, soil composition, and agricultural methods creates further difficulties. 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. These techniques further complicate the process of detecting quickly and accurately, especially for farmers who have limited knowledge and limited access to agricultural specialists.

Deep learning is one of the most powerful tools for picture recognition and classification in recent years, which is a subsection of machine learning. Convolutional neural networks, or CNNs, have shown magnificent results in a large number of applications, such as pattern recognition, picture segmentation, and object detection [8]. The drawbacks of conventional methods for detecting mango leaf disease can easily be recognized by this technology.

The need for a workable environment, and an effective, smooth, and easily accessible method for detecting mango tree leaf disease in Bangladesh is the main focus of our project. 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 that are unique to the area. Our creative strategy could completely change the mango production in Bangladesh. By using our easy interface tool farmers will be able to detect the mango leaf disease at an early stage. At last, we can hope that our custom CNN model can completely contribute to the enhancement of our economy and ensure

the food security as well.

1.2 Research Problem

The better production of mangoes in Bangladesh depends on the early detection of affected mango leaves. Besides, there are a number of limitations and lacking with the conventional methods of mango leaf disease detection that are used now [8]. However, manual detection of mango leaf diseases is a time-consuming and laborious process that highly requires a deep understanding of affected leaf symptoms and their various manifestations. But 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.

Bangladesh's particular environmental circumstances made it more difficult for the early detection of diseases. Accurately diagnosing the precise issue affecting the mango trees can be challenging due to the influence of various factors such as soil type, climate, and agricultural techniques. Poor productivity and lower fruit quality can arise from improper care following a delayed or inaccurate disease detection.

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. Current technologies most frequently depend on manually created characteristics, which cannot translate well to newly acquired diverse datasets. Furthermore, previous research may focus on other crops or other environmental production that would have limited its application to Bangladesh's particular mango fruit production scenario.

Furthermore, diseases can easily be treated when they are detected in their early stages, a time when traditional methods frequently fall short. These diseases may already have seriously harmed the crop by the time symptoms become noticeable. As a result of increasing treatment costs and decreasing effective detection mango cultivation's economic sustainability is hampered.

Thus, this thesis looks 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, our thesis involves creating a comprehensive dataset that captures around 13200 images along with both healthy and affected leaves using cutting-edge CNN architectures to accomplish high accuracy classification.

1.3 Research Objective

Our research's main goal is to create a cutting-edge, deep learning-based architecture that can accurately identify and classify fresh mango leaves and affected leaves. The custom CNN model will be specially designed to fit Bangladesh's particular climatic circumstances and disease verities. The goal of this project is to make the farmers

use modern technology in order to make mango production more efficient and risk-free. To do this, these specific objectives need to be established:

- **Develop Disease Detection and Management:** Our main focus is to give farmers a precise and user-friendly interface technology for disease detection systems by creating an innovative deep-learning architecture. By doing so, the farmers will be able to detect several leaf diseases at an early stage that impact mango production. This will facilitate efficient management techniques and reduce the possibility of fruit production and quality losses and farmers can reduce their economic loss as well.
- **Enhance Agricultural Sustainability:** Through our custom CNN model, the long-term viability of mango farming in Bangladesh will be enhanced. We are hoping to improve the productivity and ensure better quality of mangoes by tackling the problems caused by leaf diseases. This would guarantee that farmers can reduce the loss and can make a steady supply of premium mangoes for both home and foreign markets.
- **Address Regional Specificities:** The aim is to observe the distinct climatic circumstances and disease varieties that are particular to Bangladesh. We intend to create a system that is suited enough to fulfill the requirements of local farmers, ensuring its easy applicability and efficacy in their particular setting, by gathering a huge dataset of both fresh and affected leaves that reflects various areas, seasons, and mango tree species.
- **Boost Usability and Accessibility:** We want to create a straightforward and easy-to-use interface that works for farmers of all technological backgrounds. Our goal is to develop a user-friendly tool that will make disease identification as simple as uploading leaf photographs and receiving real-time updates. As a result, farmers will be able to take proactive steps and raise yields.
- **Educate and Raise Awareness:** We mainly focus on educating farmers, other relevant stakeholders, and the agricultural community at large about the benefits of deep learning-based disease detection in addition to making the system. We believe that educating farmers about innovations can lead to higher yields, improved farming methods, and wider adoption. If they become enough educated about using deep learning-based CNN architecture then a huge improvement will come not only in disease detection but also in making economic profit.
- **Promote Economic Growth:** A big portion of Bangladesh's economy is heavily dependent on the cultivation of mangoes. By overcoming the problems caused by leaf diseases, our project is hoping to boost the nation's economy and ensure food security as well. Mango production and quality growth have the potential to improve export earnings and the agricultural sector, which would benefit farmers' livelihoods and the development of the country as a whole.

When combined, these described objectives show the importance and benefits that this study is seeking to achieve. Our real objective is to develop a deep learning-based system for identifying mango leaf disease in order to revolutionize the way

mangoes are farmed in Bangladesh and this will create a new movement about mango leaf disease detection which we did not observe in the past. This will ensure more strong mango cultivation profit, more harvests, and sustainable development.

Chapter 2

Literature Review

In order to accurately distinguish between mango leaves that are infested with the devastating fungal disease anthracnose and those that are not, Singh et al. [21] developed two datasets containing 2200 pictures. The datasets included leaves from other plants that were infected with the same disease in addition to a mixed input of mango leaves that were diseased and uninfected. utilising Multilayer Convolutional Neural Network (MCNN) for classification, they got 97.16% accuracy instead of utilising traditional approaches like PSO, SVM, or RBFNN.

With the goal of identifying and classifying mango diseases that are hard to diagnose with the naked eye, Srunitha et al. [13] have suggested a new strategy. In this technique, diseases are classified using multiclass SVM and segmented data using k-means clustering with an accuracy of 96%.

For mango leaf disease identification, Mia et al. [24] suggested the neural network and supported vector machine methods. The dataset contained four different forms of diseased mango leaves such as, (Dag disease, Golmachi disease, Moiricha disease, and Shutimold disease). A support Vector Machine (SVM), was used in the study dataset containing both diseased and non-diseased mango leaves. Finally, the test resulted in 80% accuracy.

Rahaman et al.[35] proposed a smartphone app that could not only identify mango diseases by deep learning but also suggest appropriate pesticides to mitigate them. Machine learning techniques like DenseNet 169, InceptionV3, and MobileNetV2 were used by the app to detect mango diseases depending on pictures of infected as well as healthy leaves and fruits. 97.81% accuracy was proven by this method for disease detection.

Prabu et al. [31] has succeeded in diagnosing mango leaf diseases by using convolutional neural networks. The dataset consists of five mango diseases including Leaf burn of mango, Leaf Webber, Leaf Gall, Alternaria leaf spots and Anthracnose. They used the CNN model for identification also classification purposes and achieved 96.67% accuracy.

Iqbal et al. [9] have introduced different methods for the identification and classification of citrus leaves' diseases. The main concepts of the review paper were

methodologies, merits, demerits and obstacles while deep learning image processing techniques to guarantee accurate disease detection.

Malao et al. [19] introduced a computer-based programme that utilises user responses for disease diagnosis. The system contains three primary parts which include an image analyzer, feature extraction and a classifier. An orderly procedure was maintained in the system, starting from RGB image acquisition, conversion to HSV colour space, masking and removal of green pixels, segmentation of the infected region, extraction of useful segments, and ended with computation of texture features using colour co-occurrence methodology.

In order to prevent a decrease in mango production, Veling and colleagues [22] presented a method that employed the Gray Level Co-occurrence Matrix (GLCM) in addition to SVM. The GLCM captured the vital texture features for disease identification and the SVM classifier was used for disease classification. The overall procedure has shown 90% accuracy in identifying diseases.

Arivazhagan et al.[7] have suggested a CNN model that has achieved 96.67% accuracy in identifying mango leaf diseases. This study uses technology based on deep learning to automatically identify leaf diseases in many varieties of mango plants. They used five different leaf diseases, such as Anthracnose, Alternaria leaf spots, Leaf Gall, Leaf Webber, and Leaf Burn, included in a dataset of 1200 images of healthy and diseased mango leaves. Extraction and classification procedures were executed using the Convolutional Neural Network (CNN) with 8 different layers. Different poses, pixels etc were also taken into account in the image processing system as they play a vital part in CNN. The proposed CNN model was trained using 100 images per class which is a total of 600 training photos. Each of the 600 photos was tested and 96.67% accuracy was achieved.

With the goal of identifying leaf diseases in mango and grape crops, Sanath Rao and colleagues [28] have proposed a deep-learning methodology. Their research has used pre-trained neural networks including these steps of data collection, preprocessing, transfer learning, and evaluation which have shown greater precision in disease detection. This methodology is also very potent for practical application in farming.

With the aim of identifying surface irregularities in mangos, the use of colour computer vision methods was introduced by Patel, Kar, and Khan [20]. They employed a comparative analysis technique with existing ones, including image processing methods like segmentation, feature extraction and classification algorithms. An efficiency of 93.3% followed by an accuracy of 88.6% was achieved from this strategy.

In Pham et al.'s studies, [25] a feed-forward neural network and a hybrid metaheuristic feature selection algorithm were suggested where a feedforward neural network (FFNN) was used as the classification model and a hybrid metaheuristic feature selection algorithm that combines Particle Swarm Optimization (PSO) and Cuckoo Search (CS) was used to determine the targeted features. The model had achieved an impressive 89.41% accuracy which suppressed the other CNN models.

A competent model was suggested by Trongtorkid et al. [15] that can examine the leaf symptoms to identify mango diseases. In order to develop the system, CLIPS and MATLAB software were used, also, 100 images of both healthy and infected mango leaves were used to assess it. The system achieved 89.92% of accuracy which proved to be better than the existing methods.

Sutrodhor et al. [14] proposed a new strategy that combines Neural Network and Support Vector Machine (SVM) to efficiently identify mango leaf diseases. This methodology is not only an achievable solution for automated mango disease detection, but also a potent approach that has outperformed individual neural networks and SVM classifiers.

In order to diagnose the mango leaf disease anthracnose, Wongsila, Chantrasri and Sureephong [30] employed machine learning algorithms. For classification, various machine learning techniques were tested such as Support Vector Machines (SVM), K-nearest neighbours (KNN), Decision Trees and Random forests. Finally, the test has achieved 70% accuracy in detecting mango leaf diseases.

Gulavnai et al. [18] provide new deep-learning models to recognize mango leaf diseases and a cost-effective approach to image recognition. The study tackles the essential and initial need for early identification of diseases such as Powdery Mildew, Anthracnose, Red Rust, and Golmorich. Utilizing a dataset of 8,853 images across from the Konkan area of India, the researchers trained their pre-trained model with transfer learning and the ResNet architecture, reaching a remarkable 91% accuracy. Their approach required preprocessing the images and improving the data to improve their model accuracy. Different CNN architectures like ResNet18, ResNet34, and ResNet50, were used. Testing results indicate idea and actual accuracy across different structures, with the ResNet50 model beating the others. The research highlights the value of deep learning models in agricultural applications that provide an efficient and affordable solution for early disease detection in mango crops.

In order to highlight the automation of the detection of eight different mango leaf diseases, Vijay et al. [36] have proposed a hybrid deep learning strategy. In this research, the challenges faced in the initial stage of disease detection are mentioned which is very crucial for enhancing the yield and quality of crops. A total of 4873 images containing both diseased and healthy leaves are included in the dataset. Some of the diseases such as Anthracnose, Bacterial Canker, Cutting Weevil, Die Back, Gall, Midge, Powdery Mildew, Red Rust, and Sooty Mould were included. The methodology includes data analysis, preprocessing, image segmentation and finally data augmentation which is required to verify class imbalance. This study has mentioned 4 models that are- Custom CNN, VGG-16, EfficientNetB4 and a hybrid proposed model. 50 epochs are operated in order to evaluate the models and results have shown 84% accuracy by the Custom CNN model, 81.54% accuracy by the VGG-16 and finally an increased accuracy by 3.15% over the earlier models by the EfficientNetB4. On the other hand, 94.72% accuracy was observed in the proposed hybrid model which is the highest acquired accuracy rate with practical estimation. This technique has proven to be efficient for our farmers as it plays a vital role in saving both time and money by minimising the quantity of farming equipment.

In order to accurately identify and classify the diseases found in mango leaves, Rabia et al. [29] have proposed a model that can ensure increased production and return from agricultural land. In their study, they have suggested a CNN-based model (FrCNnet) for disease classification. Using the preprocessed data, the model directly trains pixel-level features and 99.2% accuracy was achieved. Also, their proposed model contains certain similarities with models like Vgg16, Vgg-19, and Unet. The method consists of developing a dataset containing images of diseased and healthy mango leaves, then preprocessing, data augmentation and many more. When traits, textures, and other features are taken into account, their suggested CNN model exhibits 98% accuracy, demonstrating its great potency for categorization. Therefore, in order to efficiently detect the diseases in mango leaves, this research has demonstrated the importance of automated models.

Sharma et al. [33] explain a Convolutional Neural Network (CNN)-based model for the quick detection and classification of mango leaf diseases. The research involves collecting pictures of mango leaves harmed with diseases such as anthracnose, red rust, and powdery mildew. One of the data pre-processing techniques is Data Augmentation that were used to improve the amount of data, and a CNN model was trained on this vital data. The suggested algorithms detected mango leaf diseases having an accuracy of 90.36%. The model modifications basically include different layers for feature extraction and pooling layers to reduce the sampling size. The accuracy of the model was determined by a number of factors, including the F1 Score and recall.

In the study conducted by Aditya et al. [27], transfer learning and convolutional neural networks (CNN) were used to construct an automated system for disease detection in mango leaves. The study inspected and compared different CNN architectures, including DenseNet201, InceptionResNetV2, InceptionV3, ResNet50, ResNet152V2, and Xception implementing measures like accuracy, F1 score, precision, and so on, false negative rate (FNR), and false positive rate (FPR). To solve the imbalance in the dataset they had done data segmentation into training and testing data. DenseNet201 had the best results, with an accuracy of 98.00%, an F1 score of 98.33%, and precision, sensitivity, and specificity of 99.58%. ResNet50 came in second with a 97.00% accuracy, while InceptionResNetV2 and InceptionV3 both achieved 96.67%. The results show that DenseNet201 and ResNet50 improve the accuracy.

The study by Sandhya S. et al. [32] aimed to identify diseases in mango leaves using deep learning specifically focusing on the pre-trained ResNet-50 model. Combining both healthy and diseased leaves of 435 images through scaling and Contrast Limited Adaptive Histogram Equalization (CLAHE) to enhance contrast. Data augmentation was used in order to address the problem of class imbalance. The features extracted by the fine-tuned ResNet-50 model were classified based on machine learning classifiers such as Support Vector Machine (SVM) and Logistic Regression, which caused an improvement in accuracies to 100% with those classifiers and 97.7% with the approach based on tuned parameters of the ResNet-50 model. The results emphasised the robustness of multiple approaches such as incorporating CNN features

with ML classifiers, using transfer learning schemes in combination with different model choices and additional perspective for accurate disease detection modules in mango leaves to support the industry with crop management assistance and prompt action recommendations.

This thesis, "Review on CNN applied to plant leaf disease classification," thoroughly addresses the value of Deep Learning (DL) in plant disease detection and identification, the classification method, problems and their solutions, and prospective future developments. Jiang [26] and the authors have been concerned about the detection error of the models that can occur due to insufficient data and lack of diversity. Transfer learning and data augmentation are some of the solutions they have suggested in order to mitigate it. The researchers have also mentioned that one of the reasons for the declining accuracy rate is insufficient robustness. Therefore, to accelerate the robustness, the authors have come up with some solutions such as Increasing dataset diversity, using a compressed model, utilising unsupervised Deep Learning(DL) methods and multi-condition training. Distinct types of CNN models such as AlexNet, GoogLeNet, and VGG were implemented by Ferentinos for the detection and classification of plant diseases with the public dataset. Lastly, 99.35% of high accuracy had been achieved. Therefore we can say that increased diversity of datasets and robustness plays a crucial role in achieving a higher accuracy rate, also for practical use, authors look forward to employing mobile and server-side applications.

The title of the thesis is "Plant Leaf Disease Detection Using Machine Learning" The Issue of plant disease in agriculture emphasizes the importance of early disease detection with high accuracy. The paper explores various machine learning techniques and methodologies such as K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Artificial Neural Networks (ANN) and Principal Component Analysis (PCA) that can be used for better disease detection. Shima R. et al [12] presented a method of image preprocessing, segmentation, and feature extraction using the GLCM (Gray Level Co-occurrence Matrix) algorithm and KNN classification and achieved Acc. The author's survey previously- highlighted its limitations and proposed some prospective modifications to the current methodology. The system they proposed could detect 7 types of plant leaf diseases with 98.56% accuracy, which demonstrates the potential of machine learning.

