A PROJECT REPORT ON

" A COMPARATIVE STUDY OF VARIOUS IMAGE CLASSIFICATION MODELS AND THEIR INTERPRET-ABILITY USING THE PLANT VILLAGE DATASET "

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF ENGINEERING (B.E.)
IN
COMPUTER SCIENCE AND ENGINEERING
(2020-2024)

Submitted By:

Devraj Thokchom
MU Roll No: **202009**

Deedal Wahengbam
MU Roll No: **202013**

Lenin Khangjrakpam
MU Roll No: **204036**

Under the Guidance of:

Dr. Th Ibungomacha Singh
Head of Department, CSE
Manipur Institute of Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(A Constituent College of Manipur University)
MANIPUR INSTITUTE OF TECHNOLOGY
CANCHIPUR, MANIPUR, 795004

A COMPARATIVE STUDY OF VARIOUS IMAGE CLASSIFICATION MODELS AND THEIR INTERPRET-ABILITY USING THE PLANT VILLAGE DATASET

Devraj Thokchom[1] Deedal Wahengbam[2] Lenin Khangjrakpam[3]

August 2024

[1]Manipur Institute of Technology (MIT) and devrajthok@gmail.com
[2]Manipur Institute of Technology (MIT) and deedalwahengbam005@gmail.com
[3]Manipur Institute of Technology (MIT) and leninkhangjrakpam@gmail.com

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MANIPUR INSTITUTE OF TECHNOLOGY
Canchipur, IMPHAL, 795004

CERTIFICATE

This is to certify that the project entitled ***"A Comparative Study of Various Image Classification Models and their Interpret-Ability using the Plant Village Dataset"*** *is a genuine work done by*

Mr. Devraj Thokchom*, MU Roll No:* ***202009***
Mr. Deedal Wahengbam*, MU Roll No:* ***202013***
Mr. Lenin Khangjrakpam*, MU Roll No:* ***204036***

students of ***B.E.*** *from* ***Manipur Institute of Technology*** *under my guidance and supervision. This project has not been submitted to any other examination and does not form part of any other project undertaken by the candidate.*

I hereby forward this project work for the partial fulfillment of the requirements for the award of the degree of ***Bachelor of Engineering in Computer Science and Engineering, 2024.***

I wish them all success in life.

Head of Department & Internal Project Guide

(Th. Ibungomacha Singh)
Department of Computer Science & Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MANIPUR INSTITUTE OF TECHNOLOGY
Canchipur, IMPHAL, 795004

CERTIFICATE OF APPROVAL

This is to certify that the project entitled ***"A Comparative Study of Various Image Classification Models and their Interpret-Ability using the Plant Village Dataset"*** *is a genuine work done by*

Mr. Devraj Thokchom*, MU Roll No:* ***202009***
Mr. Deedal Wahengbam*, MU Roll No:* ***202013***
Mr. Lenin Khangjrakpam*, MU Roll No:* ***204036***

Further certified that no part of this project constitutes any part of a book or a journal already published elsewhere for the award of any other Degree of Bachelor of Engineering Department of the Institute.

I wish them all success in their academic endeavor.

Date.....
Place.....

Signature of Examiner

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MANIPUR INSTITUTE OF TECHNOLOGY
Canchipur, IMPHAL, 795004

UNDERTAKING

We declare that the work presented in this project titled ***"A Comparative Study of Various Image Classification Models and their Interpret-Ability using the Plant Village Dataset"*** *submitted to the Department of Computer Science & Engineering, Manipur Institute of Technology, for the fulfilment of the requirement of Bachelor of Engineering in Computer Science & Engineering, is our original work. We have not plagiarized or submitted the same work for the award of any other degree. When we used external resources, we commit to properly citing all sources of such resources in our project to give due credit to the original authors. Any external content used has been appropriately referenced in accordance with academic standards. In case this understanding is found incorrect, we accept that our degree may be unconditionally withdrawn.*

Mr. Devraj Thokchom*, MU Roll No:* ***202009***
Mr. Deedal Wahengbam*, MU Roll No:* ***202013***
Mr. Lenin Khangjrakpam*, MU Roll No:* ***204036***

Head of Department & Internal Project Guide

(Th. Ibungomacha Singh)
Department of Computer Science & Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MANIPUR INSTITUTE OF TECHNOLOGY
Canchipur, IMPHAL, 795004

DECLARATION

We, ***Devraj Thochom (202009), Deedal Wahengbam (202013)*** *and* ***Lenin Khangjrakpam (204036)****, students of Bachelor of Engineering in Computer Science and Engineering of Manipur Institute of Technology, hereby declare that the project entitled* ***"A Comparative Study of Various Image Classification Models and their Interpret-ability using the Plant Village dataset"*** *has been carried out independently at the Institute under the guidance of* ***Professor Th Ibungomacha****, Head of Department (HOD) of Computer Science and Engineering.*

We hereby declare that the work submitted in the thesis is our own, except where acknowledged in the text and has not been previously submitted for the award of the degree of Manipur Institute of Technology.

Date.....
Place.....

Deedal Wahengbam
Devraj Thokchom
Lenin Khangjrakpam

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MANIPUR INSTITUTE OF TECHNOLOGY
Canchipur, IMPHAL, 795004

ACKNOWLEDGEMENT

First and Foremost, we would like to express our deepest gratitude to ***Professor Th Ibungomacha****, Head of Department (HOD) of Computer Science and Engineering, Manipur Institute of Technology for his valuable advice, guidance, and for providing us the opportunity to carry out our project entitled:* ***A Comparative Study of Various Image Classification Models and their Interpret-ability using the Plant Village dataset****.*

We are also thankful to ***Professor N Basanta Singh****, Principal, Manipur Institute of Technology, for providing us the opportunity to undergo our project in Manipur Institute of Technology.*

We express our deep sense of gratitude and thanks to all other staff members for their keen interest and help in our project development.

We are also thankful to our colleagues for their kind co-operation by giving us inspiration and encouragement throughout the development of this project.

Date.....
Place.....

Deedal Wahengbam
Devraj Thokchom
Lenin Khangjrakpam

ABSTRACT

In this study, we conduct a comprehensive comparative analysis of various image classification machine learning models using the Plant Village dataset, which we reduced to 11 distinct classes of plant diseases. Leveraging the power of transfer learning and fine-tuning, we evaluate three pre-trained CNN models—**VGG16, Xception, and MobileNetV2**—as base models. To further compare performance with nonneural network models, we utilize the pre-trained MobileNetV2 model as a feature extractor, training a **Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Random Forest and Extreme Gradient Boosting (XGBoost)** classifiers on the extracted features. Our findings reveal that neural network-based models, particularly those using transfer learning, significantly outperform traditional classifiers in terms of accuracy.

Additionally, we design and train a custom CNN model from scratch named ***CNN10L***, structured similarly to the VGG architecture, which, while achieving commendable accuracy, still falls short of the pre-trained models' performance.

To gain deeper insights into the model's behavior, we also perform an in-depth analysis of the fine tuned (on Plant Village Dataset) VGG16-based model by visualizing its convolutional layers and interpreting the learned features. Techniques such as image masking and activation mapping are employed to understand the model's focus areas and decision-making process.

To further interpret our CNN model, we also, perform visualization of the feature extracted by our model on a 2D and 3D feature space using **PCA and t-SNE**. We also used a **cosine similarity score** to find all the $\boldsymbol{N}$ data instances that are nearest neighbors to a given/query feature vector (extracted from image)

Our results underscore the efficacy of transfer learning in image classification tasks and provide a detailed comparison of different model architectures and classification approaches. This study not only highlights the superior performance of fine-tuned pre-trained models but also sheds light on the internal workings of CNNs through comprehensive visualization techniques.

Keywords:

Convolutional Neural Network, Transfer Learning, Fine Tuning, Plant Disease Classification, Image Classification, Plant Village Dataset, VGG16, MobileNetV2, Xception, CNN10L, CNN Model Interpretation, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Random Forest, Extreme Gradient Boosting (XGBoost), Feature Extraction, Model Performance Comparison, Cosine Similarity, PCA, t-SNE, Agricultural Technology

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Machine Learning is a field of Artificial Intelligence that focuses on developing algorithms and models, by learning from patterns in a dataset and using it to make predictions on new data. These algorithms can be very useful for identifying patterns and relationships in a given database, which can be invaluable to extract insights or make predictions without being explicitly programmed.

Machine Learning can be classified into various categories; supervised learning, unsupervised learning, and reinforcement learning being the most common ones. Supervised Learning Algorithms learn from labelled data, making predictions based on the given data. Unsupervised Learning Algorithms learn from unlabelled data, where the algorithm identifies patterns and structures within the data. Reinforcement Learning Algorithms learns through interacting with an environment, and aiming to maximize cumulative rewards.