A Deep Convolutional Neural Network (CNN) model has been proposed by Geetharamani [17] and colleagues in order to successfully identify plant leaf diseases. A total of 54,305 images including 13 types of plant leaves were collected from plant villages for their dataset. Six data augmentation techniques that include image flipping, gamma correction, noise injection, PCA colour augmentation, rotation, and scaling have been utilised by the authors to achieve optimal results. Their dataset was divided into 38 classes which were labelled as either infected or non-infected leaves. The model has achieved a significant level of accuracy compared to other conventional machine learning models such as ResNet and SVM, hence proving the model's potency in identifying plant diseases for practical implementation.

Luna et al. (2018) [10] research smart farming systems aimed at improving tomato

output by using deep learning (DL) and computer vision techniques to find and detect disease in tomato plant leaves. The recommended technology includes an image collection box to picture tomatoes from multiple angles, which makes it simpler to recognize diseases including Diamante Max Breed, Phoma Rot, Leaf Miner, and Target Spot. A total of 4923 pictures containing four types of leaf samples (Healthy, Leaf Miner, Phoma Rot and Target Spot) were used in order to train the author's proposed models on the AlexNet architecture, which was specifically affected by transfer learning and reconnecting the weights of the final fully connected layer (fc8). A total of 91.6% accuracy was achieved with 36 sample tests proving its capability of initialising a smart farming system by employing deep learning and computer vision for maximum agricultural outputs.

The thesis "Attention Embedded Residual CNN for Disease Detection in Tomato Leaves" has mentioned the importance of deep learning methods for automated plant leaf disease detection. Here, the CNN model is suggested to overcome the limitations of conventional methods. Residual Learning Architecture and Attention Mechanism Architecture are two Deep learning Architectures that were used by the writers [23]. All of the tests were performed using a dataset from the plant village and a high accuracy of 98% was obtained which proves the efficiency of the 5-fold cross-validation strategy. The result was evaluated with the TensorFlow Deep Learning framework performed on an NVIDIA Tesla P100 GPU. The authors carried out three experiments in total, including a baseline model for detection and classification, integration of residual connections and finally incorporating attention and residual connection. Hence it is proven that advanced deep learning methods play an essential role in disease diagnosis of plants which can provide agricultural benefits.

Chapter 3

Methodology

3.1 Proposed Methodology

Figure 3.1 illustrates the overall system for detecting mango leaf diseases of several phases. Raw images that we collected from various sources were first passed through the pre-processing phase. Data augmentation was also done in the pre-processing phase. The pre-processed data set was then divided into a training set, a testing set, and a validation set. In phase 2, we used the training data to train the chosen models. The confusion matrix, precision, accuracy, recall, and F1-score were the evaluation measures used to gauge the performance of our models.

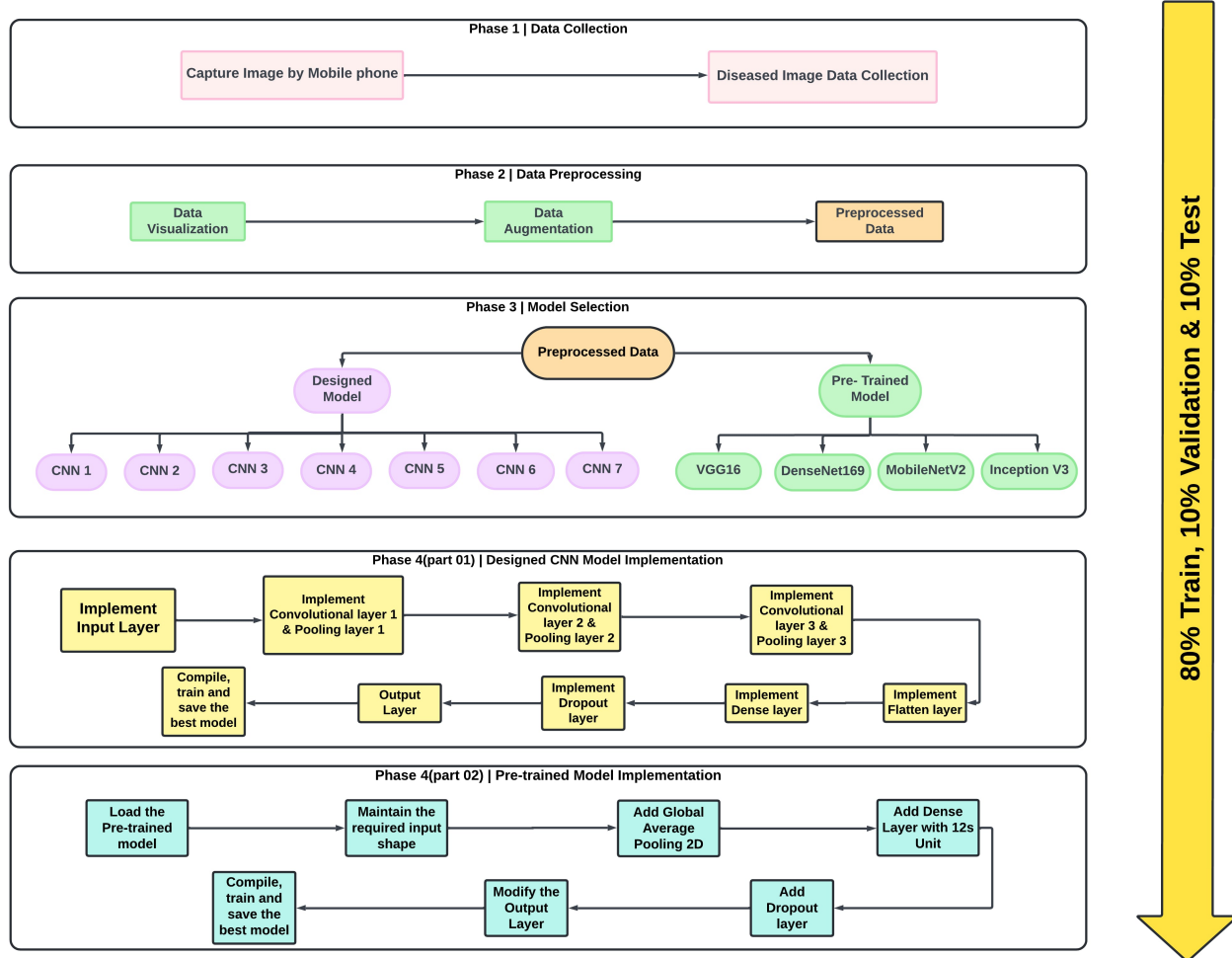


Figure 3.1: Top-level overview of the method

3.2 Dataset

Agriculture is a sector that has not yet fully benefited from the advancements in Deep Learning. The significance of datasets in Deep Learning is immense, and the absence of standardized, publicly available agricultural datasets limits the potential of these advanced computational techniques. To help bridge this gap, we have created what we believe to be the first comprehensive, ready-to-use dataset of mango leaves. This dataset includes 7,535 images of around 1,907 unique leaves, representing eight different diseases, collected from various mango orchards in Bangladesh, one of the world’s top mango producers. Although our dataset is based on mango leaves from Bangladesh, the diseases it covers are prevalent in many countries, making it applicable for global use and potentially enhancing mango production worldwide. We expect this dataset to garner significant interest from Deep Learning researchers and practitioners focusing on automated agriculture.

3.3 Data Collection Methodology

3.3.1 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. [1] [2] [34] 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

3.4 Data Pre-processing

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

3.4.1 Dataset Visualization

In our research, we curated a dataset comprising approximately 1907 images depicting both diseased and healthy mango leaves. In Figure 3.2, the number of images for each class is displayed in a table. Figure 3.3 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. Figure 3.4 shows

the sample image before augmentation and Figure 3.5 shows the sample image after augmentation.

Number of Images Before Augmentation (Original 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

Figure 3.2: Number of Images Table Before Augmentation (Original Dataset)

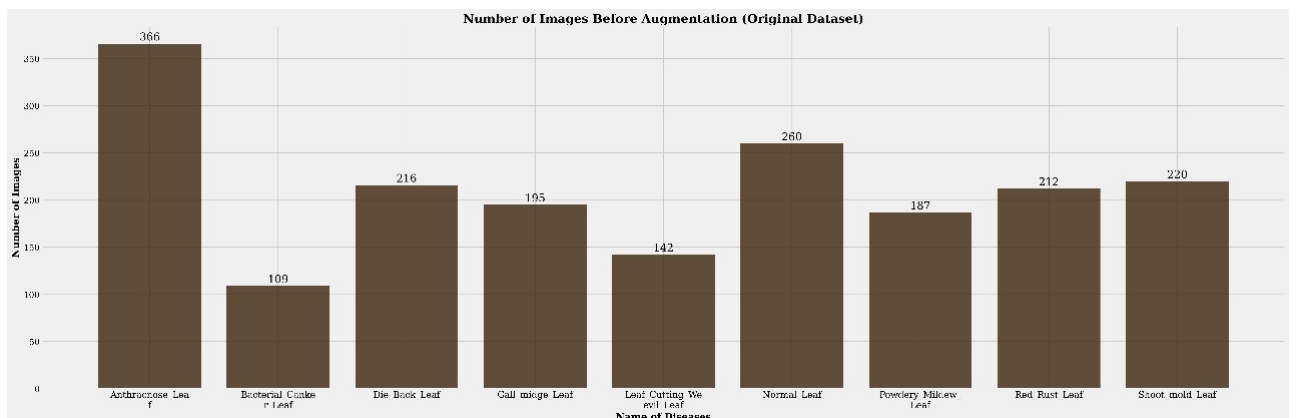


Figure 3.3: Number of Images Before Augmentation (Original Dataset)

3.4.2 Data Augmentation

Data augmentation can significantly enhance the performance and outcomes of machine learning models by generating new and diverse examples for training datasets. This is achieved using domain-specific strategies to create unique training examples from the existing data. If the dataset used by the machine learning model is extensive and sufficient, the model performs better and achieves higher accuracy. Image data augmentation is the most well-known type of data augmentation, involving the transformation of images from the training dataset into modified duplicates that belong to the same class as the original images. These transformations include shifts, flips, zooms, and other operations.

Examples of conventional data augmentation techniques include geometric transformations, color adjustments, rotation, reflection, and noise injection. Various strate-

gies are commonly applied in model training. Numerous studies indicate that conventional data augmentation techniques and other technologies can improve model performance, with geometric transformations being particularly significant [6] [5]. In our research, we expanded our dataset using a TensorFlow-based data augmentation approach, employing techniques such as rotation, and flipping. We only use data augmentation on our training data.

Figure 3.4 represents the number of images after augmentation per class. Figure 3.3 and Figure 3.5 shows the distribution of the images Before Augmentation and After Augmentation. Figure 3.6 and Figure 3.7 shows one sample image from the dataset before augmentation and after augmentation respectively.

Number of Training Images After Augmentation	
Diseases Name	No. of images
Anthracoese_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

Figure 3.4: Number of Training Images Table After Augmentation

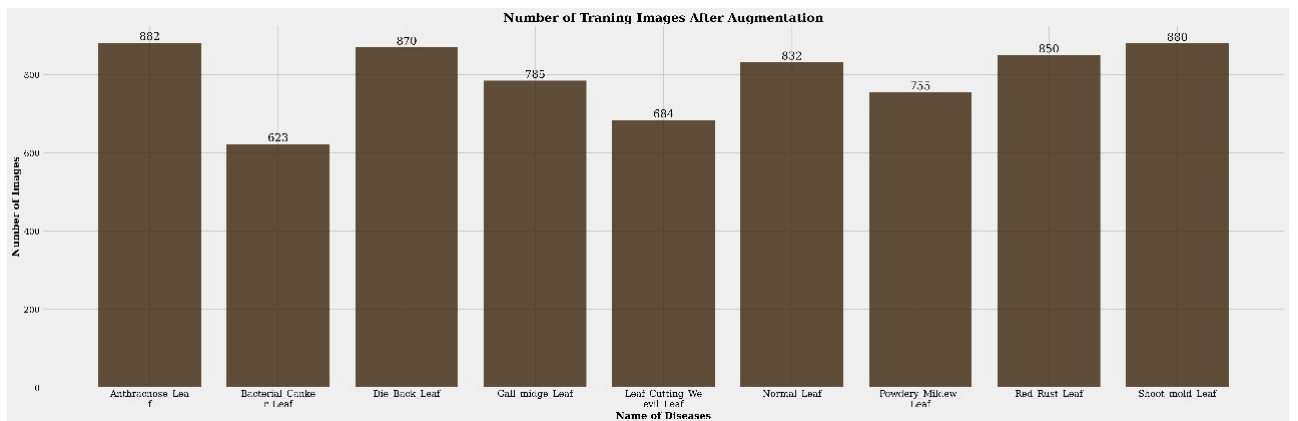


Figure 3.5: Number of Training Images After Augmentation

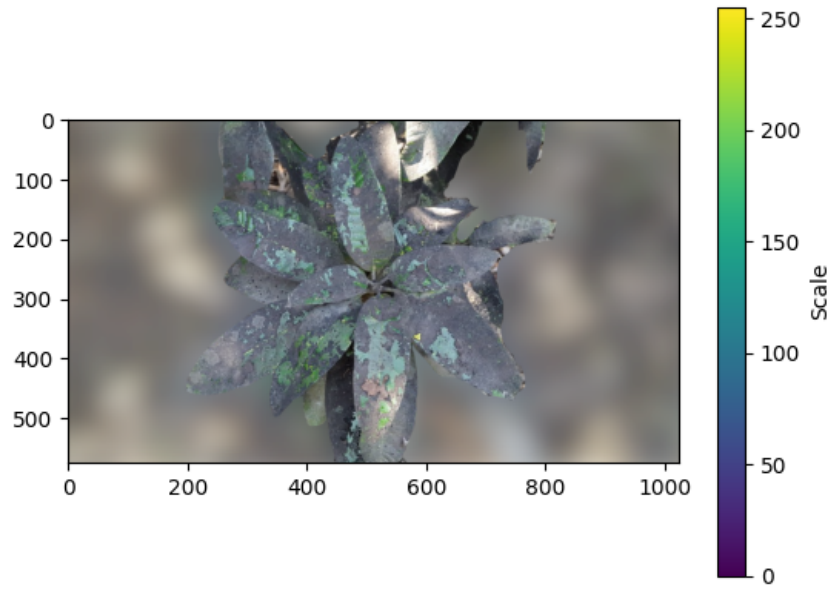


Figure 3.6: Before Augmentation

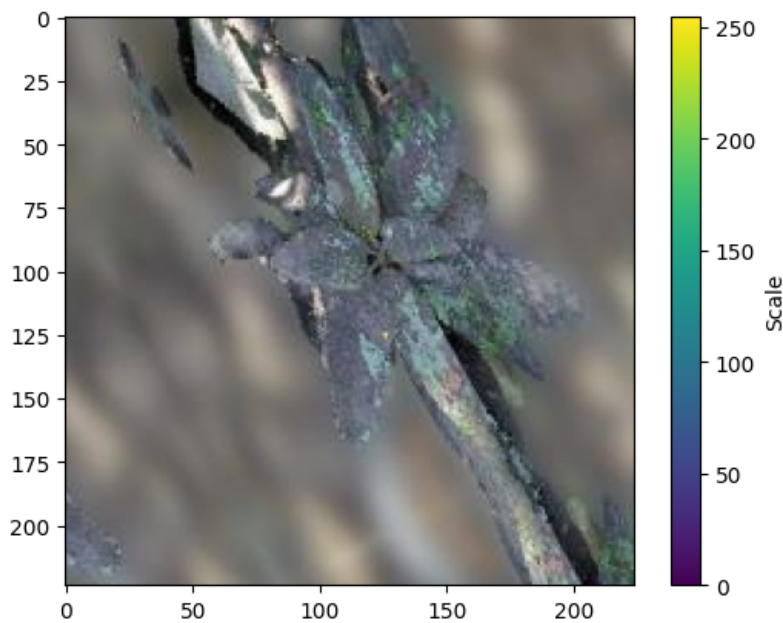


Figure 3.7: After Augmentation

3.4.3 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 3.8 shows the distribution of the dataset after splitting.