Convolutional Neural Networks or CNNs are a subset of deep learning models especially useful for tasks involving processing of visual data. Its architecture is particularly inspired by the connectivity patterns of the human brain, especially the visual cortex. Its primary goal is to automatically learn the hierarchical representations of features directly from raw data.

CNNs use specialized architectures that incorporate convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply convolution operations to the input image, using learnable filters to extract features such as edges, textures, or shapes. Pooling layers then reduce the spatial dimensions of the feature maps generated by the convolutional layers, helping to decrease computational complexity and control overfitting. Finally, fully connected layers use the extracted features to classify the input image into various categories.

1.2 Problem Statement

Accurate and efficient detection of plant diseases is critical for ensuring food security and agricultural productivity. Traditional methods of plant disease identification often rely on expert knowledge and manual inspection, which can be time-consuming, labor-intensive, and prone to human error. With the advent of deep learning and advanced machine learning techniques, there is a growing potential to automate and enhance the accuracy of disease classification using image-based analysis.

The Plant Village dataset provides a robust foundation for developing machine learning models aimed at identifying various plant diseases from images. However, the challenge lies in selecting and optimizing the appropriate model architectures to achieve high classification accuracy. While Convolutional Neural Networks (CNNs) have shown great promise in image recognition tasks, there is a need to systematically compare their performance with traditional classifiers and evaluate the benefits of using transfer learning and fine-tuning techniques with pre-trained models.

This study aims to address the following key questions:

1. How do pre-trained models like VGG16, Xception, and MobileNetV2 perform in the context of plant disease classification when fine-tuned on the Plant Village dataset?

2. Can traditional classifiers such as SVM, KNN, Random Forest, and XGBoost, when used with features extracted from pre-trained models, achieve comparable performance to CNN-based models?

3. What is the effectiveness of a custom-designed CNN model in classifying plant diseases compared to fine-tuned pre-trained models?

4. How can we interpret and visualize the features learned by CNN models to gain insights into their decision-making processes?

By addressing these questions, this research aims to identify the most effective machine learning approaches for plant disease classification and provide a deeper understanding of the underlying mechanisms of CNN models through visualization techniques.

1.3 Scope of Study

The scope of this study encompasses a comprehensive evaluation of various image classification machine learning models applied to the Plant Village dataset, specifically focusing on the detection and classification of plant diseases. The key components of this study are as follows:

1. **Dataset Utilization:**
   The Plant Village dataset, consisting of images categorized into 11 different plant disease classes, is used to train and evaluate the models. This dataset provides a diverse and challenging testbed for assessing model performance in real-world scenarios.

2. **Model Selection and Training:**

   - Three pre-trained models, VGG16, Xception, and MobileNetV2, are employed as base models for transfer learning and fine-tuning. These models are chosen for their proven efficacy in image classification tasks and their ability to leverage pre-learned features from large datasets like ImageNet.
   - In addition to these neural network models, traditional machine learning classifiers such as SVM, KNN, Random Forest, and XGBoost are trained using features extracted from the fine-tuned MobileNetV2 model. This provides a comparative analysis of deep learning models against conventional machine learning approaches.

3. **Model Development:**
   A custom Convolutional Neural Network (CNN) with 10 layers is designed and trained from scratch, inspired by the VGG architecture. This model serves as a baseline to compare the performance of transfer learning models and to understand the effectiveness of custom architectures on the dataset.

4. **Model Interpretation and Analysis:**
   Various techniques are employed to interpret and analyze the models, including:

   - Visualization of the first layer filters and feature maps at different layers.
   - Comparative analysis of feature maps using normal and masked images.
   - Feature space visualization using PCA and t-SNE before and after fine-tuning.
   - Nearest neighbor search using cosine similarity to evaluate the feature representations.
   - Saliency Map Generation

5. **Deployment:**
   The best-performing model is deployed using Flask to serve as an API for real-time image classification. The frontend of the application is developed using React Native, enabling users to submit images and receive predictions.

6. **Performance Evaluation:**
   The models are evaluated based on their accuracy, confusion matrices, and interpretability. This includes a thorough comparison of the classification performance of neural network models and traditional classifiers.

This study aims to provide a detailed comparative analysis of different machine learning models for plant disease classification, highlighting the strengths and limitations of each approach. By leveraging transfer learning, custom model development, and comprehensive model interpretation techniques, this study contributes valuable insights into the application of machine learning in agricultural disease detection.

Chapter 2

Literature Review

In this chapter, we list all the papers, that we used to to perform our research work

2.1 Literature Review

1. Konstantinos P. Ferentinos (February 2018)
   Deep learning models for plant disease detection and diagnosis
   https://www.sciencedirect.com/science/article/abs/pii/S0168169917311742
   The paper develops neural network models to perform plant disease detection and diagnosis using simple leaves image of healthy and diseased leaves. Training used an open database of 87,848 images, containing 25 different plants, in a set of 58 different classes.Several model architectures were trained, with the best performance reaching a 99.53% success rate in reaching the correct diagnosis

2. Sharada Prasanna Mohanty, David Hughes, Marcel Salathé (2016)
   Using Deep Learning for Image-Based Plant Disease Detection
   https://arxiv.org/abs/1604.03169
   The paper presents an innovative approach to addressing the significant issue of crop disease diagnosis by leveraging deep learning technology. The authors successfully trained a deep convolutional neural network on a large dataset of plant leaf images, achieving a high accuracy of 99.35% in controlled conditions. However, the model's performance dropped to 31.4% when tested with images from different sources, highlighting the need for more diverse training data. This study demonstrates the importance of improving data diversity to enhance model generalizability for real-world applications.

3. Francois Chollet (2017)
   Xception: Deep Learning with Depthwise Separable Convolutions
   https://doi.org/XXXXXX
   The paper introduces an interpretation of Inception modules in CNN as an

intermediate step between regular convolution and depthwise separable convolution (a depthwise convolution followed by a pointwise convolution). By understanding depthwise separable convolution as an Inception module with a maximally large number of towers, the authors propose a new CNN architecture, Xception, which replaces Inception modules with depthwise separable convolutions.

4. Muhammad Hammad Saleem ,Johan Potgieter and Khalid Mahmood Arif (31 October 2019)
   Plant Disease Detection and Classification by Deep Learning
   https://www.mdpi.com/2223-7747/8/11/468
   This paper provides a comprehensive explanation of DL models used to visualize various plant diseases. In addition, some research gaps are identified from which to obtain greater transparency for detecting diseases in plants, even before their symptoms appear clearly.

5. Muhammad Shoaib Shaker El-sappagh,Babar Shah,Akhtar Ali,Asad Ullah,Fayadh Alenezi,Tsanko Gechev,Tariq Hussain and Farman Ali (2023)
   An advanced deep learning models-based plant disease detection: A review of recent research
   https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2023.1158933/full
   This paper addresses the challenges and limitations associated with using ML and DL for plant disease identification, such as issues with data availability, imaging quality, and the differentiation between healthy and diseased plants. The research provides valuable insights for plant disease detection researchers, practitioners, and industry professionals by offering solutions to these challenges and limitations, providing a comprehensive understanding of the current state of research in this field, highlighting the benefits and limitations of these methods, and proposing potential solutions to overcome the challenges of their implementation.

6. Edna Chebet Too , Li Yujian , Sam Njuki and Liu Yingchun(June 2019)
   A comparative study of fine-tuning deep learning models for plant disease identification
   https://www.sciencedirect.com/science/article/abs/pii/S0168169917313303
   This paper is a comparative study of state-of-the-art deep learning for plants disease detection using images of leaves. The results show that deeper models are not only accurate but have fewer number of parameters. DenseNet model also perform better than other models studied with no signs of overfitting and performance deterioration.

7. Murk Chohan, Adil Khan, Rozina Chohan, Saif Hassan Katpar and Muhammad Saleem Mahar (May 2020)

Plant Disease Detection using Deep Learning
https://www.researchgate.net/profile/Saif-Katper/publication/341025012_Plant_Disease_Detection_using_Deep_Learning/links/5eabf714299bf18b958a94a8/Plant-Disease-Detection-using-Deep-Learning.pdf
This paper proposes a deep learning based model named plant disease detector. The model is able to detect several diseases from plants using pictures of their leaves. Plant disease detection model is developed using neural network. First of all augmentation is applied on dataset to increase the sample size. Later Convolution Neural Network (CNN) is used with multiple convolution and pooling layers. PlantVillage dataset is used to train the model. After training the model, it is tested properly to validate the results.

8. Faye Mohameth, Chen Bingcai and Kane Amath Sada (June 2020)
Plant Disease Detection with Deep Learning and Feature Extraction Using Plant Village
https://www.scirp.org/journal/paperinformation?paperid=100958
This paper evaluates CNN's architectures applying transfer learning and deep feature extraction. All the features obtained will also be classified by SVM and KNN. Our work is feasible by the use of the open source Plant Village Dataset. The result obtained shows that SVM is the best classifier for leaf's diseases detection.

9. Rasim Alguliyev, Yadigar Imamverdiyev, Lyudmila Sukhostat and Ruslan Bayramov (01 September 2021)
Plant disease detection based on a deep model
https://link.springer.com/article/10.1007/s00500-021-06176-4
This paper proposes an accurate approach to identify plant leaf diseases based on the deep convolutional neural network and gated recurrent units. The proposed model is trained to identify common plant leaf diseases of 14 species. The experimental results are compared to other well-known models. This study shows that the proposed model based on deep learning provides the best solution in the diagnosis of plant diseases with high accuracy, and that the gated recurrent unit neural network considered as a classifier can improve the accuracy of the convolutional neural network model.

10. Shima Ramesh, Ramachandra Hebbar,Niveditha M.,Pooja R., Prasad Bhat N., Shashank N., Vinod P.V. (2018)
Plant Disease Detection Using Machine Learning
https://ieeexplore.ieee.org/abstract/document/8437085
This paper makes use of Random Forest in identifying between healthy and diseased leaf from the data sets created. Our proposed paper includes various phases of implementation namely dataset creation, feature extraction, training the classifier and classification. The created datasets of diseased and healthy leaves are collectively trained under Random Forest to classify the diseased and healthy images.

11. **Laha Ale, Alaa Sheta, Longzhuang Li, Ye Wang, Ning Zhang** (2019)
Deep Learning Based Plant Disease Detection for Smart Agriculture
https://ieeexplore.ieee.org/abstract/document/9024439
This paper proposes a Densely Connected Convolutional Networks (DenseNet) based transfer learning method to detect the plant diseases, which expects to run on edge servers with augmented computing resources. It then proposes a lightweight Deep Neural Networks (DNN) approach that can run on Internet of Things (IoT) devices with constrained resources.

Chapter 3

Theoretical Background

In this chapter, we provide a theoretical background, concept require by the reader to understand our research work.

3.1 Convolutional Neural Network (CNN)

Convolutional neural networks (CNN) are a regularized type of feed-forward neural network that learns feature engineering by itself via filters (or kernel) optimization. [10]

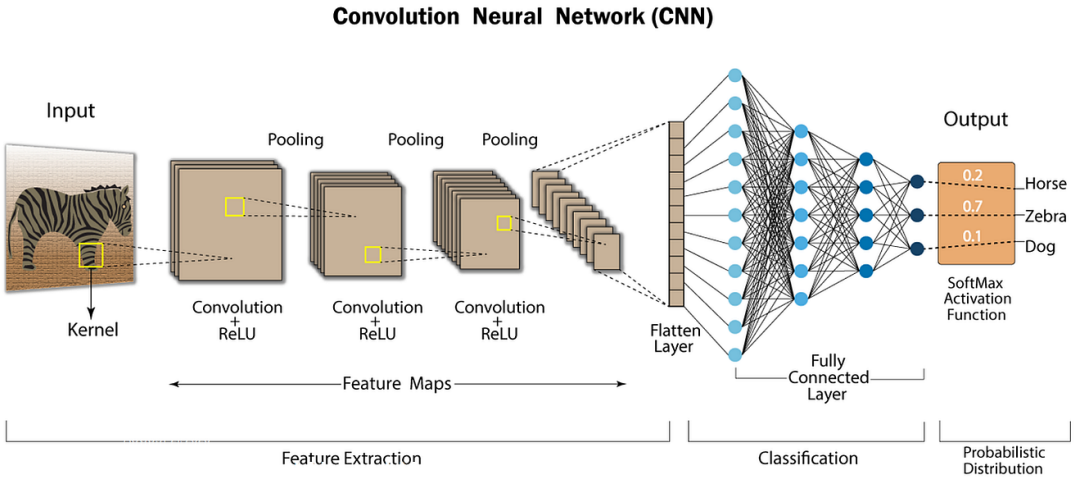

Figure 3.1: A typical ConvNet Classification Model, Source: [1]

3.1.1 Architecture of CNN

A convolutional neural network consists of an input layer, hidden layers and an output layer. In a convolutional neural network, the hidden layers include one or more layers that perform convolutions. Typically this includes a layer that performs a dot product of the convolution kernel with the layer's input matrix.

This product is usually the Frobenius inner product, and its activation function is commonly ReLU. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers. Here it should be noted how close a convolutional neural network is to a matched filter [11].

Convolution Layers

In a CNN, the input is a tensor with shape:

$$(\text{number of inputs}) * (\text{input height}) * (\text{input width}) * (\text{input channels})$$

After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape:

$$(\text{number of inputs}) * (\text{feature map height}) * (\text{feature map width}) * (\text{feature map channels})$$

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus [12]. Each convolutional neuron processes data only for its receptive field.

Pooling Layers

Convolutional networks may include local and/or global pooling layers along with traditional convolutional layers. Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, tiling sizes such as 2×2 are commonly used. Global pooling acts on all the neurons of the feature map.There are two common types of pooling in popular use: max and average. Max pooling uses the maximum value of each local cluster of neurons in the feature map, while average pooling takes the average value [13].

Fully Connected Layers

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is the same as a traditional multilayer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the image

Activation Function

Activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

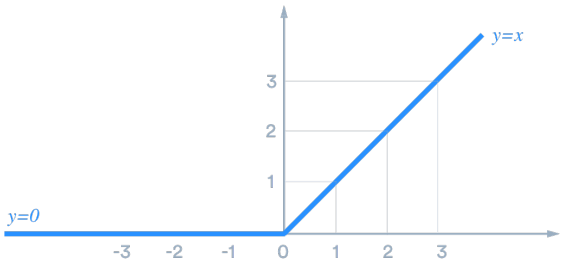

Figure 3.2: Rectified Linear Unit Activation Function (ReLU)

Rectified Linear Unit Activation Function (RELU)

ReLU applies the non-saturating activation function

$$f(z) = max(0, z)$$

It effectively removes negative values from an activation map by setting them to zero. Rectified Linear Unit (ReLU) or rectifier activation function introduces the property of nonlinearity to a deep learning model and solves the vanishing gradients issue. It interprets the positive part of its argument [14].

3.1.2 Backpropagation in CNN

Backpropagation is a process involved in training a neural network. It takes the error rate of a forward propagation and feeds this loss backward through the neural network layers to adjust the weights. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration.) Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

3.2 Transfer Learning and Fine Tuning

3.2.1 Transfer Learning

It is a technique in which knowledge learned from a related task is re-used to boost the perfomance on a related task. [15] It is mainly used when the data available for training is very small. Specifically, in ConvNet network, many image features such as lines, edges (seen in almost every image) are common to a diffrent image datasets. So, we don't, need to really train our model from scratch as we can used this pre-learned features on our new datasets. That is why large CNN models are rarely trained completely from scratch as large quality labelled datasets and heavy computational resources are hard to find and expensive.

For eg: in image classification tasks, knowledge gained from learning to recognize car can be used in recognizing trucks.

3.2.2 Fine Tuning

It is an approach to transfer learning in which the parameter of a pretrained model are trained on a new datasets [16]. Fine tuning can be done on the entire neural network layer or a subset of the layers, in this case, the layers that are not selected for fine tuning, have their weights frozen during back-propagation (i.e. these weights are not trained/changed) during fine tuning process). A model can also be augmented with adapters, which consists of a far fewer params than the original model, and fine-tuned in a parameter efficient way by adjusting the weights of the adapters models only and leaving the other weights frozen [17].

For architecture such as large, deep ConvNet models, its common to keep the lower layer frozens (i.e. layers that are closed to the input layers) as they capture low level features like detecting edges, lines, textures, etc. while the upper layers capture high-level features that are more related to the task in which the models is trained on [18]. More detailed visualisation of the feature learned by ConvNet model are present in chapter 7

Models that are pretrained on large , general corpora are usually fine tuned by reusing the model parameter as starting points and adding task specific layer from scratch. Fine-tuning the full model is common and often yield better results than training from scratch, however they are computationally expensive. [19] [20]

3.3 Machine Learning Classifier

Classifier is a type of machine learning algorithm used to assign a class label to a data input. Classifier algorithms are trained using labeled data; in the image recognition example, for instance, the classifier receives training data that labels images. After sufficient training, the classifier then can receive unlabeled images as inputs and will output classification labels for each image.