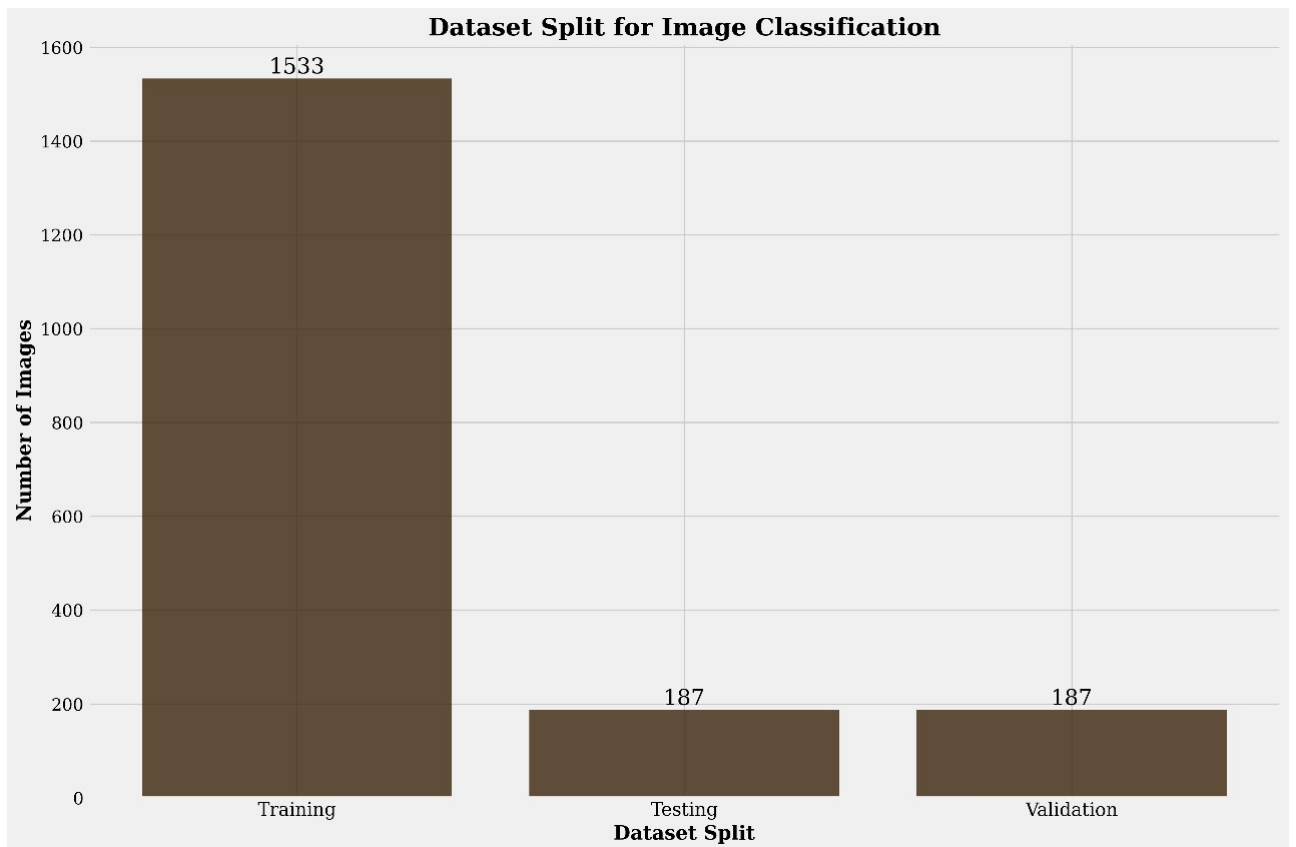


Figure 3.8: Dataset Split for Image Classification

3.5 Model Specification

3.5.1 Convolutional Neural Network(CNN)

Convolutional neural networks (CNNs) are a reliable neural network model, which is used not only in computer vision but also for image identification. The structure consists of three basic layers that are the convolutional layer, the pooling layer, and the fully connected layer. The first stage of a CNN is the input layer where raw data is inserted. It is called the convolution layer. It collects unprocessed pictures built with units of pixels, where each pixel indicates its color or lighting.

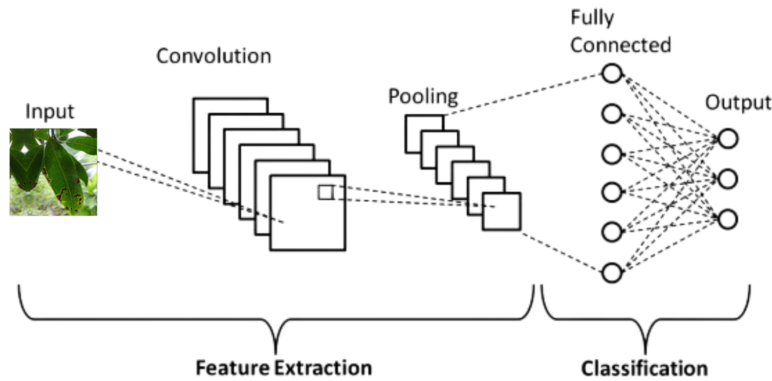


Figure 3.9: Architecture of CNN model

Convolution Layer: The convolutional layer is the most important part of The CNN model structure. It is mostly used for separating necessary data from raw input images. A dot product operation is executed by this layer between the input image and a set of smaller kernels of filters. The filters help the convolution layer to find various patterns such as edges, textures, and intricate structures. Essential visual features are captured and passed to subsequent layers for further examination. A stride is the size at which a kernel slides.

For a black and white image with dimensions $n*n*1$ and filter size $f*f*c$:

$$\text{Formula: } (n-f+1)*(n-f+1)*c \dots\dots\dots (1)$$

Here, 'n' is the image size, 'f' is the filter size, and 'c' represents the number of filters.

For a color image in RGB format with dimensions $n*n*3$ and the kernel size $f*f*3$:

$$\text{Formula: } (n-f+1)*(n-f+1)*1 \dots\dots\dots (2)$$

Here, 'n' denotes the image size, 'f' is the number of filters and we are calculating the output size for a single channel.

Finally, a feature map is obtained from the convolution layer processing that gives vital data about the input image, corners, and edges and serves as the input for subsequent layers, promoting advanced features and patterns.

Pooling Layer: After the convolutional layer comes the pooling layer which is used for minimising computational complexity. This layer performs the downsam-

pling of convolved feature maps which decreases the amount of input that needs to be compiled in subsequent layers. Three common pooling operations are mentioned here such as max pooling, sum pooling, and average pooling. Max pooling selects the maximum values from the feature map for required calculations whereas avg pooling takes the average of the values. However, sum pooling collects the total added value present in the feature map. These methods act as a bridge between the convolutional and fully connected layers by assisting in capturing the necessary data and minimizing the dimensionality of the data. [3]

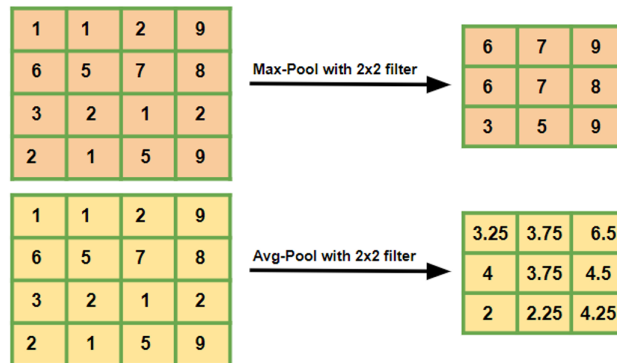


Figure 3.10: Example pooling

Therefore, the pooling layer is very important for generalising the separated features from the convolution layer and reducing computational requirements within the network. The pooling layer acts as the middle line between the convolution layer and fully connected layer, that results in smoother data processing which makes the model more potent in recognizing the important features.

Fully Connected Layers: In this layer, every neuron is interconnected with each neuron present in the subsequent layer. This step involves transformation of the input images into a flattened layer that is sent to the Fully Connected (FC) layer. Because of the flattened feature, the classification process is more efficient. As two connected layers result in a better model performance, a connection between two FC layers is established.

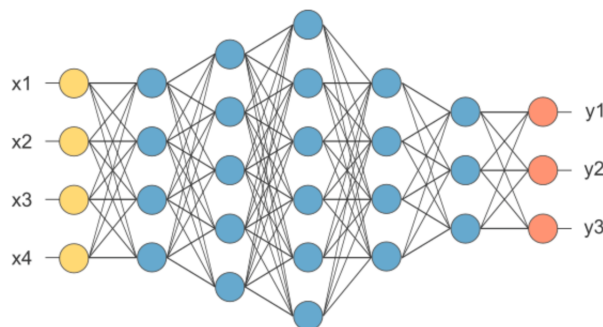


Figure 3.11: Fully connected layer

3.5.2 MobileNet V2

The MobileNetV2 model is a Convolutional Neural Network architecture that is mostly designed for portable mobile and edge devices known for its efficiency and performance. To begin with it has 53 convolutional layers and a single average pooling layer which results in around 350 GFLOP that is Gigaflop operations. Moreover, A key feature of MobileNetV2 is the implementation of depthwise independent convolutions layers that are divided into two operations- depthwise and pointwise methods. This model significantly reduces computation time by using individual spatial filters for each input channel. Therefore, MobileNetV2 achieves a perfect balance and equilibrium between computational precision and model performance that is much more effective for image processing and different applications that use different machine learning. The architecture consists of two key elements- effectiveness and also efficiency. The blocks are:

- Inverted Residual block with Stride 1.
- Bottleneck Residual Block with stride 2.

Internal component of stride 1 and 2 blocks:

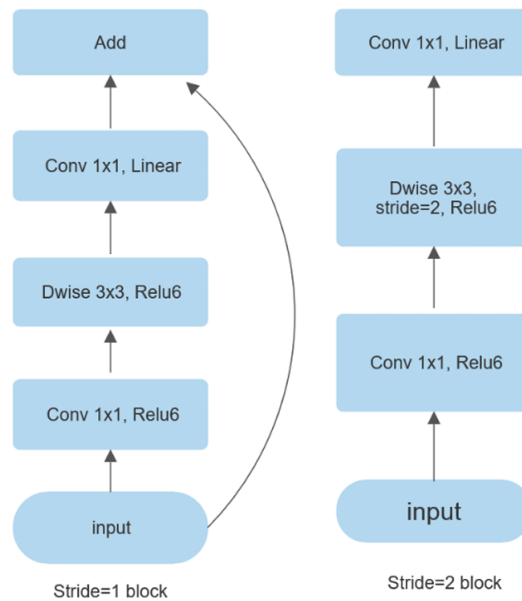


Figure 3.12: Fully connected layer

There are three different layers for each block:

- 1x1 Convolution with Relu6.
- Depthwise Convolution.
- 1x1 Convolution without any non-linearity.

3.5.3 VGG 16

VGG16 is a well known and widely recognized convolutional neural network architecture proposed by the Visual Graphics Group (VGG) at the University of Oxford. It is specially renowned for image classification because of its simplicity and uniform architecture. The VGG16 architecture is characterised by its depth. In VGG16 the “16” indicates the number of weight layers into the network, that is 13 convolutional layers and 3 fully connected layers.

The key feature of VGG16 is that this architecture uses small 3x3 convolutional filters with a stride of 1 and padding of 1 to maintain the spatial resolution. Moreover this model allows for the persistence of spatial information resulting in finer spatial details. Normally VGG16 uses max pooling with a 2x2 filter and stride of 2 to efficiently reduce the spatial dimensions from the input image.

VGG16 is designed in such a way that it can handle input images of size 224x224 pixels with three RGB colour channels ensuring that it can effectively capture and process visual information. The model consists of 13 convolutional layers divided into 5 blocks. A max-pooling layer is followed by each of the blocks. The architecture further includes 18 fully connected layers with the first three fully connected layers containing 4096 neurons each. The final output layer is a softmax layer that gives output a probability distribution over the 1000 classes.

VGG16 is now widely used in various computer vision tasks, object detection, and feature extraction and marked a significant improvement over previous configurations showcasing the power of deep neural networks in computer vision. Besides it is widely used for transfer learning due to its simplicity or uniform architecture and strong feature extraction capabilities. At last, VGG16 is influencing practical applications in various fields including image recognition, object detection, and more.

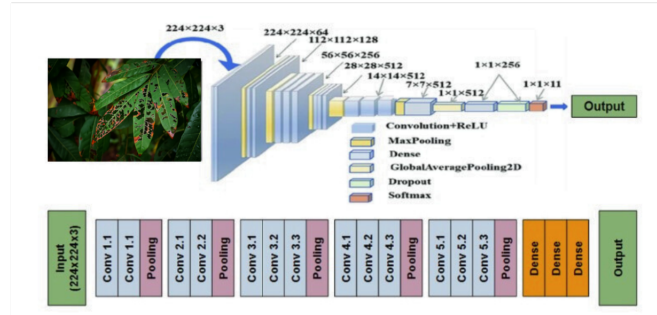


Figure 3.13: VGG16 architecture

3.5.4 DenseNet169

DenseNet169 is a kind of densely connected convolutional network with 169 layers; this includes convolutional layers, pooling, dense blocks, transition layers, and final layers. It utilizes a feed-forward system in such a way that each layer is connected to all previous layers, therefore enhancing feature re-use and smoothness in the vanishing gradient problem.

Suitable for input images of size 224x224 pixels with RGB colour channels, DenseNet169 performs very well in image classification. Dense connectivity patterns are the cause of its improved gradient flows and network efficiency compared with earlier architectures.

Exciting DenseNet169 is accomplished by the dense blocks, each made of some convolutional layers set as an input of the previous layers. Transition layers in between dense blocks help in managing feature map sizes along with keeping network complexity in check.

DenseNet169 was an advance important beyond prior DNNs, which showed its impact on many applications of computer vision for academic research to practical applications of image recognition and detection. Partly being more efficient at the extraction and classification of features, it has contributed to actually obtaining

high-level analysis of visual data.

In general terms, DenseNet169 has shown the gains that the dense connections bring, now setting benchmarks for both performance and efficiency in image-processing tasks.

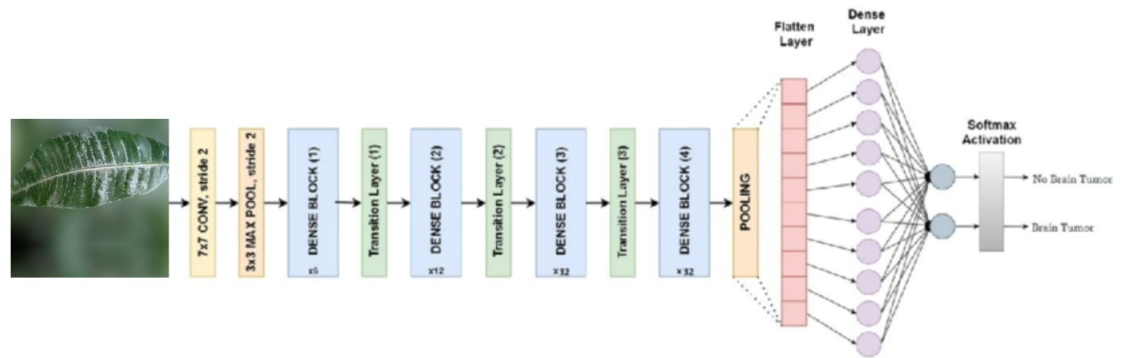


Figure 3.14: DenseNet169 model architecture

3.5.5 InceptionV3

InceptionV3 is a widely used architecture in convolutional neural Networks, introduced by Google. The motive behind launching InceptionV3 is to use multiple convolution filters of different sizes to detect various characteristics from the input image that helps in recognising different spatial features from the input image. As InceptionV3 provides unique performance and efficiency that's why this model is particularly organised for image processing and classification tasks. The use of 1x1 convolutions has made this architecture more exceptional, which eventually improves computational efficiency during dataset training and inference. [16]

InceptionV3 speeds up training and enhances the model's robustness by normalising the activations of the previous layer by using batch normalisation. Moreover, this model provides more efficient grid sizes, and factorised convolutional layers. Therefore, auxiliary classifiers are also included in InceptionV3, making its performance more efficient and accurate. [4]

By breaking larger convolutions into smaller convolutions, InceptionV3 strikes a brilliant balance between complexity and efficiency. As InceptionV3 architecture provides auxiliary classifiers, it provides additional gradient signals and improves convergence while dataset training. By using label smoothing InceptionV3 prevents overfitting. InceptionV3 has not only pushed the boundaries of image processing but also set a new milestone for future CNN architectures to aspire to. [11]

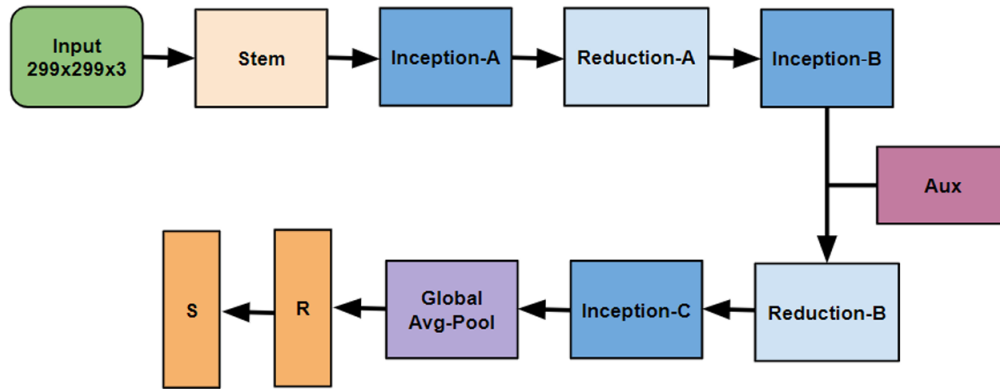


Figure 3.15: InceptionV3 architecture

Chapter 4

Implementation

4.1 Workflow

Our research journey started with the development of a dataset tailored for mango leaf disease detection. We created our own extensive dataset, capturing leaf images from various mango gardens and trees in different regions of Bangladesh. This involved a huge data collection and preprocessing phase, ensuring a diverse representation of leaf condition and disease manifestation.

The dataset was carefully divided into segments allocated for training, validation, and testing. Approximately 80% of the data was used for training the deep learning models, 10% for validating their performance, and the remaining 10% for final testing to assess the model's accuracy and generalization capabilities.

The model selection process was a critical step where we identified and evaluated suitable models for leaf disease detection. Our chosen models fell into two categories: custom models, in which we customized a model based on CNN architecture including ConvolutionNet-5, ConvolutionNet-3, ConvolutionNet-4, ConvolutionNet-4M1, ConvolutionNet-5M0, ConvolutionNet-5M1, ConvolutionNet-6). On the other hand, pre-trained models, such as VGG16, MobileNetV2, InceptionV3, and DenseNet169, offered a solid foundation due to their proven success in various image classification tasks.

With the models selected, we proceeded to set up and fine-tune their hyperparameters for optimal performance. This implementation phase involved training and evaluating both our custom design model and the chosen pre-trained models on our mango leaf disease dataset. The goal was to identify the most promising models that could accurately detect and classify leaf diseases, providing a practical solution for farmers to enhance mango cultivation practices.

Our research plan, as illustrated in Figure 4.1, showcases the evolution of our project, from the creation of the specialized dataset to the selection and implementation of models. It highlights our systematic and comprehensive approach ensuring the effective utilization of deep learning techniques for accurate mango leaf disease detection.

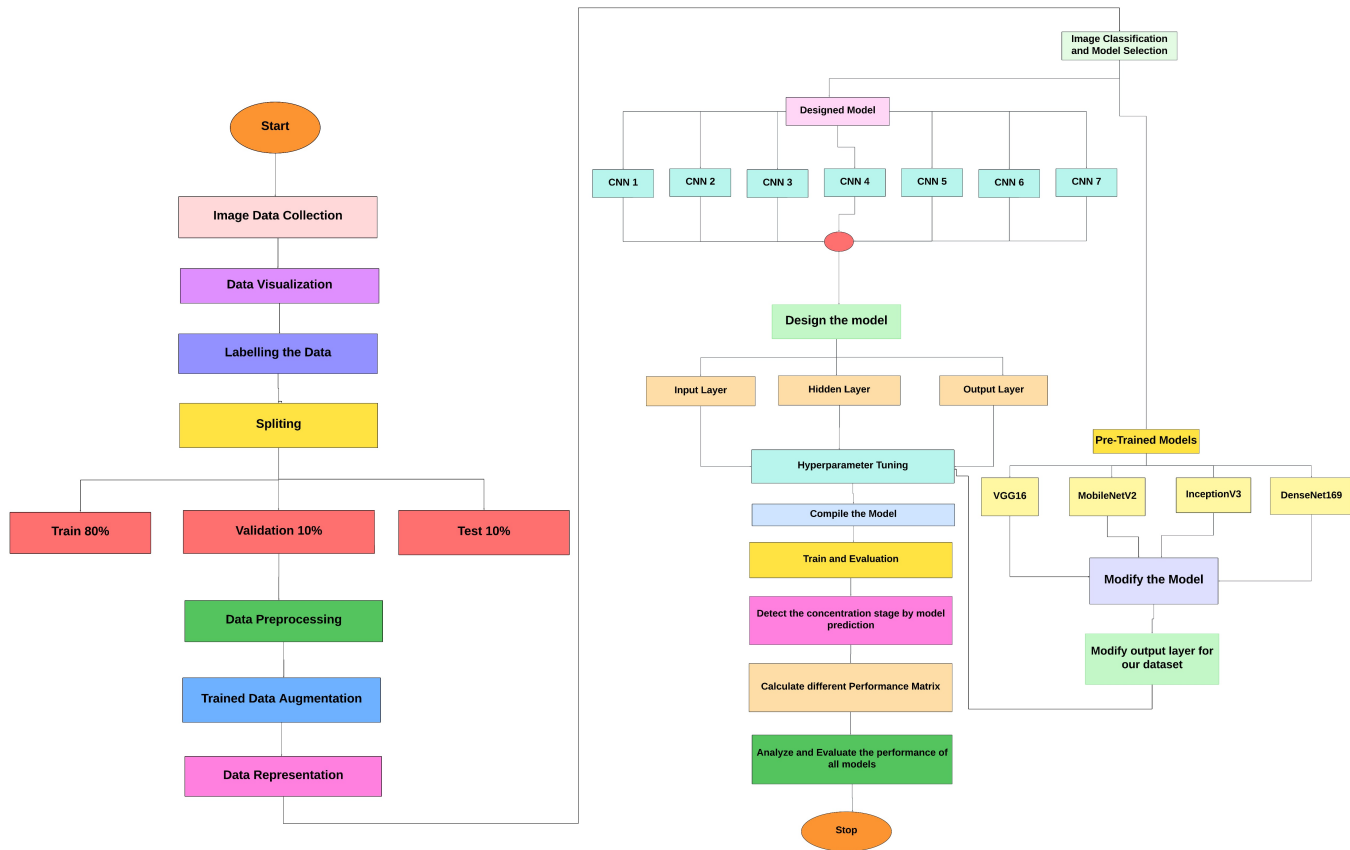


Figure 4.1: Workflow details for implementation

4.2 Setup for Experiment

4.2.1 Training hardware and Software

Experiments were conducted on a device with the following hardware specifications:

CPU	Core i5 11th Gen
GPU	NVIDIA GeForce GTX 1650 Ti
RAM	16GB
ROM	1TB
OS	Windows 11

Figure 4.2: Summary of hardware characteristics

On this device, we used Google Colab to run the code.

4.2.2 Library List

Serial No	Library	Version
1	OpenCV	4.8.1
2	Numpy	1.13.4
3	Matplotlib	3.7.0
4	Pandas	1.5.2
5	Scikit-learn	1.2.1
6	PIL	9.3.0
7	Keras	2.15.0
8	Tensorflow	2.15.0

Figure 4.3: Summary of Python library list

4.2.3 Structural view of the code skeleton

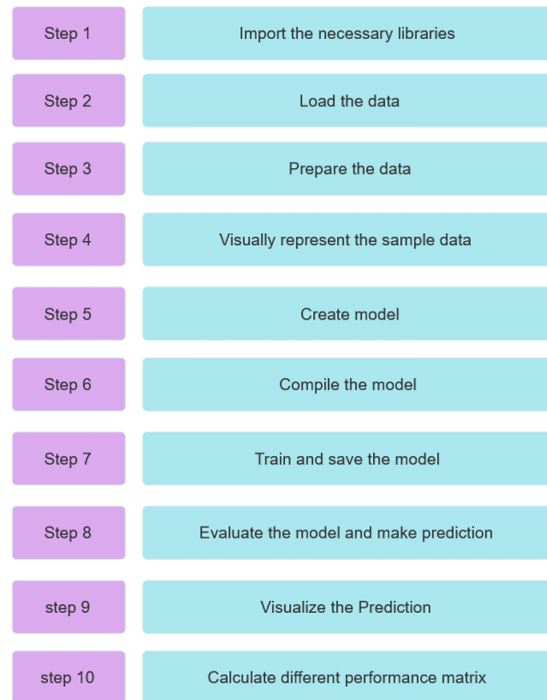


Figure 4.4: Structural view of the code skeleton

4.3 Model Selection

For the concentration measurement task, we focused on selecting lightweight models to efficiently classify images. We employed a CNN-based architecture including (ConvolutionNet-5, ConvolutionNet-3, ConvolutionNet-4, ConvolutionNet-4M1, ConvolutionNet-5M0, ConvolutionNet-5M1, ConvolutionNet-6), designing custom model that strikes a balance between accuracy and computational efficiency. Our designed model is very lightweight while delivering impressive accuracy.

In our project, we have chosen four pre-trained models which are known for their lightweight architecture compared to other pre-trained models. These pre-trained models are VGG16, MobileNetV2, InceptionV3, and DenseNet169, each of them giving a unique combination of performance and accuracy.

While doing the model selection, we have given a strong emphasis on both robustness and unique performance. We have carefully handled the transfer learning layer, which has ensured that the models can easily adapt to our specific measurement task, despite not showing similar data previously. So the careful selection and fine-tuning process will enhance the models' performance and efficiency in our workflow. Figure 4.5 shows the diagram of model selection.

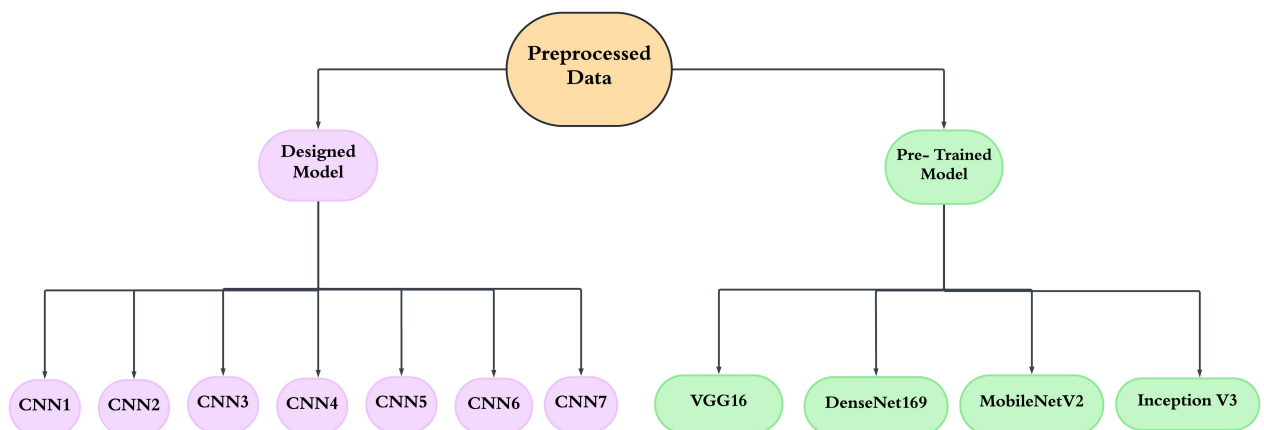


Figure 4.5: Diagram of model selection

4.4 Hyperparameter Tuning

Hyperparameters are those settings that we specify when calling a function or class. These settings play a crucial role for the better performance of an algorithm. The accurate values can significantly increase the algorithm's efficiency and performance. So for better performance of the models, proper hyperparameter selection is very much necessary that directly influences better efficiency and performance. So, by selecting appropriate hyperparameter values, essential success for any machine learning algorithm can be ensured.

For instance, when we trained our model, we initially used Adam as the optimizer with a learning rate of 0.001, resulting in very low validation accuracy. By adjusting the learning rate to 0.00001 to 0.000001, we observed a significant improvement in validation accuracy. In another scenario, we set the steps per epoch value randomly while training the model, which prevented the model from fully training because it could only learn from a limited number of images ('steps per epoch * batch size'). To ensure the model trained on the entire dataset, we adjusted 'steps per epoch' to be the number of images in the training dataset divided by the batch size. This adjustment led to the expected validation accuracy.

4.5 Design and Compile the Models

4.5.1 ConvolutionNet-5

The model consists of five convolutional layers, along with a max-pooling layer with a 2x2 pool size in each convolutional layer. Max-pooling layer used for down sampling. It reduces the image information. As a result, the computational cost gets reduced. The first layer has 16 filters, the second has 32 filters, the third has 64 filters, the fourth has 128 filters, the fifth has 256 filters. We used ReLU as activation function in every convolutional layer. Then we implement flatten layer. The flattened output is passed through a densely connected layer with 256 and 512 neurons and a ReLU activation function. We use five BatchNormalization before every max-pooling layer and two after densely connected layer for stabilizes, accelerates training, improves generalization. After that, we used a dropout layer with a dropout rate of 0.5. This layer helps to allay from overfitting. Finally, we implement the output layer that consists eleven neurons with a softmax activation function. The softmax function is suitable for multiclass. The aim of this model is to learn hierarchical features through convolution and pooling operations, dense layers for classification, while dropout helps enhance generalization by preventing overfitting during training. Then we compile the model using the Adam optimizer with a learning rate of 0.00001. We choose sparse categorical crossentropy as loss function, which is suitable for multiclass classification. Additionally, the accuracy metric is specified for monitoring the model's performance during training and evaluation.

4.5.2 ConvolutionNet-5M0

The model consists of five convolutional layers, along with a max-pooling layer with a 2x2 pool size in each convolutional layer. Max-pooling layer used for downsampling. It reduces the image information. As a result, the computational cost gets reduced. The first layer has 32 filters, the second has 64 filters, the third has 128 filters, the fourth has 256 filters, and the fifth has 512 filters. We used ReLU as activation function in every convolutional layer. Then we implement flatten layer. The flattened output is passed through a densely connected layer with 512 and 1024 neurons and a ReLU activation function. We use five BatchNormalization before every max-pooling layer and two after densely connected layer to stabilize, accelerates training, improves generalization. After that, we used a dropout layer with a dropout rate of 0.5. This layer helps to allay from overfitting. Finally, we implement the output layer that consists eleven neurons with a softmax activation function. The softmax function is suitable for multiclass. The aim of this model is to learn hierarchical features through convolution and pooling operations, dense layers for classification, while dropout helps enhance generalization by preventing overfitting during training. Then we compile the model using the Adam optimizer with a learning rate of 0.0001. We choose sparse categorical crossentropy as loss function, which is suitable for multiclass classification. Additionally, the accuracy metric is specified for monitoring the model's performance during training and evaluation.

4.5.3 ConvolutionNet-5M1

The model consists of five convolutional layers, along with a max-pooling layer with a 2x2 pool size in each convolutional layer. Max-pooling layer used for down sampling. It reduces the image information. As a result, the computational cost gets reduced. The first layer has 16 filters, the second has 32 filters, the third has 64 filters, the fourth has 128 filters, the fifth has 256 filters. We used ReLU as activation function in every convolutional layer. Then we implement flatten layer. The flattened output is passed through a densely connected layer with 256 and 512 neurons and a ReLU activation function. We use five BatchNormalization before every max-pooling layer and two after densely connected layer for stabilizes, accelerates training, improves generalization. After that, we used a dropout layer with a dropout rate of 0.4. This layer helps to allay from overfitting. Finally, we implement the output layer that consists eleven neurons with a softmax activation function. The softmax function is suitable for multiclass. The aim of this model is to learn hierarchical features through convolution and pooling operations, dense layers for classification, while dropout helps enhance generalization by preventing overfitting during training. Then we compile the model using the AdamW optimizer with a learning rate of 0.00001. We choose sparse categorical crossentropy as loss function, which is suitable for multiclass classification. Additionally, the accuracy metric is specified for monitoring the model's performance during training and evaluation.

4.5.4 ConvolutionNet-4

The model consists of four convolutional layers, along with a max-pooling layer with a 2x2 pool size in each convolutional layer. Max-pooling layer used for down sampling. It reduces the image information. As a result, the computational cost gets reduced. The first layer has 32 filters, the second has 64 filters, the third has 128 filters and the fourth has 256 filters. We used ReLU as activation function in every convolutional layer. Then we implement flatten layer. The flattened output is passed through a densely connected layer with 512 and 256 neurons and a ReLU activation function. We use four BatchNormalization before every max-pooling layer and two after densely connected layer for stabilizes, accelerates training, improves generalization. After that, we used a dropout layer with a dropout rate of 0.5. This layer helps to allay from overfitting. Finally, we implement the output layer that consists eleven neurons with a softmax activation function. The softmax function is suitable for multiclass. The aim of this model is to learn hierarchical features through convolution and pooling operations, dense layers for classification, while dropout helps enhance generalization by preventing overfitting during training. Then we compile the model using the Adam optimizer with a learning rate of 0.00001. We choose sparse categorical crossentropy as loss function, which is suitable for multiclass classification. Additionally, the accuracy metric is specified for monitoring the model's performance during training and evaluation.

4.5.5 ConvolutionNet-4M1

The model consists of four convolutional layers, along with a max-pooling layer with a 2x2 pool size in each convolutional layer. Max-pooling layer used for down sampling. It reduces the image information. As a result, the computational cost

gets reduced. The first layer has 16 filters, the second has 32 filters, the third has 64 filters and the fourth has 256 filters. We used ReLU as activation function in every convolutional layer. Then we implement flatten layer. The flattened output is passed through a densely connected layer with 256 and 512 neurons and a ReLU activation function. We use four BatchNormalization before every max- pooling layer and two after densely connected layer to stabilize, accelerates training, improves generalization. After that, we used a dropout layer with a dropout rate of 0.5. This layer helps to allay from overfitting. Finally, we implement the output layer that consists eleven neurons with a softmax activation function. The softmax function is suitable for multiclass. The aim of this model is to learn hierarchical features through convolution and pooling operations, dense layers for classification, while dropout helps enhance generalization by preventing overfitting during training. Then we compile the model using the Adam optimizer with a learning rate of 0.00001. We choose sparse categorical crossentropy as loss function, which is suitable for multiclass classification. Additionally, the accuracy metric is specified for monitoring the model's performance during training and evaluation.