3.3.1 ConvNet Models

VGG16

VGG16 is a deep convolutional neural network model used for image classification tasks. The network is composed of 16 layers of artificial neurons, which each work to process image information incrementally and improve the accuracy of its predictions.

VGG16 uses convolution layers with a 3x3 filter and a stride 1 that are in the same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has two fully connected layers, followed by a softmax for output [21].

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Figure 3.3: VGG16 Architecture, Source: [2]

In VGG16, 'VGG' refers to the Visual Geometry Group of the University of Oxford, while the '16' refers to the network's 16 layers that have weights.

MobileNetV2

MobileNetV2, is a lightweight convolutional neural network (CNN) architecture, specifically designed for mobile and embedded vision applications. Google researchers developed it as an enhancement over the original MobileNet model. A remarkable aspect of this model is its ability to strike a good balance between model size and accuracy, rendering it ideal for resource-constrained devices [3].

MobileNetV2 architecture incorporates several key features that contribute to its efficiency and effectiveness in image classification tasks. These features include depthwise separable convolution, inverted residuals, bottleneck design, linear bottlenecks, and squeeze-and-excitation (SE) blocks [22]. Each of these features plays a crucial role in reducing the computational complexity of the model while maintaining high accuracy.

Xception

Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. It was developed by Google researchers. Google presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 3.4: MobileNetV2 Block Diagram, Source: [3]

depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution) [4].

In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads them to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions.

Xception is an efficient architecture that relies on Depthwise Separable Convolution and Shortcuts between Convolution blocks as in ResNet.

3.3.2 CNN10L - Scratch Model

CNN10L (short 10 Convolutional layer model) is ConvNet Model built by our team. It's composed of *10 convolutional layers, 3 max pooling layers, 2 Dense Layers, 1 Global Average 2D Pooling Layers and 1 Dropout Layer.* The Layer architecture is shown in figure 3.7. We used non-linear activation function ReLu in each of the output of convolutional layer.

The model is designed in such a way that there is balance in depth and width of feature learned. Our aim is to check whether such a shallow model can perform on par with famous very deep convnet model like VGG16, Xception, etc on narrow domain of plant disease detection tasks. Our main emphasis is also to reduce the computational power used by the model by developing a shallow model. And also by training our model from scratch and others by transfer learning, we can check

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 3.5: MobileNetV2 Architecture, Source: [3]

how features learned from larger datasets like **ImageNet** contribute significantly to improving performance and efficiency when training on narrower domain tasks.

A detailed description of the model with its input shape and output shape is also given in figure 3.8.

Key features of CNN10L Model:

1. **Multiple Convolution Layers Before Pooling**
   Feature Extraction: By stacking multiple convolutional layers before each pooling layer, the model can capture more detailed and abstract features at each stage. This allows for a richer and more complex feature representation.

2. **Hierarchical Feature Learning**
   Layered Learning: Early layers focus on simple features like edges, while deeper layers capture more complex patterns such as shapes and textures. This hierarchical approach enables the network to learn a comprehensive representation of the input data.

3. **Reduced Spatial Dimensions**
   Pooling Benefits: The max pooling layers help in reducing the spatial dimensions of the feature maps, decreasing computational load and preventing overfitting. This leads to a more efficient and effective model.

4. **Receptive Field Expansion**
   Context Awareness: Each pooling operation increases the receptive field of the subsequent layers, allowing them to incorporate more context from the

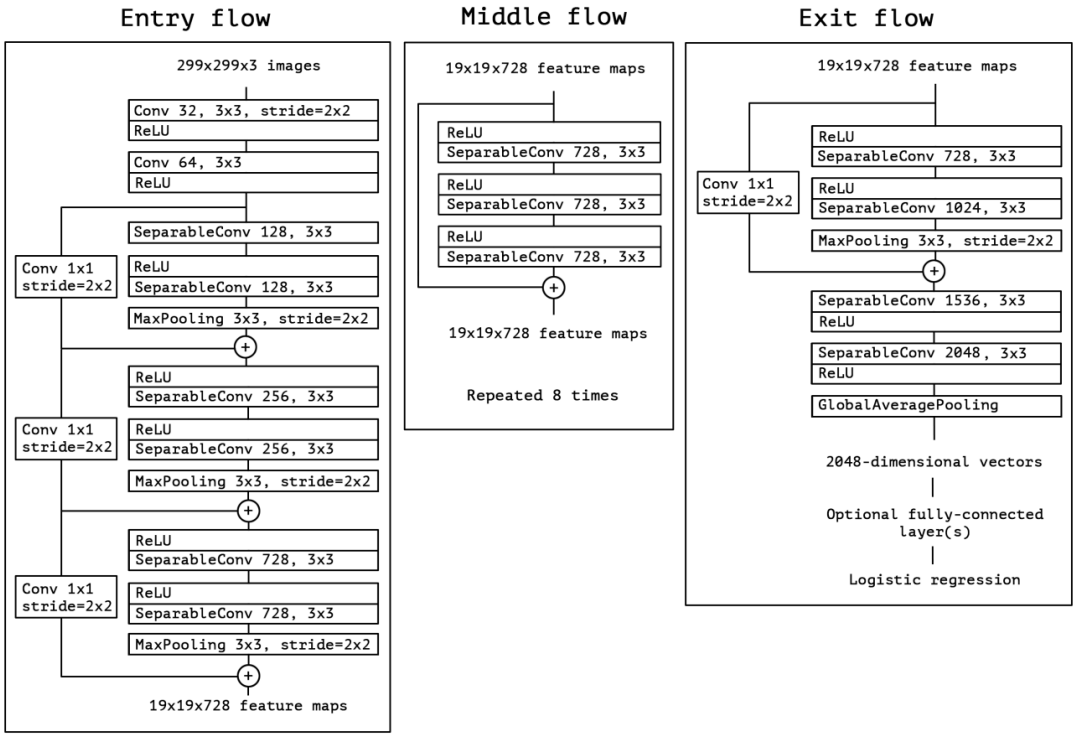

Figure 3.6: Xception Architecture, Source: [4]

input image. This helps in detecting more abstract and spatially extended features.

5. **Non Linearity and Expressive Power**
   Activation Functions: Convolutional layers typically include non-linear activation functions (e.g., ReLU). Stacking multiple convolutions increases the model's expressive power, enabling it to capture complex relationships in the data.

The reasons, why we build our model in this form are described as follows:

- **Enhanced Feature Learning**
  By having multiple convolutional layers before pooling, the model can learn more complex and detailed features, which are crucial for high performance in tasks like image classification.

- **Balanced Computational Efficiency**
  The architecture balances the depth and width of the network, ensuring that it can handle complex features without becoming too computationally expensive.

- **Improved Performance**
  imilar architectures, such as VGGNet, have shown high performance in

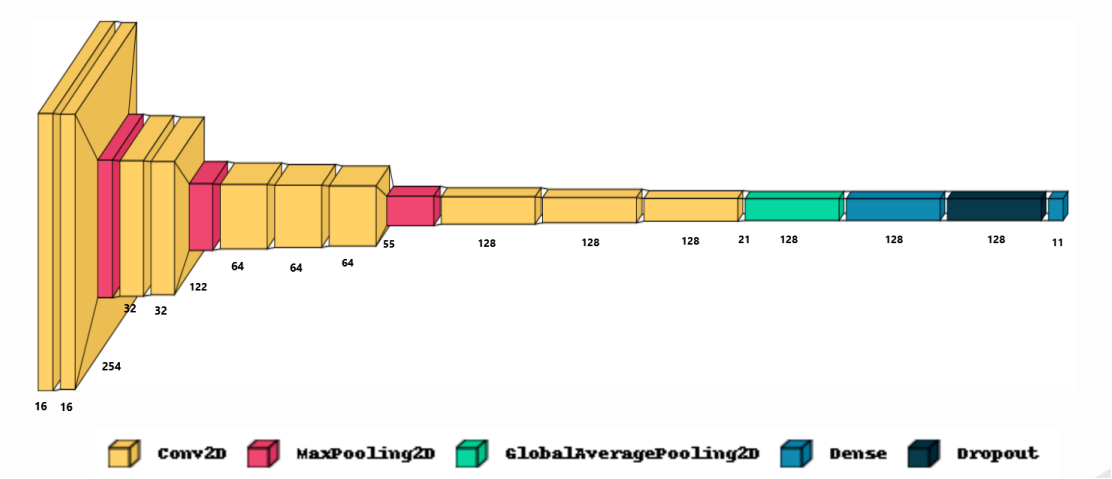

Figure 3.7: CNN10L Architecture

various image classification benchmarks. The design choices are often guided by empirical validation and successful practices in the field.