4.5.6 ConvolutionNet-6

The model consists of four convolutional layers, along with a max-pooling layer with a 2x2 pool size in each convolutional layer. Max-pooling layer used for down sampling. It reduces the image information. As a result, the computational cost gets reduced. The first layer has 16 filters, the second has 32 filters, the third has 64 filters, the fourth has 128 filters, the fifth has 256 filters and the fifth has 512 filters. We used ReLU as activation function in every convolutional layer. Then we implement flatten layer. The flattened output is passed through a densely connected layer with 256 and 256 neurons and a ReLU activation function. We use six BatchNormalization before every max-pooling layer and two after densely connected layer for stabilizes, accelerates training, improves generalization. After that, we used a dropout layer with a dropout rate of 0.5. This layer helps to allay from overfitting. Finally, we implement the output layer that consists eleven neurons with a softmax activation function. The softmax function is suitable for multiclass. The aim of this model is to learn hierarchical features through convolution and pooling operations, dense layers for classification, while dropout helps enhance generalization by preventing overfitting during training. Then we compile the model using the Adam optimizer with a learning rate of 0.00001. We choose sparse categorical crossentropy as loss function, which is suitable for multiclass classification. Additionally, the accuracy metric is specified for monitoring the model's performance during training and evaluation.

4.5.7 ConvolutionNet-3

The model consists of three convolutional layers, along with a max-pooling layer with a 2x2 pool size in each convolutional layer. Max-pooling layer used for down sampling. It reduces the image information. As a result, the computational cost gets reduced. The first layer has 16 filters, the second has 32 filters and the third has 64 filters. We used ReLU as activation function in every convolutional layer. Then we implement flatten layer. The flattened output is passed through a densely

connected layer with 256 neurons and a ReLU activation function. We use three BatchNormalization before every max-pooling layer and one after densely connected layer for stabilizes, accelerates training, improves generalization. After that, we used a dropout layer with a dropout rate of 0.2. This layer helps to allay from overfitting. Finally, we implement the output layer that consists eleven neurons with a softmax activation function. The softmax function is suitable for multiclass. The aim of this model is to learn hierarchical features through convolution and pooling operations, dense layers for classification, while dropout helps enhance generalization by preventing overfitting during training. Then we compile the model using the Adam optimizer with a learning rate of 0.000001. We choose sparse categorical crossentropy as loss function, which is suitable for multiclass classification. Additionally, the accuracy metric is specified for monitoring the model's performance during training and evaluation.

4.5.8 VGG16

The VGG16 is one of the pre-trained models that trained on a large amount of dataset. At first, we loaded the model and removed its original fully connected layers. Secondly, we have modified it by adding a global average pooling layer to the output of the VGG16 model. After this, by using ReLU as the activation function we added a densely connected layer with 512 and 256 neurons. We have used a dropout layer with a rate of 0.5. The final output layer is a dense layer with three neurons. We have employed the softmax activation function, which is appropriate for multi-class classification. By retaining only the weights of the mode we freeze the top layer and the base model. This process can be considered as feature extraction. Through this setup, the pre-trained VGG16 features can be leveraged for accurate image feature extraction.

Then we can compile the model using the Adam optimizer with a learning rate of 0.00001. During the time of compilation, we can use sparse categorical crossentropy that is considered as the loss function, which is suitable for multi-class classification. Finally, we specified the accuracy metric for monitoring the model's performance during training and evaluation.

4.5.9 InceptionV3

The InceptionV3 is one of the pre-trained models that trained on a large amount of dataset. At first, we loaded the model and removed its original fully connected layers. Secondly, we have modified it by adding a global average pooling layer to the output of the InceptionV3 model. After this, by using ReLU as the activation function we added a densely connected layer with 128 and 512 neurons. We have used a dropout layer with a rate of 0.5. The final output layer is a dense layer with three neurons. We have employed the softmax activation function, which is appropriate for multi-class classification. By retaining only the weights of the mode we freeze the top layer and the base model. This process can be considered as feature extraction. Through this setup, the pretrained InceptionV3 features can be leveraged for accurate image feature extraction.

Then we can compile the model using the Adam optimizer with a learning rate of 0.00001. During the time of compilation, we can use sparse categorical crossentropy that is considered as the loss function, which is suitable for multi-class classification. Finally, we specified the accuracy metric for monitoring the model's performance during training and evaluation.

4.5.10 MobileNetV2

The MobileNetv2 is one of the pre-trained models that trained on a large amount of dataset. At first, we loaded the model and removed its original fully connected layers. Secondly, we have modified it by adding a global average pooling layer to the output of the MobileNetv2 model. After this, by using ReLU as the activation function we added a densely connected layer with 128 and 512 neurons. We have used a dropout layer with a rate of 0.5. The final output layer is a dense layer with three neurons. We have employed the softmax activation function, which is appropriate for multi-class classification. By retaining only the weights of the model we freeze the top layer and the base model. This process can be considered as feature extraction. Through this setup, the pretrained MobileNetv2 features can be leveraged for accurate image feature extraction.

Then we can compile the model using the Adam optimizer with a learning rate of 0.00001. During the time of compilation, we can use sparse categorical cross-entropy that is considered as the loss function, which is suitable for multi-class classification. Finally, we specified the accuracy metric for monitoring the model's performance during training and evaluation.

4.5.11 DenseNet169

The DenseNet169 is one of the pre-trained models that trained on a large amount of dataset. At first, we loaded the model and removed its original fully connected layers. Secondly, we have modified it by adding a global average pooling layer to the output of the DenseNet169 model. After this, by using ReLU as the activation function we added a densely connected layer with 128 and 512 neurons. We have used a dropout layer with a rate of 0.5. The final output layer is a dense layer with three neurons. We have employed the softmax activation function, which is appropriate for multi-class classification. By retaining only the weights of the model we freeze the top layer and the base model. This process can be considered as feature extraction. Through this setup, the pre-trained DenseNet169 features can be leveraged for accurate image feature extraction.

Then we can compile the model using the Adam optimizer with a learning rate of 0.00001. During the time of compilation, we can use sparse categorical cross entropy which is considered as the loss function, which is suitable for multi-class classification. Finally, we specified the accuracy metric for monitoring the model's performance during training and evaluation.

Chapter 5

Result Analysis

5.1 Train and evaluate the Models

5.1.1 ConvolutionNet-5

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.9411 and validation accuracy is 0.9130. Figures 4.6 and 4.7 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

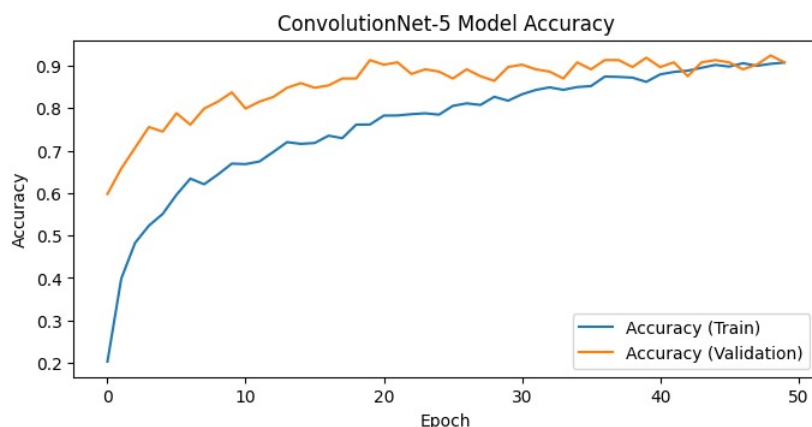


Figure 5.1: Training_accuracy Vs Validation_accuracy of ConvolutionNet-5

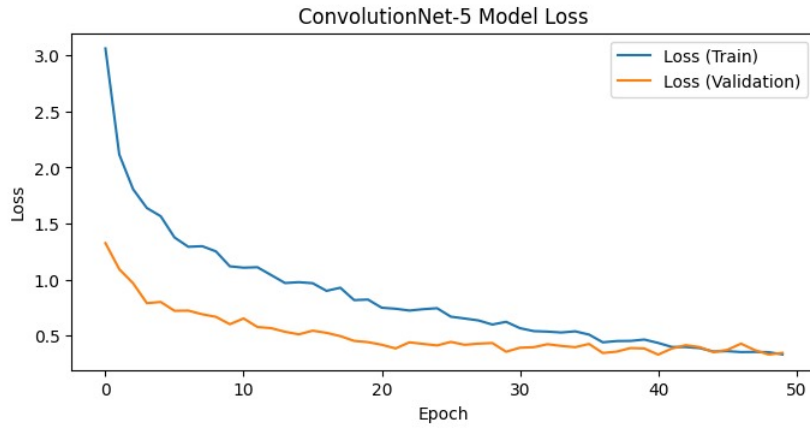


Figure 5.2: Training_loss Vs Validation_loss of ConvolutionNet-5

5.1.2 ConvolutionNet-3

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8877 and validation accuracy is 0.8478. Figures 4.8 and 4.9 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

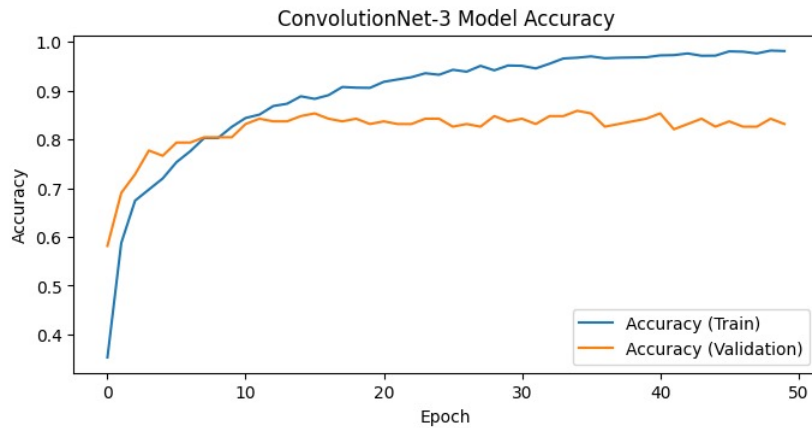


Figure 5.3: Training_accuracy Vs Validation_accuracy of ConvolutionNet-3

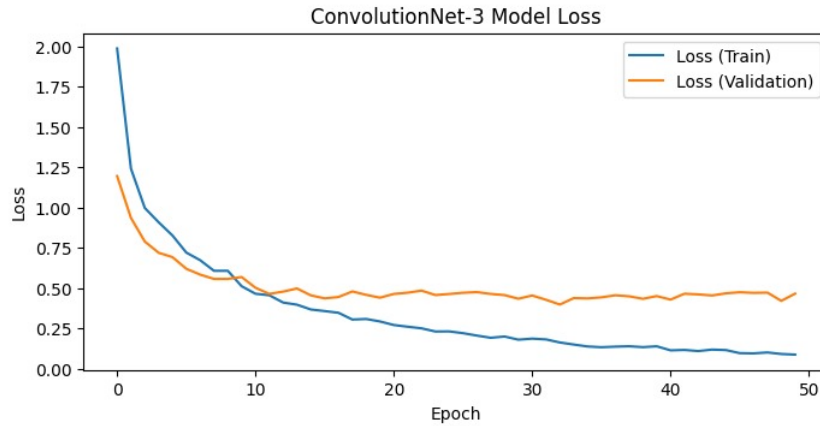


Figure 5.4: Training_loss Vs Validation_loss of ConvolutionNet-3

5.1.3 ConvolutionNet-4

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.9144 and validation accuracy is 0.8804. Figures 4.10 and 4.11 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

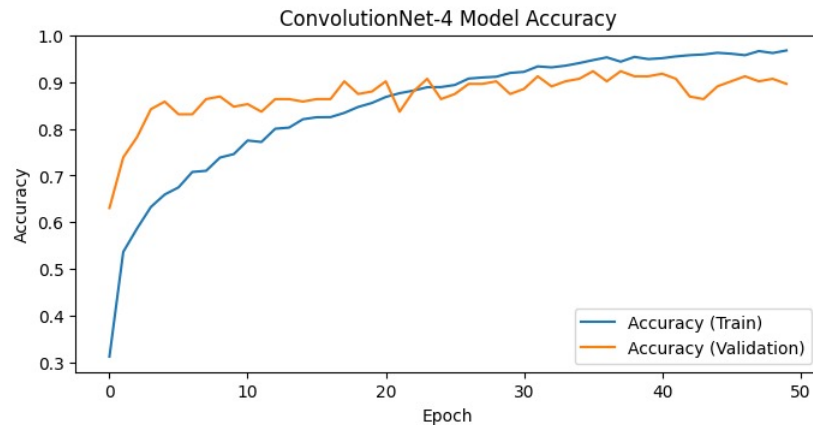


Figure 5.5: Training_accuracy Vs Validation_accuracy of ConvolutionNet-4

5.1.4 ConvolutionNet-4M1

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training

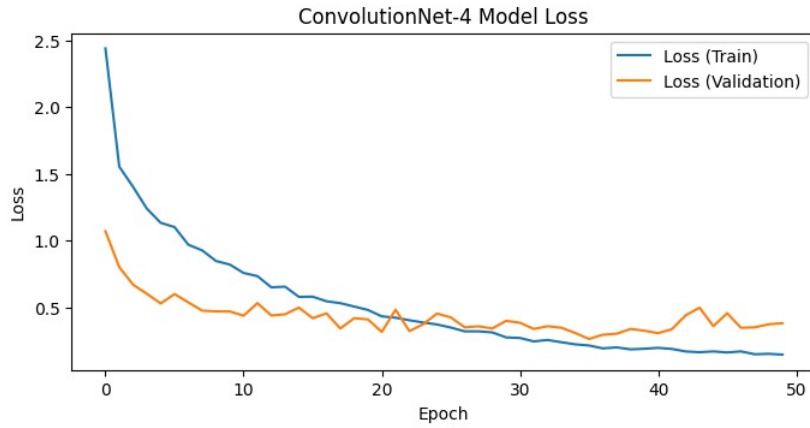


Figure 5.6: Training_loss Vs Validation_loss of ConvolutionNet-4

will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8983 and validation accuracy is 0.8967. Figures 4.12 and 4.13 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

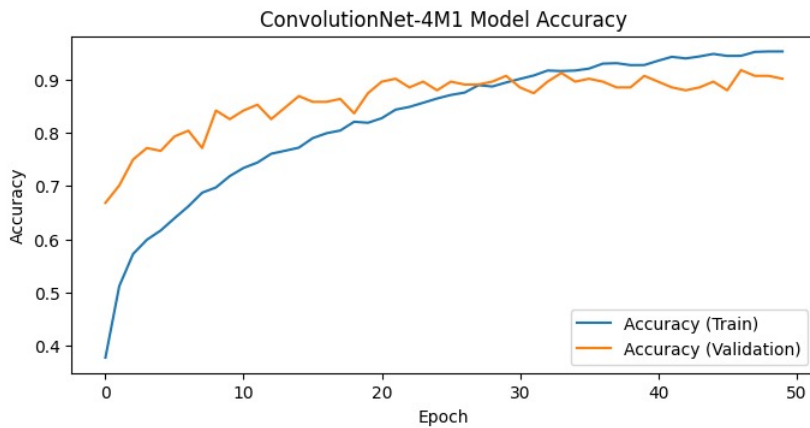


Figure 5.7: Training_accuracy Vs Validation_accuracy of ConvolutionNet-4M1

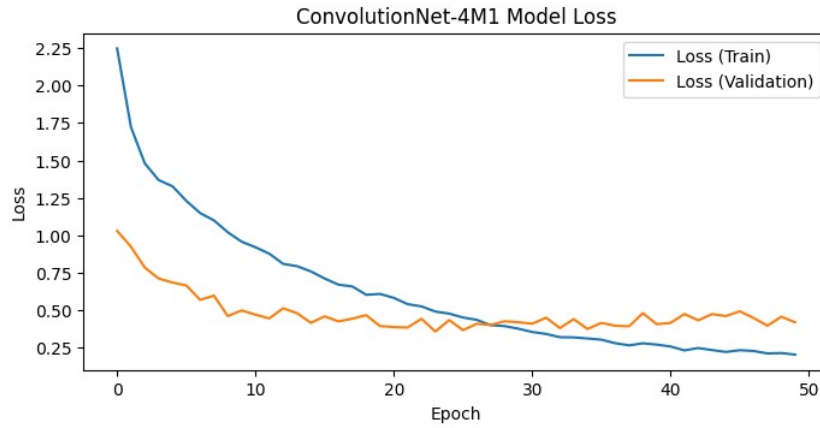


Figure 5.8: Training_loss Vs Validation_loss of ConvolutionNet-4M1

5.1.5 ConvolutionNet-5M0

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8823 and validation accuracy is 0.9022. Figures 4.14 and 4.15 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