- **Prevention of Overfitting**
  Periodic max pooling reduces the number of parameters and helps prevent overfitting, which is particularly important for deeper networks.

3.3.3 Traditional/Classical Machine Learning Classifier Model

Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane [23] [24].

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

K-Nearest Neighbor (KNN)

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While the KNN algorithm can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another [25].

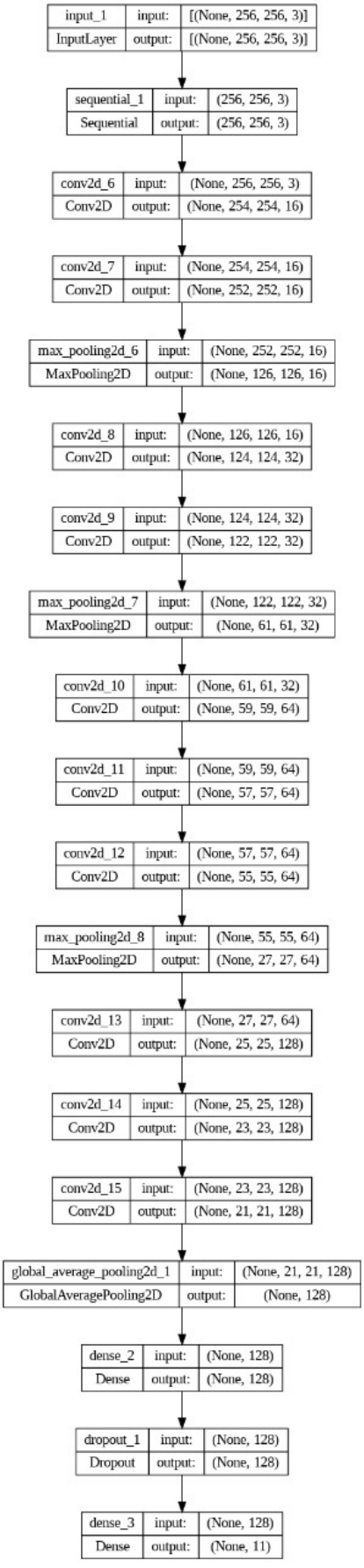

Figure 3.8: CNN10L Detail Architecture

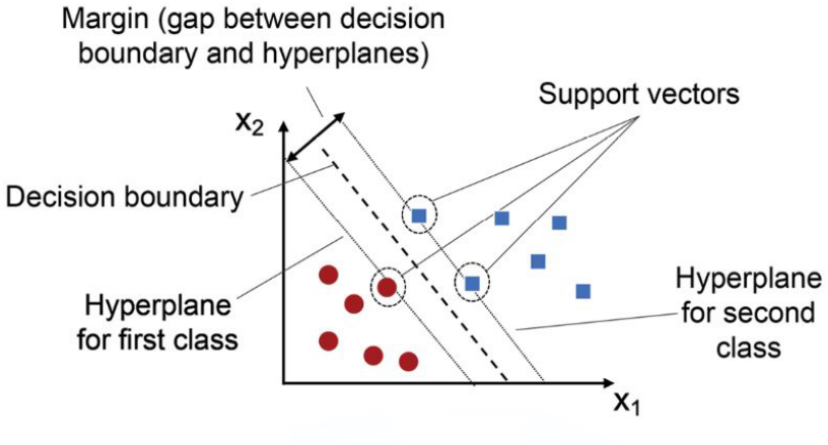

Figure 3.9: SVM in linearly separable class, Source: [5]

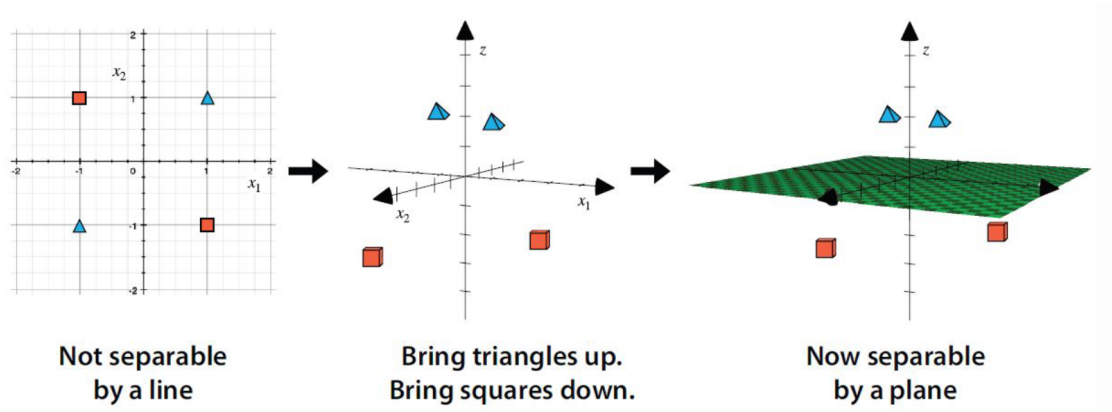

Figure 3.10: SVM using Kernel Trick to classify non-linearly separable classes, Source: [5]

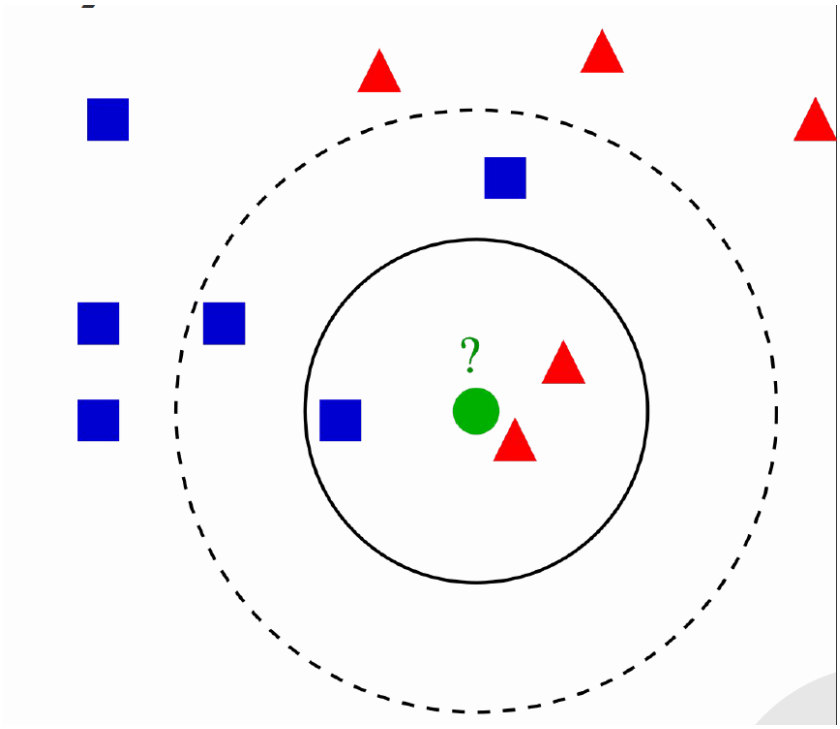

Figure 3.11: Example of k-NN classification.
The test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle) [26].

Type/Stride	Filter Shape	Input Size	Output Size
conv2d / s1	**3 x 3 x 16**	**3 x 3 x 16**	**254 x 254 x 3 x 16**
conv2d / s1	**3 x 3 x 16**	**254 x 254 x 16**	**252 x 252 x 16**
max_pool / s2	**2 x 2**	**252 x 252 x 16**	**126 x 126 x 16**
conv2d / s1	**3 x 3 x 32**	**126 x 126 x 16**	**124 x 124 x 32**
conv2d / s1	**3 x 3 x 32**	**124 x 124 x 32**	**122 x 122 x 32**
max_pool / s2	**2 x 2**	**122 x 122 x 32**	**61 x 61 x 32**
conv2d / s1	**3 x 3 x 64**	**61 x 61 x 32**	**59 x 59 x 64**
conv2d / s1	**3 x 3 x 64**	**59 x 59 x 64**	**57 x 57 x 64**
conv2d / s1	**3 x 3 x 64**	**57 x 57 x 64**	**55 x 55 x 64**
max_pool / s2	**2 x 2**	**55 x 55 x 64**	**27 x 27 x 64**
conv2d / s1	**3 x 3 x 128**	**27 x 27 x 64**	**25 x 25 x 128**
conv2d / s1	**3 x 3 x 128**	**25 x 25 x 128**	**23 x 23 x 128**
conv2d / s1	**3 x 3 x 128**	**23 x 23 x 128**	**21 x 21 x 128**
global avg pool 2D		**21 x 21 x 128**	**128**
FCN		**128**	**128**
dropout		**128**	**128**
FCN		**128**	**11**

Table 3.1: CNN10L Architecture

For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used.

Ensemble Techniques

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone [27].

Supervised learning algorithms perform the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem. Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a (hopefully) better hypothesis.

Ensemble learning trains two or more Machine Learning algorithms to a specific classification or regression task. The algorithms within the ensemble learning model are generally referred as “base models”, “base learners” or “weak learners” in literature. The base models can be constructed using a single modelling algorithm or several different algorithms. The idea is train a diverse collection of weak performing models to the same modelling task. As a result, the predicted or classified outcomes of each weak learner have poor predictive ability (high bias, i.e. high model errors) and among the collection of all weak

learners the outcome and error values exhibit high variance. Fundamentally, an ensemble learning model trains many (at least 2) high-bias (weak) and high-variance (diverse) models to be combined into a stronger and better performing model. Essentially, it's a set of algorithmic models — which would not produce satisfactory predictive results individually — that get's combined or averaged over all base models to produce a single high performing, accurate and low-variance model to fit the task as required.

Ensemble learning typically refers to Bagging (bootstrap-aggregating), Boosting or Stacking/Blending techniques to induce high variability among the base models. Bagging creates diversity by generating random samples from the training observations and fitting the same model to each different sample — also known as "homogeneous parallel ensembles". Boosting follows an iterative process by sequentially training each next base model on the up-weighted errors of the previous base model's errors, producing an additive model to reduce the final model errors — also known as "sequential ensemble learning". Stacking or Blending consists of different base models, each trained independently (i.e. diverse/high variability) to be combined into the ensemble model — producing a "heterogeneous parallel ensemble". Common applications of ensemble learning include Random Forests (extension of Baggin), Boosted Tree-Models, Gradient Boosted Tree-Models and models in applications of stacking are generally more task-specific — such as combing clustering techniques with other parametric and/or non-parametric techniques.

Random Forest

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set.

Trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model [28] [29].

Extreme Gradient Boosting (XGBoost)

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. It's a powerful machine learning algorithm especially popular for structured or tabular data. XGBoost has gained fame for

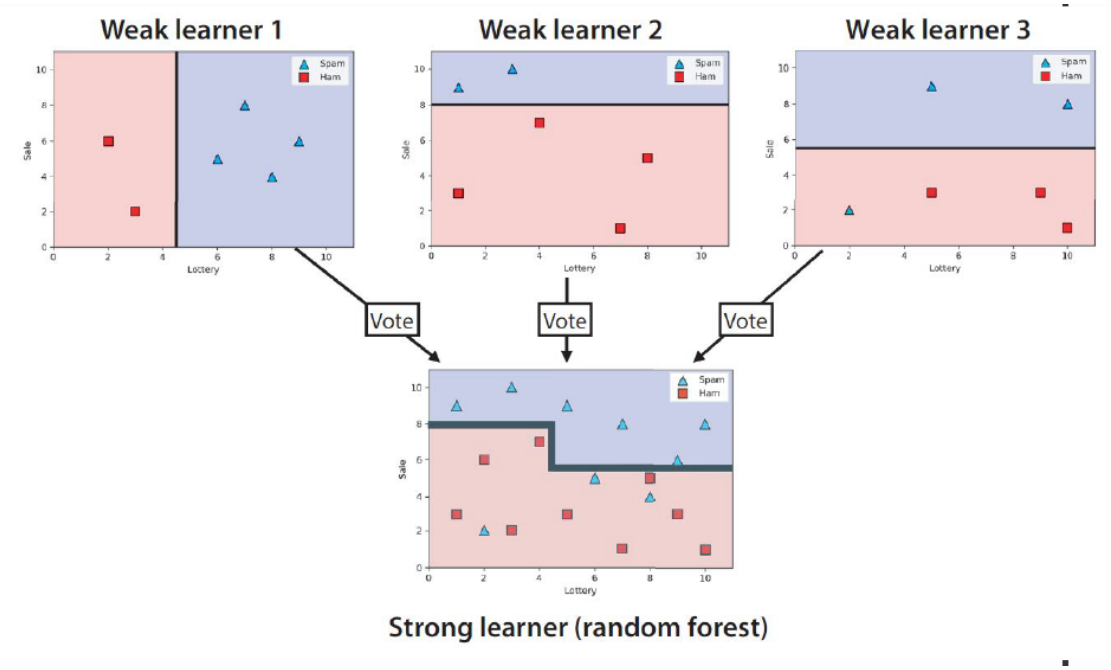

Figure 3.12: Random Forest, Source: [5]

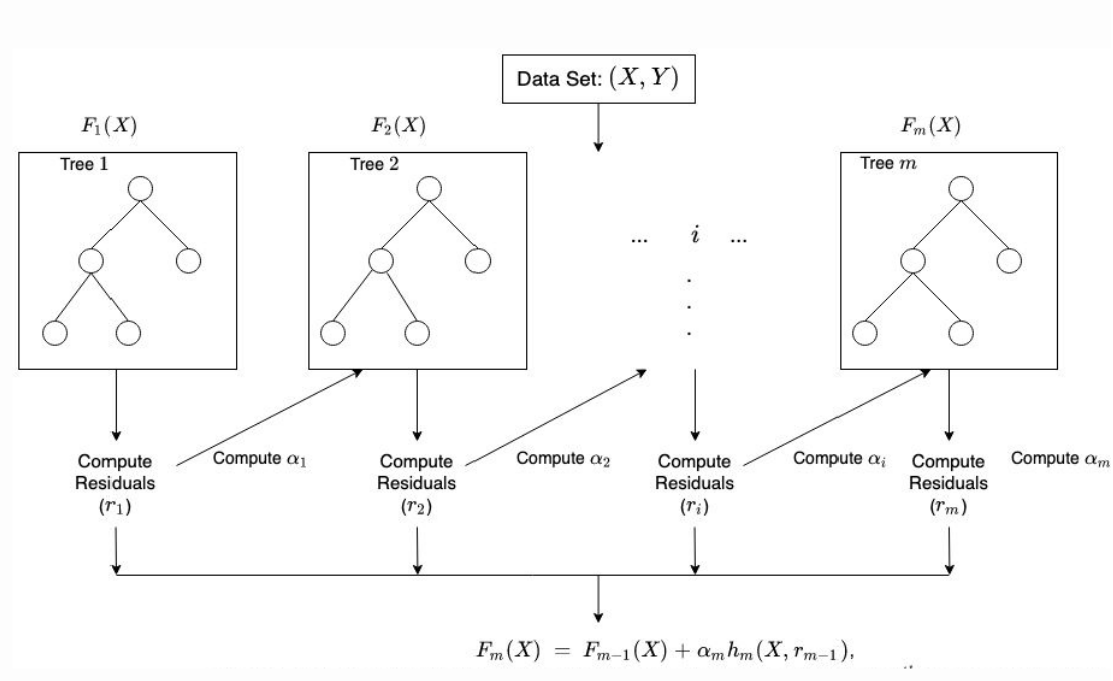

Figure 3.13: XGBoost Tree, Source: [6]

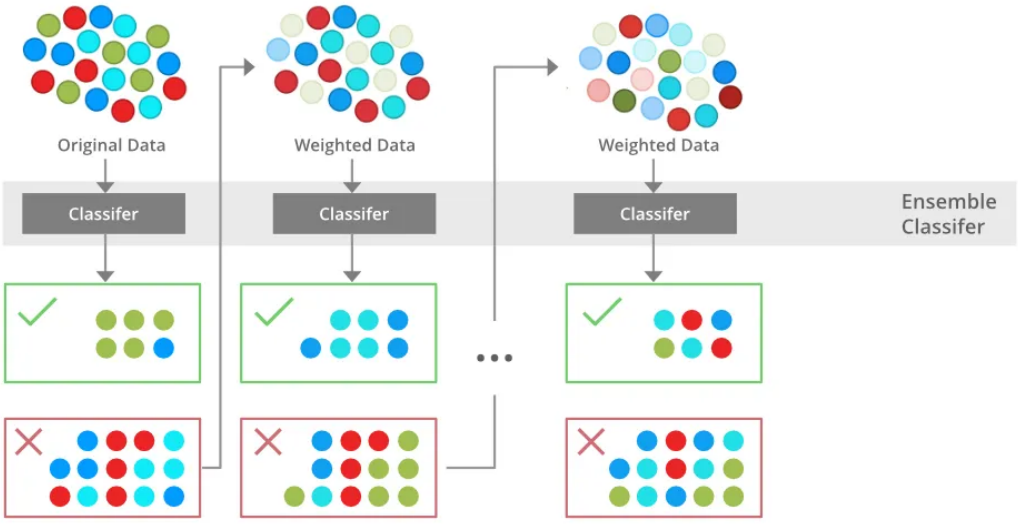

Figure 3.14: XGBoost Working, Source: [7]

its performance in a wide range of machine learning competitions and tasks [30].