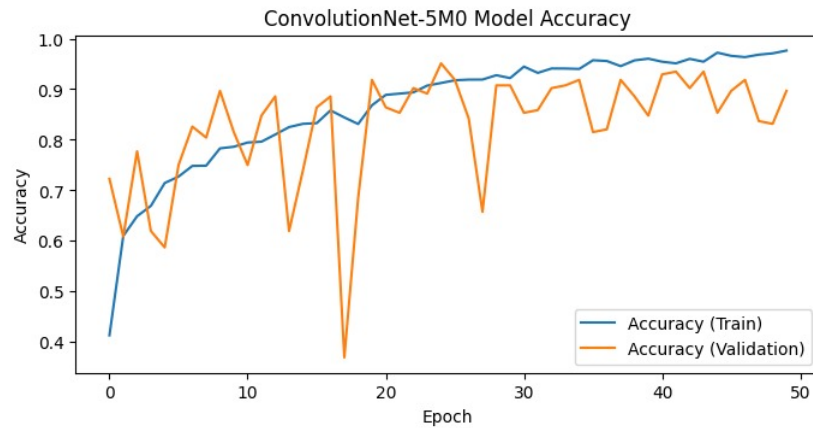


Figure 5.9: Training_accuracy Vs Validation_accuracy of ConvolutionNet-5M0

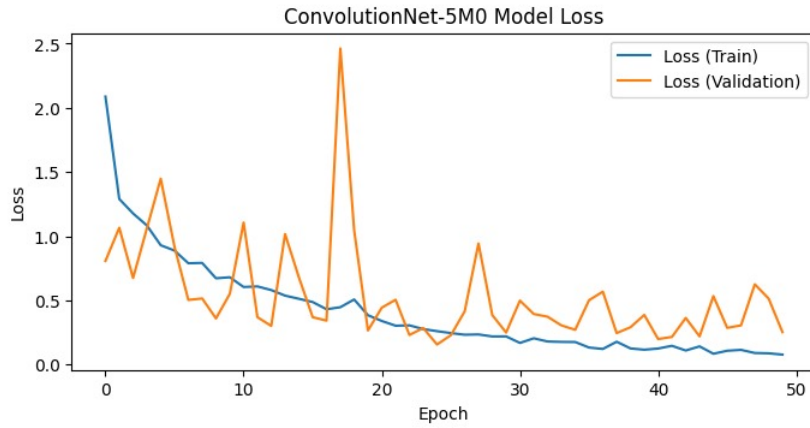


Figure 5.10: Training_loss Vs Validation_loss of ConvolutionNet-5M0

5.1.6 ConvolutionNet-5M1

We train this model by providing the train and validation dataset. The batch size is 16. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one Sixteen of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.9144 and validation accuracy is 0.8920. Figures 4.16 and 4.17 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

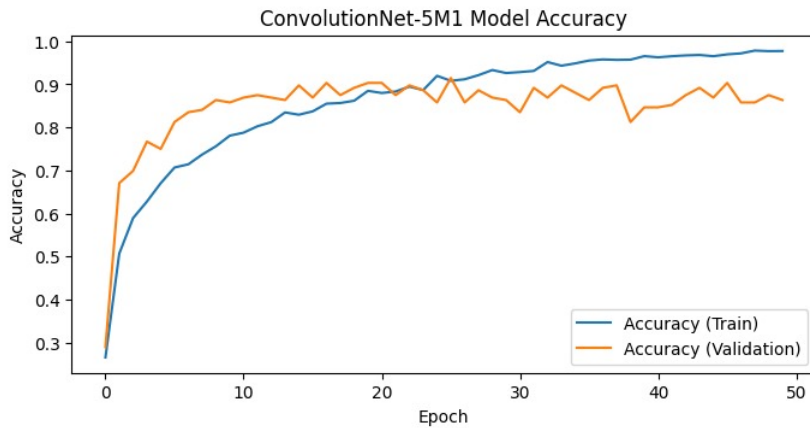


Figure 5.11: Training_accuracy Vs Validation_accuracy of ConvolutionNet-5M1

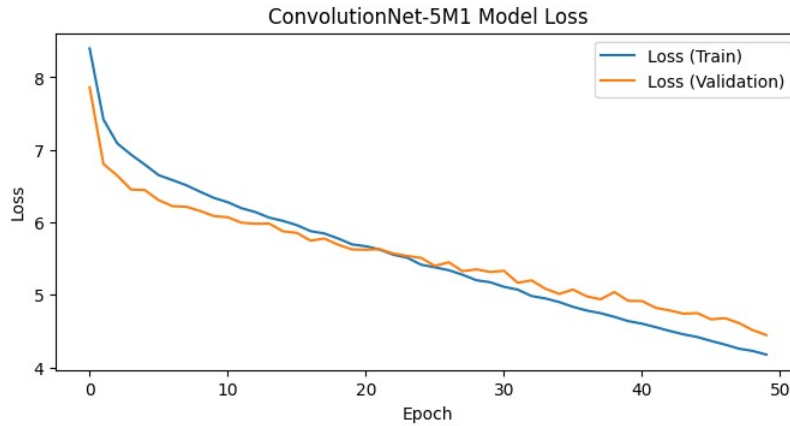


Figure 5.12: Training_loss Vs Validation_loss of ConvolutionNet-5M1

5.1.7 ConvolutionNet-6

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8823 and validation accuracy is 0.8587. Figures 4.18 and 4.19 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

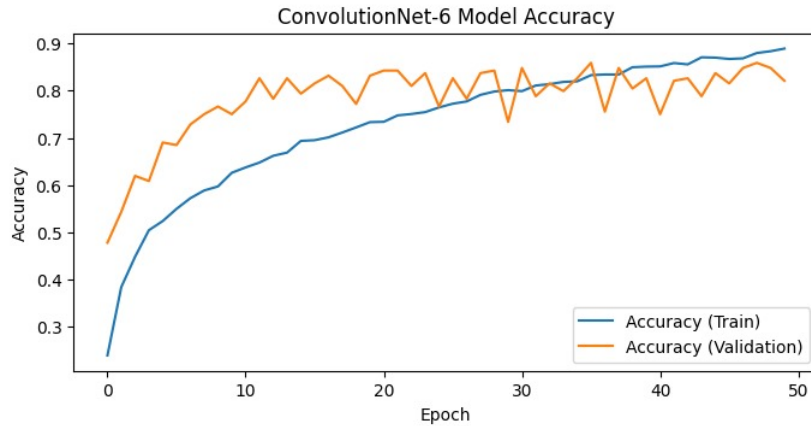


Figure 5.13: Training_accuracy Vs Validation_accuracy of ConvolutionNet-6

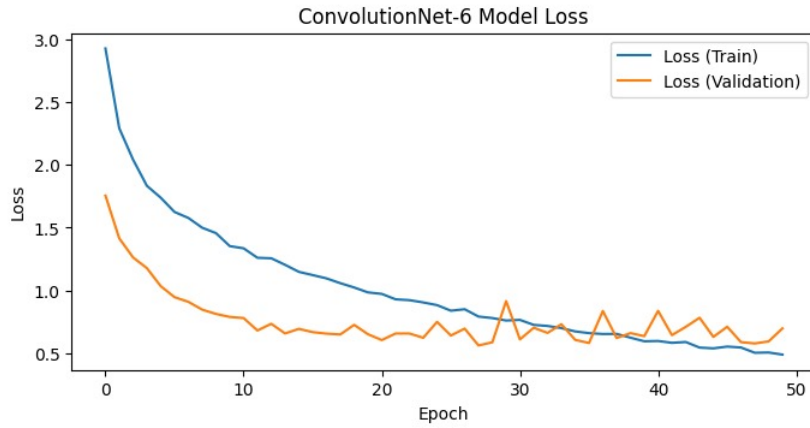


Figure 5.14: Training_loss Vs Validation_loss of ConvolutionNet-6

5.1.8 VGG16

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8770 and validation accuracy is 0.8261. Figures 4.20 and 4.21 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

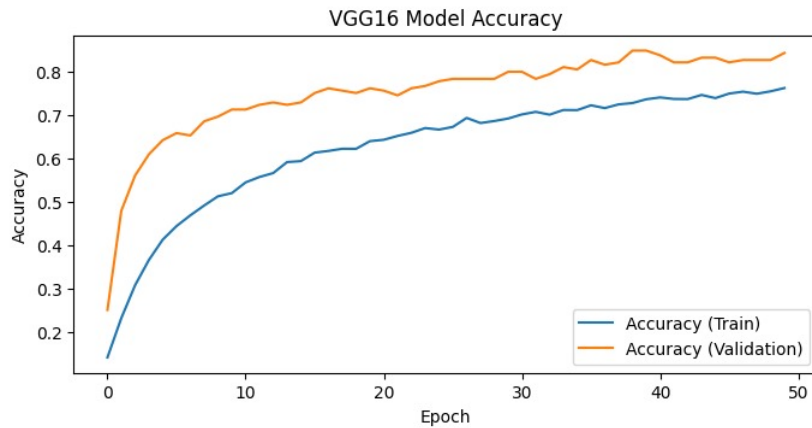


Figure 5.15: Training_accuracy Vs Validation_accuracy of VGG16

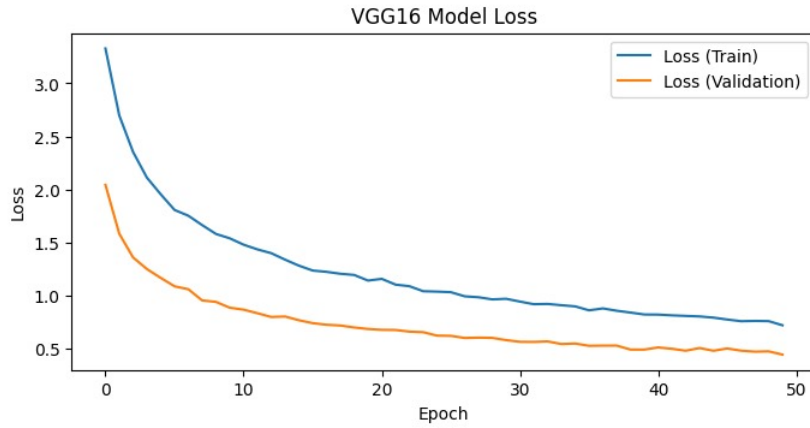


Figure 5.16: Training_loss Vs Validation_loss of VGG16

5.1.9 MobileNetV2

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8021 and validation accuracy is 0.8152. Figures 4.22 and 4.23 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

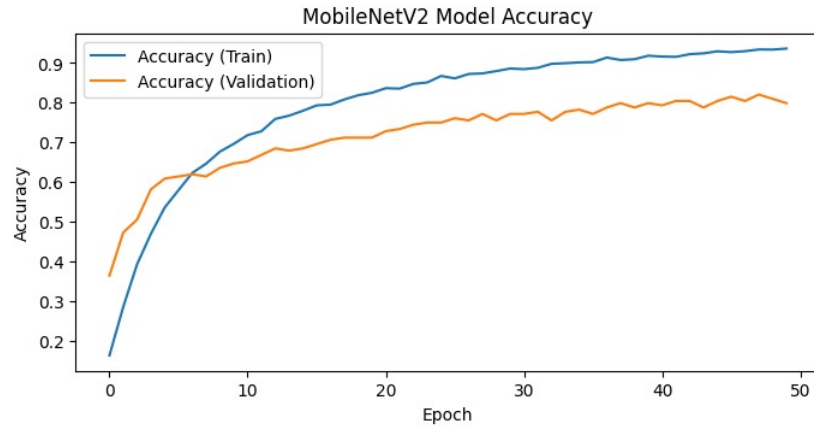


Figure 5.17: Training_accuracy Vs Validation_accuracy of MobileNetV2

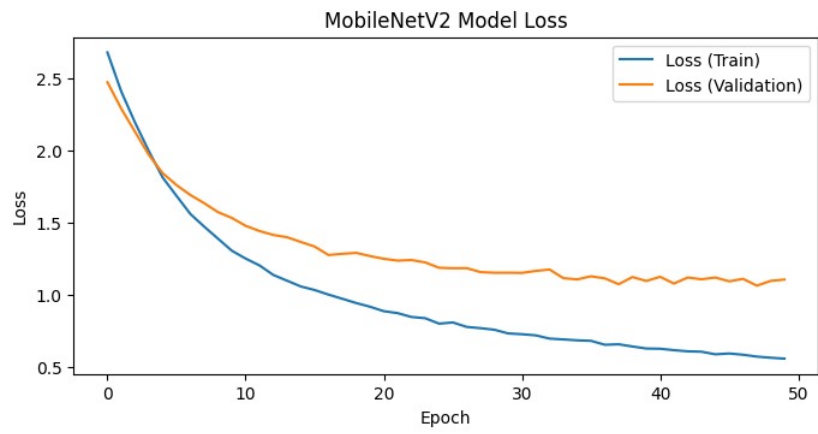


Figure 5.18: Training_loss Vs Validation_loss of MobileNetV2

5.1.10 InceptionV3

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8823 and validation accuracy is 0.8533. Figures 4.24 and 4.25 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

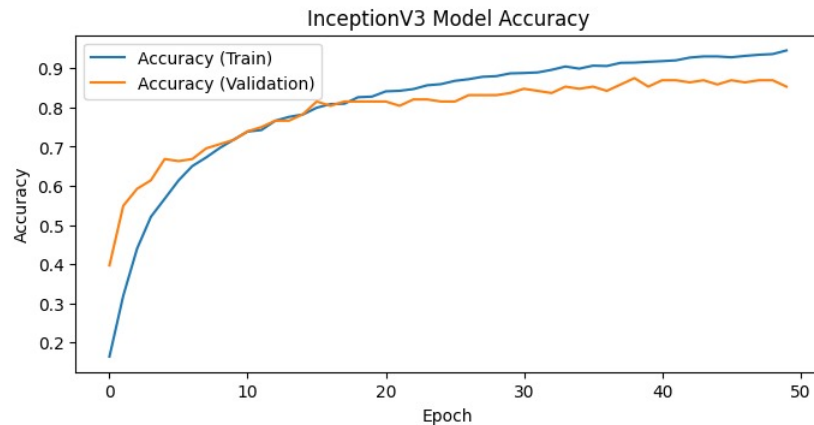


Figure 5.19: Training_accuracy Vs Validation_accuracy of InceptionV3

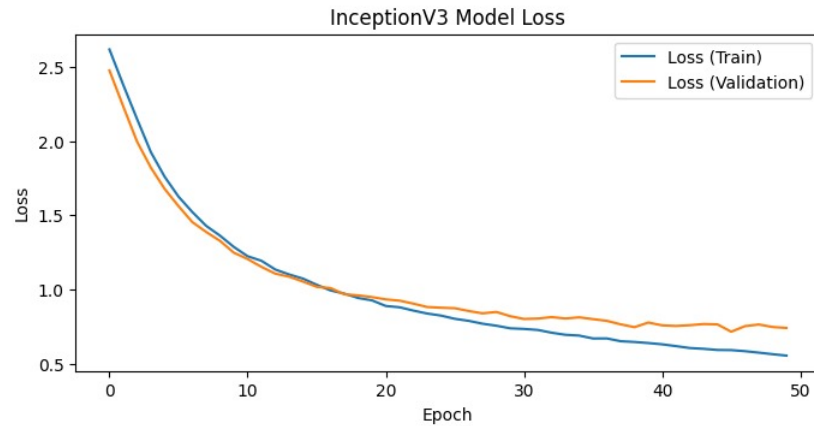


Figure 5.20: Training_loss Vs Validation_loss of InceptionV3

5.1.11 DenseNet169

We train this model by providing the train and validation dataset. The batch size is 8. So, steps per epoch = number of training samples/batch size. Which means that the training process will iterate a number of steps that is equal to one eight of the total number of samples in the training dataset during each epoch. The training will be executed for 50 epochs as we set epochs = 50. We also use callbacks. Which will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph.

will monitor the validation accuracy and save the model which is responsible for maximum validation accuracy. We also store the training history in a variable and visualize it on a graph. This model's test accuracy is 0.8342 and validation accuracy is 0.8696. Figures 4.26 and 4.27 represent two graphs one is training accuracy Vs validation accuracy and the other one is training loss Vs validation loss.