Since its introduction, this algorithm has not only been credited with winning numerous Kaggle competitions but also for being the driving force under the hood for several cutting-edge industry applications.

XGBoost Features:

1. **Parallelization:** XGBoost approaches the process of sequential tree building using parallelized implementation. This is possible due to the interchangeable nature of loops used for building base learners; the outer loop that enumerates the leaf nodes of a tree, and the second inner loop that calculates the features. This nesting of loops limits parallelization because without completing the inner loop (more computationally demanding of the two), the outer loop cannot be started. Therefore, to improve run time, the order of loops is interchanged using initialization through a global scan of all instances and sorting using parallel threads. This switch improves algorithmic performance by offsetting any parallelization overheads in computation.

2. **Tree Pruning:** The stopping criterion for tree splitting within GBM framework is greedy in nature and depends on the negative loss criterion at the point of split. XGBoost uses 'max_depth' parameter as specified instead of criterion first, and starts pruning trees backward. This 'depth-first' approach improves computational performance significantly.

3. **Hardware Optimization:** This algorithm has been designed to make efficient use of hardware resources. This is accomplished by cache

awareness by allocating internal buffers in each thread to store gradient statistics. Further enhancements such as 'out-of-core' computing optimize available disk space while handling big data-frames that do not fit into memory.

Algorithmic Enhancements:

1. **Regularization:** It penalizes more complex models through both LASSO (L1) and Ridge (L2) regularization to prevent overfitting.

2. **Sparsity Awareness:** XGBoost naturally admits sparse features for inputs by automatically 'learning' best missing value depending on training loss and handles different types of sparsity patterns in the data more efficiently.

3. **Weighted Quantile Sketch:** XGBoost employs the distributed weighted Quantile Sketch algorithm to effectively find the optimal split points among weighted datasets.

4. **Cross-validation:** The algorithm comes with built-in cross-validation method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run.

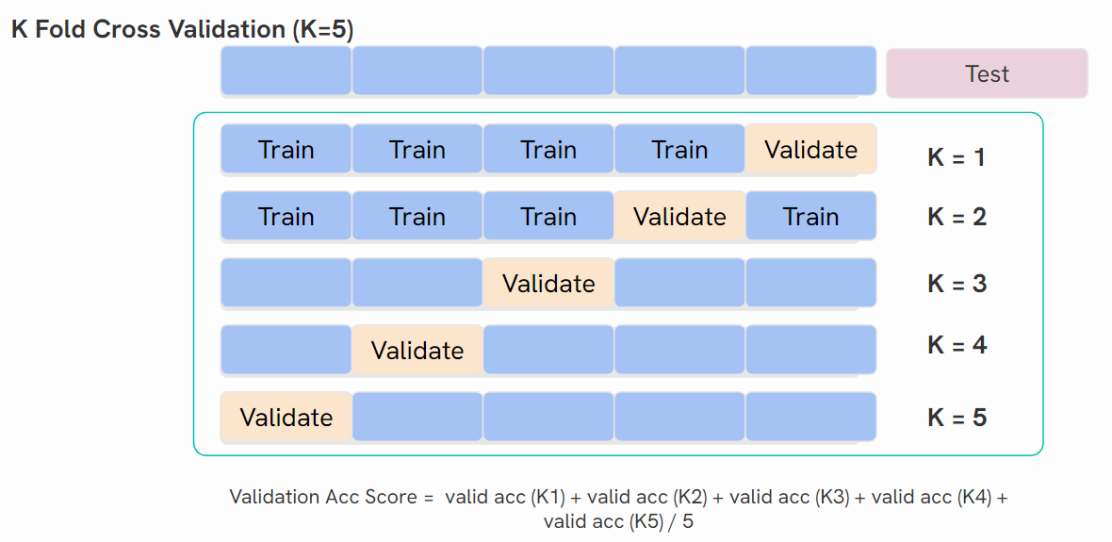

Figure 3.15: K Fold Cross Validation

3.4 K Fold Validation

K-fold cross validation is a powerful technique for evaluating predictive models. It involves splitting the dataset into k subsets or folds, where each fold is used as the validation set in turn while the remaining k-1 folds are used for training. This process is repeated k times, and performance metrics such as accuracy, precision, and recall are computed for each fold. By averaging these metrics, we obtain an estimate of the model's generalization performance. This method is essential for model assessment, selection, and hyperparameter tuning, offering a reliable measure of a model's effectiveness. Compared to leave-one-out cross-validation, which uses k equal to the number of samples, K-fold cross-validation is computationally efficient and widely used in practice [31] [32].

In each set (fold) training and the test would be performed precisely once during this entire process. It helps us to avoid overfitting. As we know when a model is trained using all of the data in a single short and give the best performance accuracy. To resist this k fold cross validation in machine learning cross-validation helps us to build the model is a generalized one.

3.4.1 Stratified K fold validation

Stratified K-Fold Cross-Validation is a variation of K-Fold Cross-Validation that ensures each fold maintains the same proportion of observations for each target class as the complete dataset. This is especially crucial for datasets where one class might be heavily underrepresented.

Figure 3.16: Stratified K Fold Validation, Source: [8]

3.5 Perfomance Metrics for Image Classification

Evaluating the performance of a Machine learning model is one of the important steps while building an effective ML model. To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics or evaluation metrics. These performance metrics help us understand how well our model has performed for the given data. In this way, we can improve the model's performance by tuning the hyper-parameters. Each ML model aims to generalize well on unseen/new data, and performance metrics help determine how well the model generalizes on the new dataset [33].

In a classification problem, the category or classes of data is identified based on training data. The model learns from the given dataset and then classifies the new data into classes or groups based on the training. It predicts class labels as the output, such as Yes or No, 0 or 1, Spam or Not Spam, etc.

To evaluate the performance of a classification model, different metrics are used, and some of them are as follows:

3.5.1 Accuracy

The accuracy metric is one of the simplest Classification metrics to implement, and it can be determined as the number of correct predictions to the total number of predictions.

$$Accuracy = \frac{\text{No Of Correct Prediction}}{\text{Total Number Of Prediction}}$$

3.5.2 Confusion Matrix

A confusion matrix is a tabular representation of prediction outcomes of any binary classifier, which is used to describe the performance of the classification

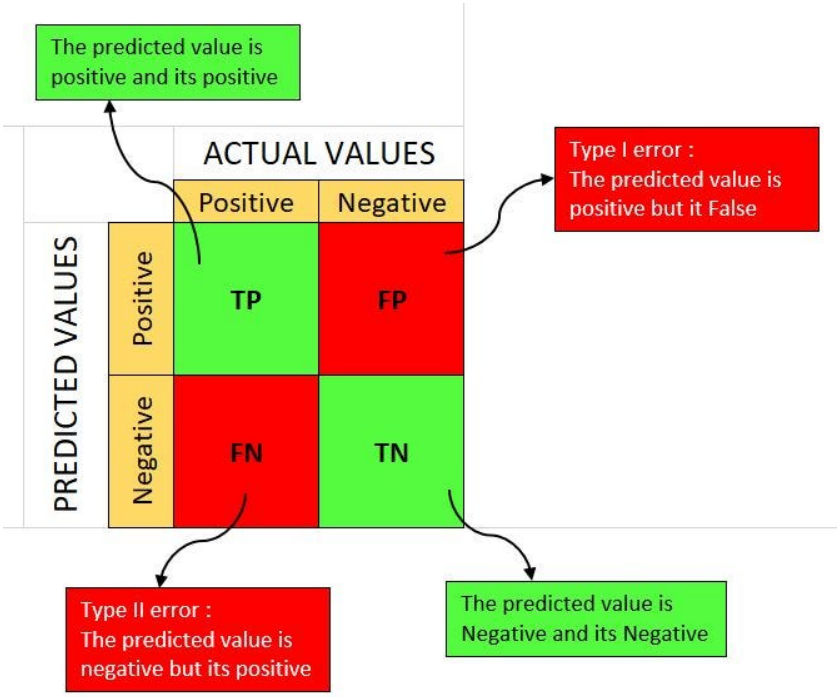

Figure 3.17: Confusion Matrix, Source: [9]

model on a set of test data when true values are known.