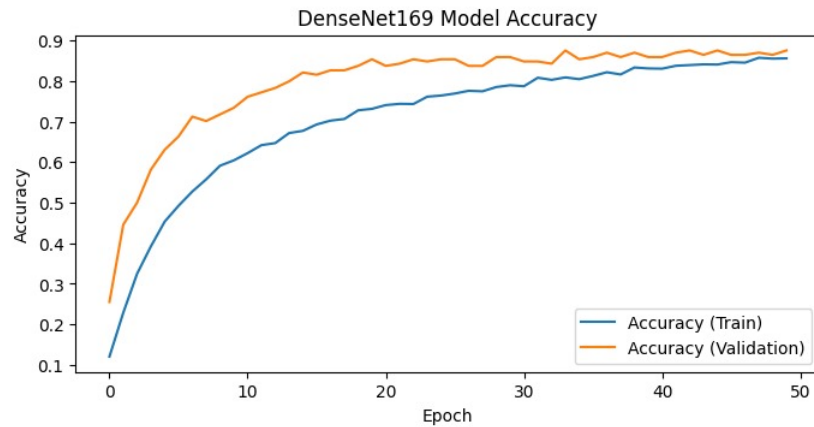


Figure 5.21: Training_accuracy Vs Validation_accuracy of DenseNet169

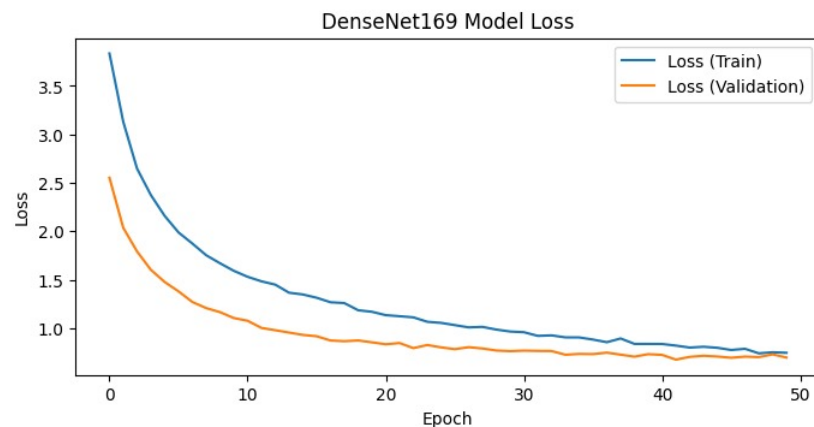


Figure 5.22: Training_loss Vs Validation_loss of DenseNet169

5.2 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.

5.2.1 Model Architectures and Performance Metrics

ConvolutionNet-5: ConvolutionNet-5 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: ConvolutionNet-4 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: ConvolutionNet-4M1 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: ConvolutionNet-5M1 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: ConvolutionNet-5M0 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 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: ConvolutionNet-6 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.

5.2.2 Analysis

Figure 5.1 and Figure 5.2 results are indicating that the architectural differences significantly impact the models' performance and accuracy. By achieving the highest test and validation accuracies with relatively low loss values, ConvolutionNet-5 will be considered as the best-performing model. Less spiking in the loss graphs reflects its stability that also suggests its architecture effectively balancing complexity and generalization. This architecture's robustness is highlighted by its consistent high precision, recall, and F1-scores across all classes.

With fewer layers and a lower dropout rate ConvolutionNet-3 has demonstrated decent performance but falls short in comparison with more complicated models, most importantly in terms of precision and recall for certain classes like Powdery Mildew Leaf. This result is completely suggesting that with a fewer convolutional layers and a lower dropout rate might not be able to capture the necessary features for accurate and efficient classification.

ConvolutionNet-4 and ConvolutionNet-4M1 are showing more improved and impressive performance compared to ConvolutionNet-3 that highlights the positive impact of having additional convolutional layers and higher dropout rates on model generalization. However, despite of having mixed filter sizes ConvolutionNet-4M1's architecture could not outplayed ConvolutionNet-4 significantly, indicating that the distribution of filters across layers which plays a critical role in model effectiveness and efficiency. Despite of achieving high accuracies ConvolutionNet-5M1 is presenting high loss values, indicating at potential issues with overfitting or lack of suitable optimizer. The using of Adam optimizer and higher batch size might focus to these issues, suggesting that by doing these changes, can improve certain metrics, that might also reflect the instability in model training.

Though ConvolutionNet-5M0 and ConvolutionNet-6 both are having more complex architectures but showing higher variance and instability. The increase of convolutional layers and filters does not necessarily provide better performance and might lead to overfitting, as evidenced from the spiky loss graphs. These models are showing that an overly complex architecture can hugely impact model stability and generalization that decreases the overall performance.

Finally, our study underscores the importance of balancing model complexity with generalization capability. So, it can be understandable that adding layers and increasing dropout rates can improve the model's performance. Besides, it is very

important to carefully tune these hyperparameters to avoid overfitting and instability. ConvolutionNet-5, with its balanced architecture, serves as a unique example of achieving high performance and efficiency through optimal 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

Figure 5.23: 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

Figure 5.24: Ablation Study Summary Table 2

5.3 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 table 5.3.

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)
Anthracoese_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

Figure 5.25: Classification Report of our custom CNN model

This table 5.4 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

Figure 5.26: Classification Report of our custom CNN model

Based on this Figure 5.4,

- 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.

5.4 Class-wise study for the confusion matrix

Class-wise study for the confusion matrix A confusion matrix evaluates the performance of a machine learning model on test data by displaying the number of

correct and incorrect predictions. It is particularly useful for classification models and breaks down predictions into four categories:

- TP (True Positive): The model correctly predicts positive instances as positive. For example, predicting an apple when it is actually an apple.
- TN (True Negative): The model correctly predicts negative instances as negative. For example, predicting not an apple when it is not an apple.
- FP (False Positive): The model incorrectly predicts negative instances as positive. For example, predicting an apple when it is not an apple.
- FN (False Negative): The model incorrectly predicts positive instances as negative. For example, predicting not an apple when it is actually an apple.

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:

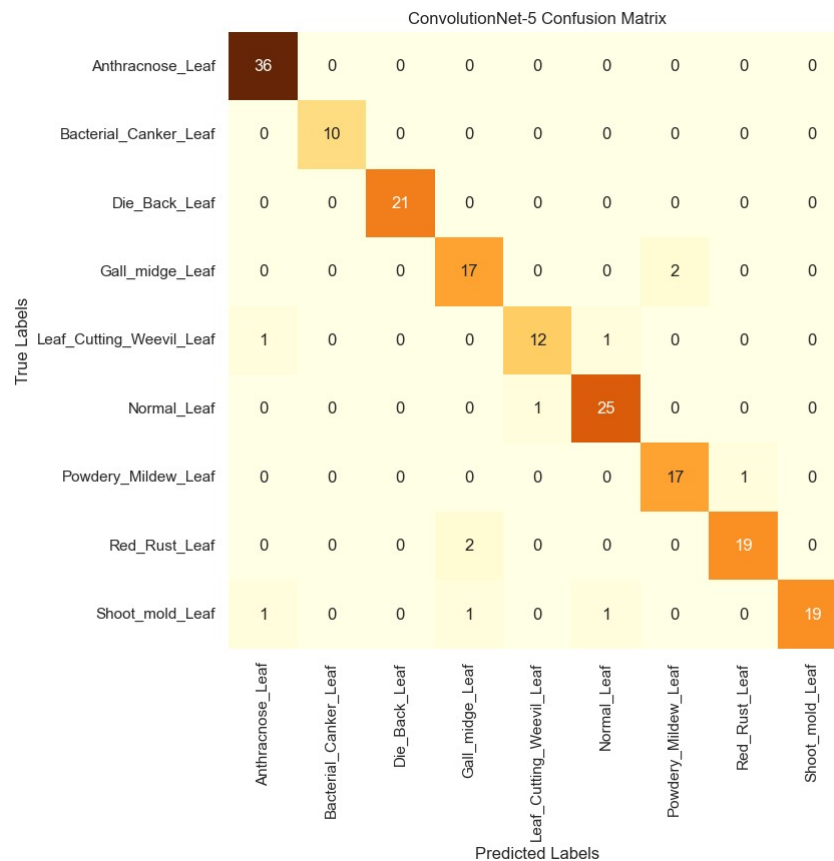


Figure 5.27: Confusion Matrix of ConvolutionNet-5

Let's explain the values in the confusion matrix. Vertically, we have 'Actual Class', and horizontally, 'Predicted Class'. In Figure 5.5, the first row shows that there are 36 instances of Anthracnose_Leaf. In the first column, the model correctly predicted all 36 instances of Anthracnose_Leaf and misclassified 1 instance of Shoot_mold_Leaf as Anthracnose_Leaf.

The second row indicates 10 instances of Bacterial_Canker_Leaf. In the second column, the model correctly predicted all 10 instances of Bacterial_Canker_Leaf.

The third row shows 21 instances of Die_Back_Leaf. In the third column, the model correctly predicted all 21 instances of Die_Back_Leaf without any errors.

Moving to the fourth row, there are 19 instances of Gall_midge_Leaf. $17+2 = 19$. In the fourth column, the model correctly predicted 17 instances of Gall_midge_Leaf and misclassified 2 instances of Red_Rust_Leaf and 1 instance of Shoot_mold_Leaf as Gall_midge_Leaf.

The fifth row indicates that 14 instances of Leaf_Cutting_Weevil_Leaf were found. $12+2 = 14$. On the other hand, column five indicates that the model had 12 correctly predicted cases of class Leaf_Cutting_Weevil_Leaf and misclassified 1 instance of Normal_Leaf as Leaf_Cutting_Weevil_Leaf.

Sixth row 26 examples of Normal_Leaf. $25+1 = 26$. Sixth column 25 examples of Normal_Leaf were correctly predicted, while 1 example each of Leaf_Cutting_Weevil_Leaf and Shoot_mold_Leaf was misclassified as Normal_Leaf.

The seventh row shows 18 examples of Powdery_Mildew_Leaf. $17+1 = 18$. The seventh column has rightly predicted 17 instances of Powdery_Mildew_Leaf and misclassified 2 instances of Gall_midge_Leaf as Powdery_Mildew_Leaf.

Eighth row contains 21 test examples of class Red_Rust_Leaf, $19+2 = 21$. In column eight, the model was found to have correctly predicted 19 examples from the class Red_Rust_Leaf and misclassified 1 of the Powdery_Mildew_Leaf as Red_Rust_Leaf.

Finally, the ninth row displays 22 instances of Shoot_mold_Leaf. $19+1+1+1 = 22$. In the ninth column, the model correctly predicted 19 instances of Shoot_mold_Leaf.

In summary, all diagonal values in this confusion matrix represent correct predictions, while off-diagonal values indicate misclassification.

VGG16:

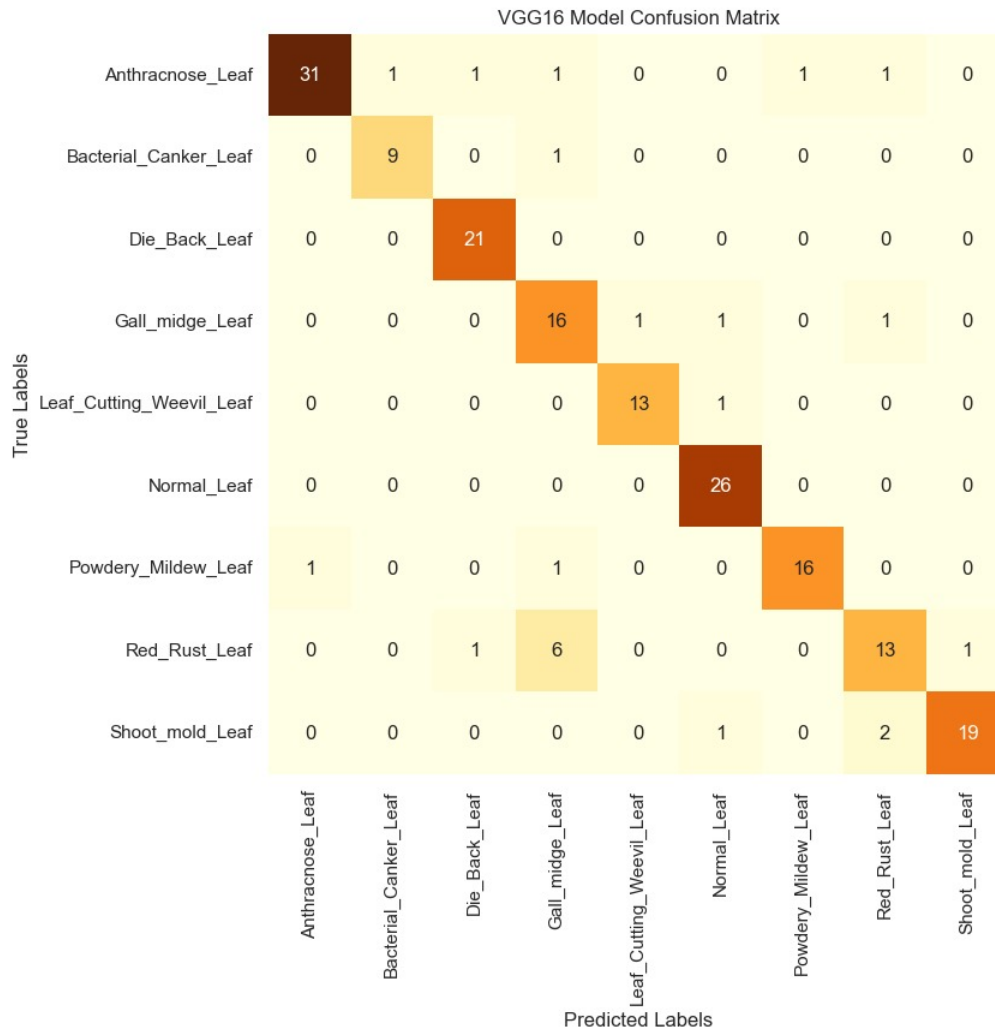


Figure 5.28: Confusion Matrix of VGG16

Likewise, in this confusion matrix (Figure 5.6), 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: 1,1,1,1,1,1,1,1,1,1,1,6,1,1,2 (Row-wise-All the bottle white boxes)

MobileNet V2:

MobileNetV2 Model Confusion Matrix

True Labels	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
Predicted Labels	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

Figure 5.29: Confusion Matrix of MobileNetV2

Similarly, in this confusion matrix (Figure 5.7), 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

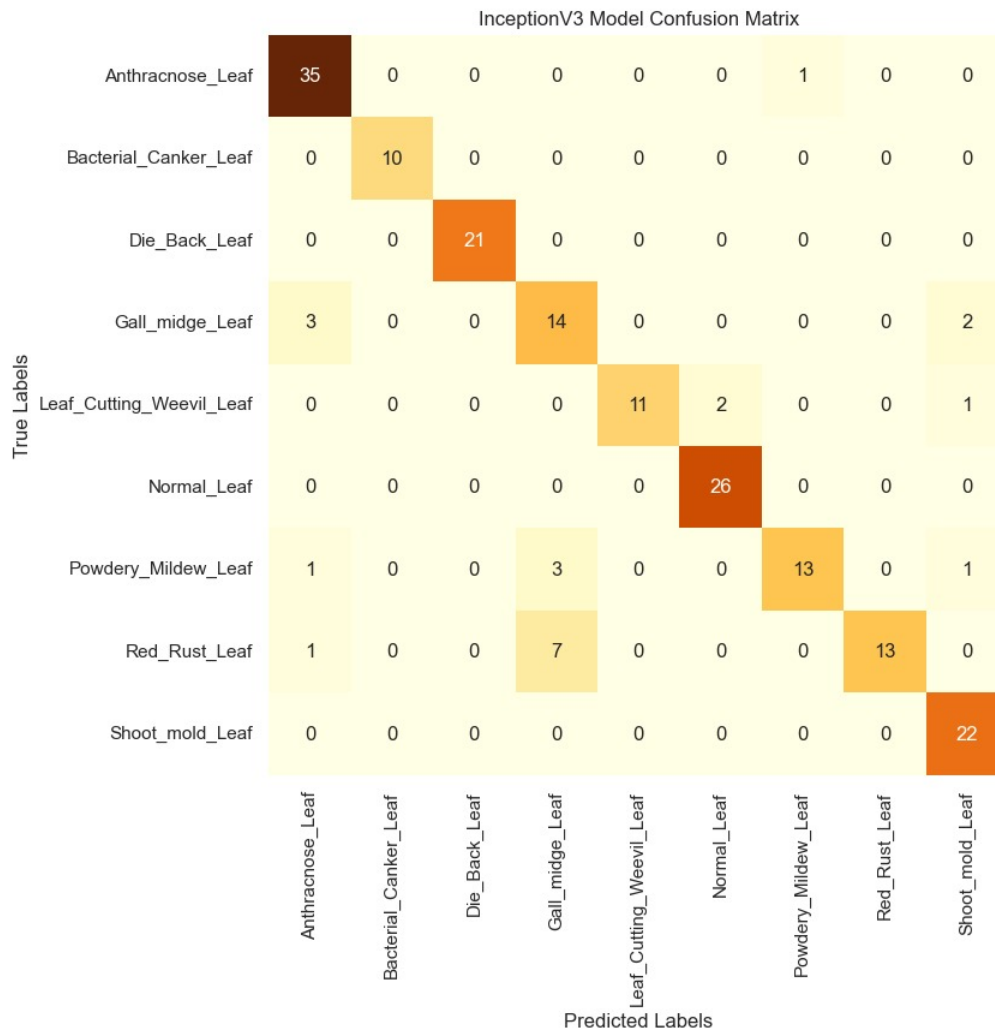


Figure 5.30: Confusion Matrix of InceptionV3

Moreover, in this confusion matrix (Figure 5.8), 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

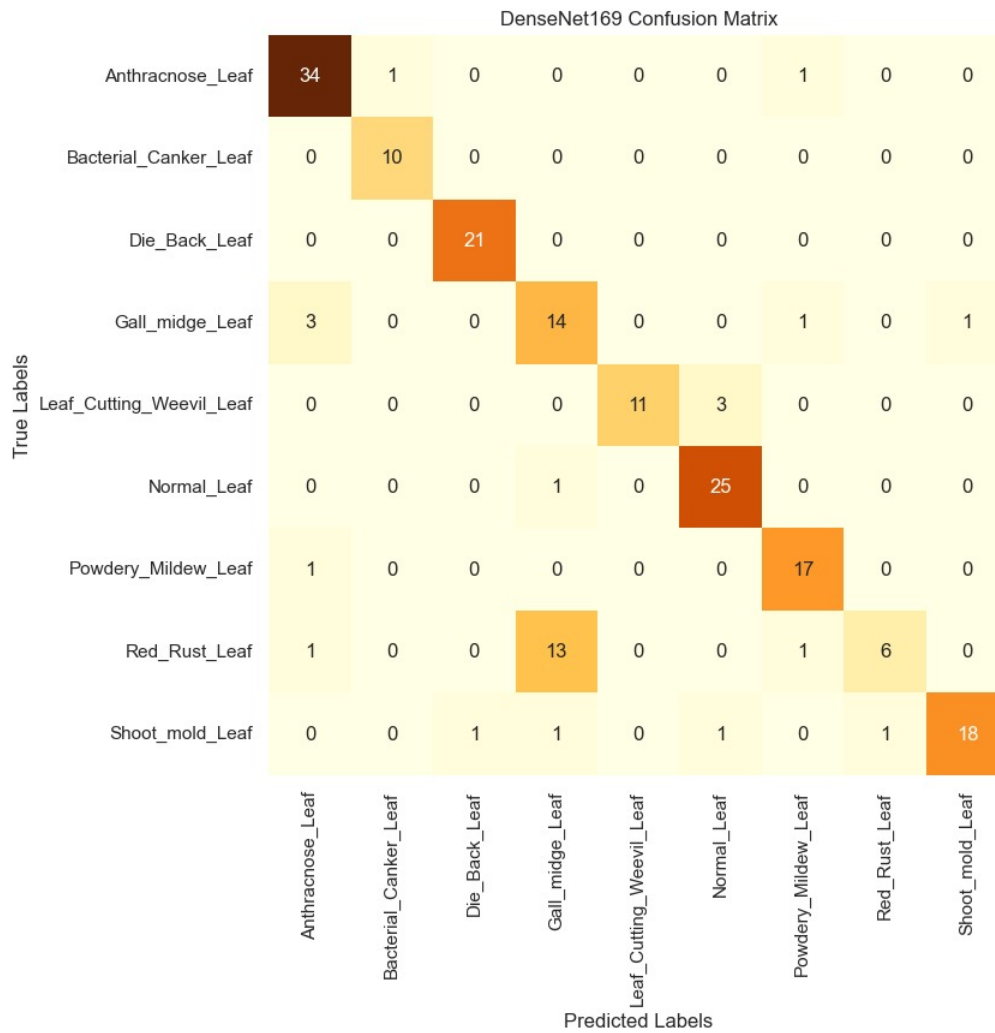


Figure 5.31: Confusion Matrix of DenseNet169

However, in this confusion matrix (Figure 5.9), 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: 1,1,3,1,1,3,1,1,1,1,13,1,1,1,1,1 (Row-wise-All the bottle white boxes)

5.5 Output

5.5.1 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. The formulas we used to calculate these metrics are as follows:

1. Accuracy = $TP / (TP + FP + FN + TN)$
2. Precision = $TP / (TP + FP)$
3. Recall = $TP / (TP + FN)$
4. F1 Score = $2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$

Here, TP stands for True Positive, TN for True Negative, FN for False Negative, and FP for False Positive.

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 5.10 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)

Figure 5.32: The Summary Of Our Proposed Model ConvolutionNet-5

5.5.2 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

Figure 5.33: Key Metrics Comparison of five different models

The comparison Figure 5.11 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.

In summary, each model has distinct characteristics: ConvolutionNet-5 excels in both inference speed and accuracy, MobileNetV2 in storage efficiency, DenseNet169 in depth and capacity, InceptionV3 in complexity, and VGG16 in balancing speed and size.

5.6 Visual Representation and Analysis of the Result Implementation:

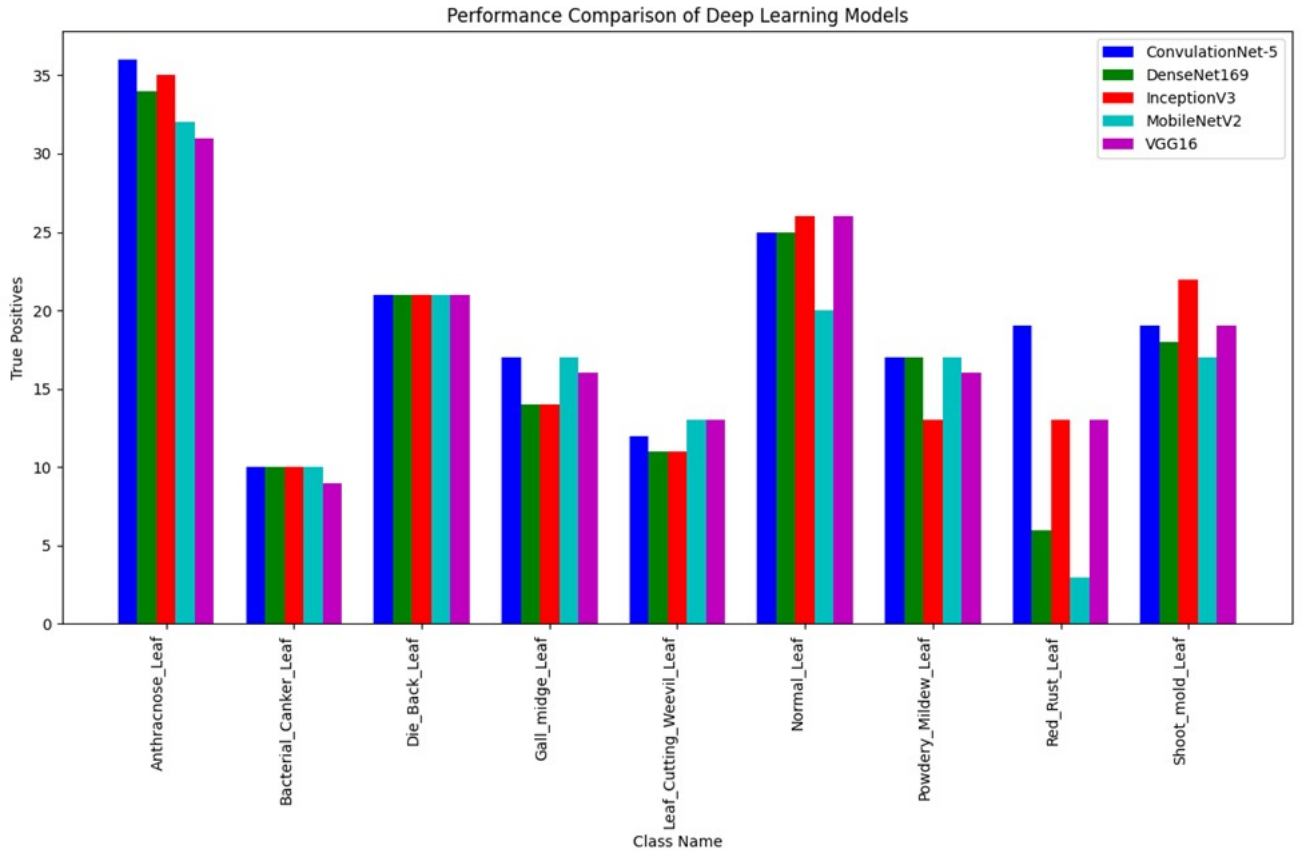


Figure 5.34: Performance Comparison of Deep Learning Models

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

Figure 5.35: Number of True Prediction of Deep Learning Models

The Figure 5.12 and Figure 5.13 provides a comparative analysis of the performance of five deep learning models (ConvolutionNet-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 ConvolutionNet-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: ConvolutionNet-5 and MobileNetV2 perform better (89-94%), while DenseNet169, InceptionV3, and VGG16 are slightly lower (74-84%).
- Leaf_Cutting_Weevil_Leaf: ConvolutionNet-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 ConvolutionNet-5, DenseNet169, MobileNetV2, and VGG16 (61-94%).
- Red_Rust_Leaf: ConvolutionNet-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, ConvolutionNet-5, and MobileNetV2 (77-86%).

2. Model-specific Observations:

- ConvolutionNet-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.

Chapter 6

Conclusion

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 because of their better-optimized architecture and parameters when utilized for identifying mango leaf disease. Although, in general, picture classification activities and tasks, the pre-trained models mentioned above are more dependable and effective, the models established by us are also very adaptable to the problems related to the mango leaf images because of the unique alterations and optimization and independent changes we had to complete in the model training performed by us which further contribute to enhancing the precision and reliability of the disease detection process.

Overall, our custom models also showed significant potential for practical application in real-world agricultural settings. By providing farmers with a reliable tool for early disease detection, our models can help mitigate the spread of diseases, reduce crop loss, and improve overall productivity.

6.1 Future Work

Despite the encouraging outcomes of our customized CNN models, there are still a number of areas for improvement:

- **Model Optimization:** By further optimizing the models to lower computational complexity, real-time field applications will be made possible and the model will be more appropriate for deployment on mobile and edge devices.
- **Dataset Expansion:** Adding more photographs from a different location and environmental circumstances into the dataset will further improve the robust-

ness of the models. More classification of illnesses and stages of infection will be able to increase the ability to diagnose.

- **IoT Integration:** By integrating the CNN models with Internet of Things (IoT) devices, continuous monitoring of crops is made possible, giving farmers access to real-time data collection and instant response.
- **User-Friendly Interfaces:** Making easy-to-navigate online or mobile applications can democratize the use of technology and make it intuitive; it really opens up when offering suggestions and insights derived from model output.
- **Cross-Crop Generalisability:** The need to further extend the model into general applicability that predicts well for various crops and their diseases substantiates the model to cover a wider region of agricultural diagnostics.

Bibliography

- [1] S. C. Nelson, *Mango powdery mildew*, 2008.
- [2] S. Nelson, *Mango powdery mildew caused by oidium mangiferae*, Retrieved from: <https://www.flickr.com/photos/scotnelson/9808512885>, Sep. 2013.
- [3] N. Rusk, “Deep learning,” *Nature Methods*, vol. 13, no. 1, pp. 35–35, 2016.
- [4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.
- [5] L. Perez and J. Wang, *The effectiveness of data augmentation in image classification using deep learning*, arXiv preprint arXiv:1712.04621, 2017.
- [6] J. Shijie, W. Ping, J. Peiyi, and H. Siping, “Research on data augmentation for image classification based on convolution neural networks,” in *2017 Chinese Automation Congress (CAC)*, IEEE, 2017, pp. 4165–4170.
- [7] 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.
- [8] J. G. A. Barbedo, “Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification,” *Computers and Electronics in Agriculture*, vol. 153, pp. 46–53, 2018.
- [9] Z. Iqbal, M. A. Khan, M. Sharif, J. H. Shah, M. H. ur Rehman, and K. Javed, “An automated detection and classification of citrus plant diseases using image processing techniques: A review,” *Computers and Electronics in Agriculture*, vol. 153, pp. 12–32, 2018.
- [10] R. G. D. Luna, E. P. Dadios, and A. A. Bandala, “Automated image capturing system for deep learning-based tomato plant leaf disease detection and recognition,” in *TENCON 2018-2018 IEEE Region 10 Conference*, IEEE, Oct. 2018, pp. 1414–1419.
- [11] L. D. Nguyen, D. Lin, Z. Lin, and J. Cao, “Deep cnns for microscopic image classification by exploiting transfer learning and feature concatenation,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2018, pp. 1–5.
- [12] S. Ramesh, R. Hebbar, M. Niveditha, R. Pooja, N. Shashank, and P. V. Vinod, “Plant disease detection using machine learning,” in *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*, IEEE, Apr. 2018, pp. 41–45.

- [13] K. Srunitha and D. Bharathi, "Mango leaf unhealthy region detection and classification," in *Computational Vision and Bio Inspired Computing*, Springer, 2018, pp. 422–436.
- [14] N. Sutrodhor, M. Hussein, M. Mridha, P. Karmokar, and T. Nur, "Mango leaf ailment detection using neural network ensemble and support vector machine," *International Journal of Computer Applications*, vol. 181, pp. 31–36, 2018.
- [15] 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.
- [16] T. Binoy and K. Lakshmi, "Comparative analysis of vehicle make and model recognition using deep learning techniques," in *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, vol. 1, IEEE, 2019, pp. 1298–1305.
- [17] G. Geetharamani and A. Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers & Electrical Engineering*, vol. 76, pp. 323–338, 2019.
- [18] M. S. Gulavnai and M. R. Patil, "Deep learning for image based mango leaf disease detection," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. X, no. X, Oct. 2019, Retrieved from: https://www.researchgate.net/publication/362536693_Deep_Learning_for_Image_Based_Mango_Leaf_Disease_Detection.
- [19] S. Malao, P. Gaikwad, P. Palve, R. Suryawanshi, and N. Suthar, *Disease diagnosis of mango leaf*, 2019.
- [20] 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.
- [21] 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.
- [22] S. Veling, "Mango disease detection by using image processing," *International Journal for Research in Applied Science and Engineering Technology*, vol. 7, pp. 3717–3726, Apr. 2019.
- [23] R. Karthik, M. Hariharan, S. Anand, P. Mathikshara, A. Johnson, and R. . Menaka, "Attention embedded residual cnn for disease detection in tomato leaves," *Applied Soft Computing*, vol. 86, p. 105 933, 2020.
- [24] M. R. Mia, S. Roy, S. K. Das, and M. A. Rahman, "Mango leaf disease recognition using neural network and support vector machine," *Iran Journal of Computer Science*, vol. 3, pp. 185–193, 2020.
- [25] T. N. Pham, L. V. Tran, and S. V. T. Dao, "Early disease classification of mango leaves using feed-forward neural network and hybrid metaheuristic feature selection," *IEEE Access*, vol. 8, pp. 189 960–189 973, 2020.
- [26] J. Lu, L. Tan, and H. Jiang, "Review on convolutional neural network (cnn) applied to plant leaf disease classification," *Agriculture*, vol. 11, no. 8, p. 707, 2021.

- [27] 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.
- [28] U. S. Rao, R. Swathi, V. Sanjana, *et al.*, "Deep learning precision farming: Grapes and mango leaf disease detection by transfer learning," *Global Transitions Proceedings*, vol. 2, no. 2, pp. 535–544, 2021, International Conference on Computing System and its Applications (ICCSA- 2021), ISSN: 2666-285X. DOI: <https://doi.org/10.1016/j.gltp.2021.08.002>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2666285X21000303>.
- [29] e. a. Saleem Rabia, *Mango leaf disease identification using fully resolution convolutional network*, Retrieved from: https://www.researchgate.net/publication/354145514_Mango_Leaf_Disease_Identification_Using_Fully_Resolution_Convolutional_Network, <https://doi.org/10.32604/cmc.2021.017700>, Jan. 2021.
- [30] S. Wongsila, P. Chantrasri, and P. Sureephong, "Machine learning algorithm development for detection of mango infected by anthracnose disease," in *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*, IEEE, 2021, pp. 249–252.
- [31] M. Prabu and B. J. Chelliah, "Mango leaf disease identification and classification using a cnn architecture optimized by crossover-based levy flight distribution algorithm," *Neural Computing and Applications*, vol. 34, no. 9, pp. 7311–7324, 2022.
- [32] S. Sandhya, A. Balasundaram, and S. Arunkumar, "Deep learning and computer vision based model for detection of diseased mango leaves," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 10, no. 6, pp. 70–79, 2022.
- [33] A. Sharma, R. K. Bijral, J. Manhas, and V. Sharma, "Mango leaf diseases detection using deep learning," *International Journal of Knowledge Based Computer Systems*, vol. 10, no. 1, 2022.
- [34] S. I. Ahmed, M. Ibrahim, and e. a. M. Nadim, "Mangoleafbd: A comprehensive image dataset to classify diseased and healthy mango leaves," *Data in Brief*, vol. 47, p. 108941, 2023.
- [35] 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.
- [36] C. Vijay and K. Pushpalatha, "Dv-pso-net: A novel deep mutual learning model with heuristic search using particle swarm optimization for mango leaf disease detection," *Journal of Integrated Science and Technology*, vol. 12, no. 5, pp. 804–804, 2024.