In general, the confusion matrix table is divided into four terminologies, which are as follows:

1. **True Positive(TP):** In this case, the prediction outcome is true, and it is true in reality, also.
2. **True Negative(TN):** in this case, the prediction outcome is false, and it is false in reality, also.
3. **False Positive(FP):** In this case, prediction outcomes are true, but they are false in actuality.
4. **False Negative(FN):** In this case, predictions are false, and they are true in actuality.

3.5.3 Precision

Precision (also called positive predictive value) gives the proportion of positive identifications that were actually correct. Written as a formula [34]:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3.5.4 Recall

It is also similar to the Precision metric, however, it aims to calculate the proportion of actual positives that was identified incorrectly. It can be calculated as

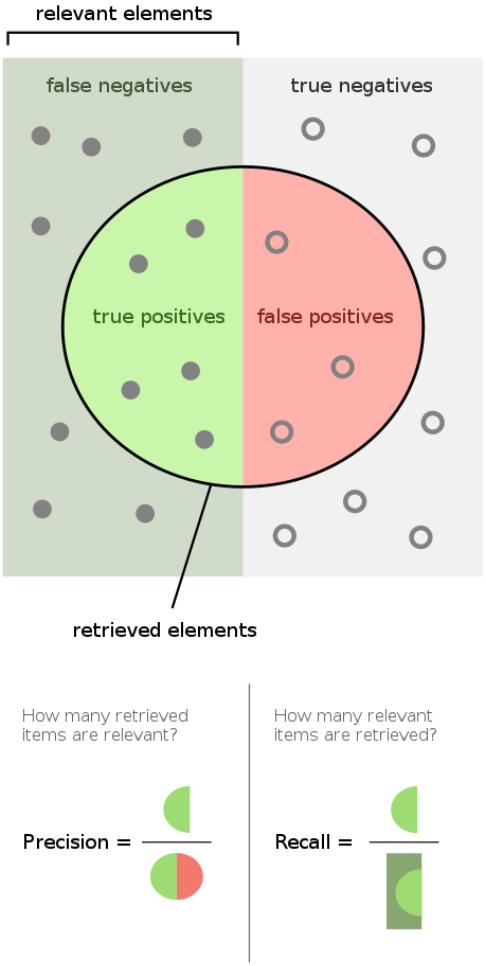

Figure 3.18: Precision and Recall

True Positive or prediction that are actually true to the total number of positives, either correctly predicted as positive or incorrectly predicted as negative (true Positive and false negative) [34].

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

3.5.5 When to use Precision and Recall

From the above definitions of Precision and Recall, we can say that recall determines the performance of a classifier with respect to a false negative, whereas precision gives information about the performance of a classifier with respect to a false positive.

So, if we want to minimize the false negative, then, Recall should be as near to 100%, and if we want to minimize the false positive, then precision should be close to 100% as possible.In simple words, if we maximize precision, it will minimize the FP errors, and if we maximize recall, it will minimize the FN error.

3.5.6 F1 Score

F-score or F1 Score is a metric to evaluate a binary classification model on the basis of predictions that are made for the positive class. It is calculated with the help of Precision and Recall. It is a type of single score that represents both Precision and Recall. So, the F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.

The formula for calculating the f1 score is given below [33]:

$$\text{F1 Score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

3.5.7 When to use F-Score

As F-score make use of both precision and recall, so it should be used if both of them are important for evaluation, but one (precision or recall) is slightly more important to consider than the other. For example, when False negatives are comparatively more important than false positives, or vice versa.
par

Chapter 4

Dataset

This chapter describe the **Plant Village datasets** that we used in our work.

4.1 PlantVillage Dataset

4.1.1 Overview

PlantVillage is a dataset consisting of 54303 healthy and unhealthy leaf images divided into 38 classes by species and disease.

4.1.2 Original classes and labels

It consists of the following 38 classes and has a total of 54303 items.

1. Apple_scab
2. Apple_black_rot
3. Apple_cedar_apple_rust
4. Apple_healthy
5. Background_without_leaves
6. Blueberry_healthy
7. Cherry_powdery_mildew
8. Cherry_healthy
9. Corn_gray_leaf_spot
10. Corn_common_rust
11. Corn_northern_leaf_blight
12. Corn_healthy
13. Grape_black_rot
14. Grape_black_measles
15. Grape_leaf_blight
16. Grape_healthy
17. Orange_haunglongbing
18. Peach_bacterial_spot
19. Peach_healthy
20. Pepper_bacterial_spot
21. Pepper_healthy
22. Potato_early_blight
23. Potato_healthy
24. Potato_late_blight
25. Raspberry_healthy

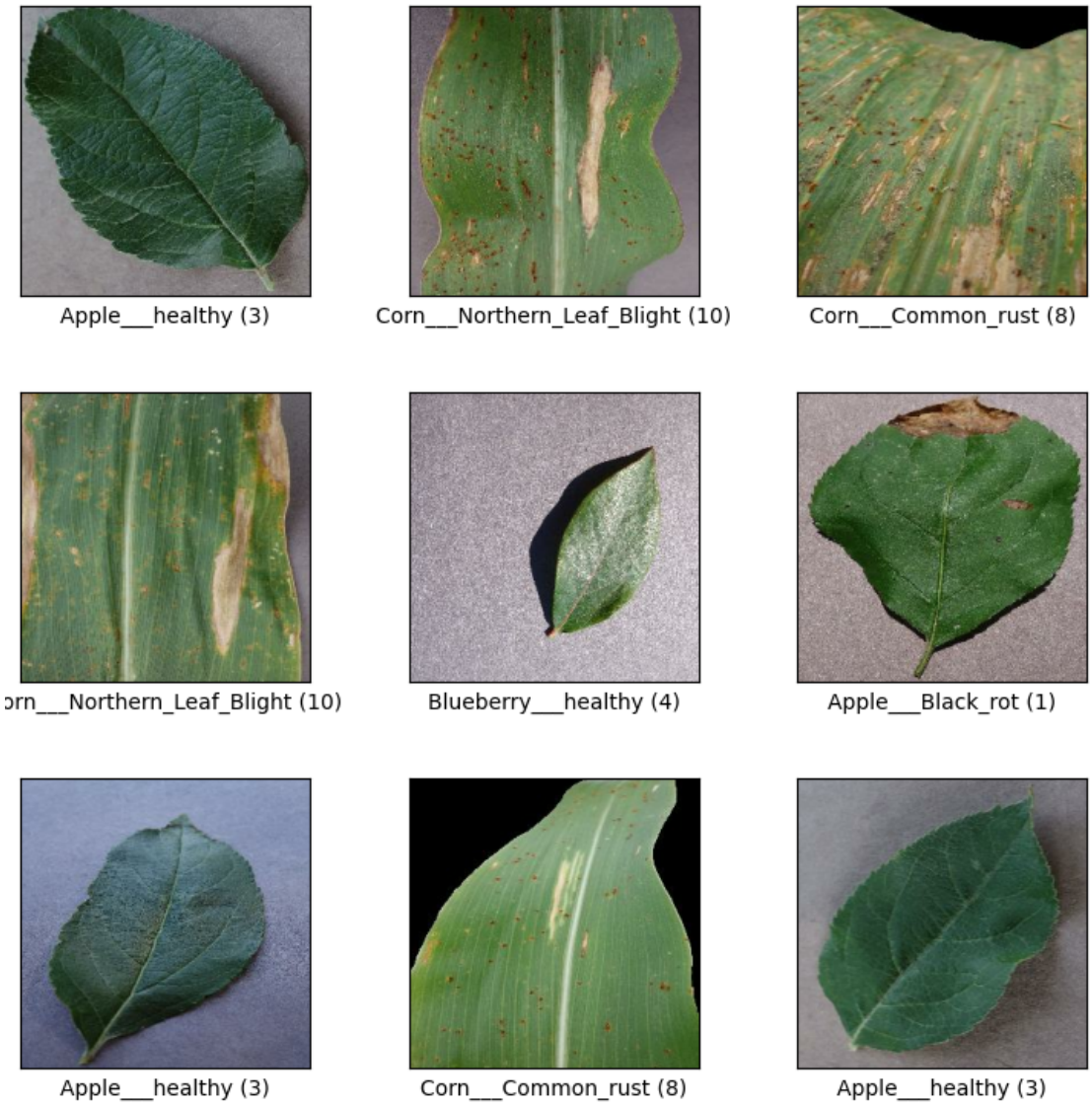

Figure 4.1: Plant Village Datasets

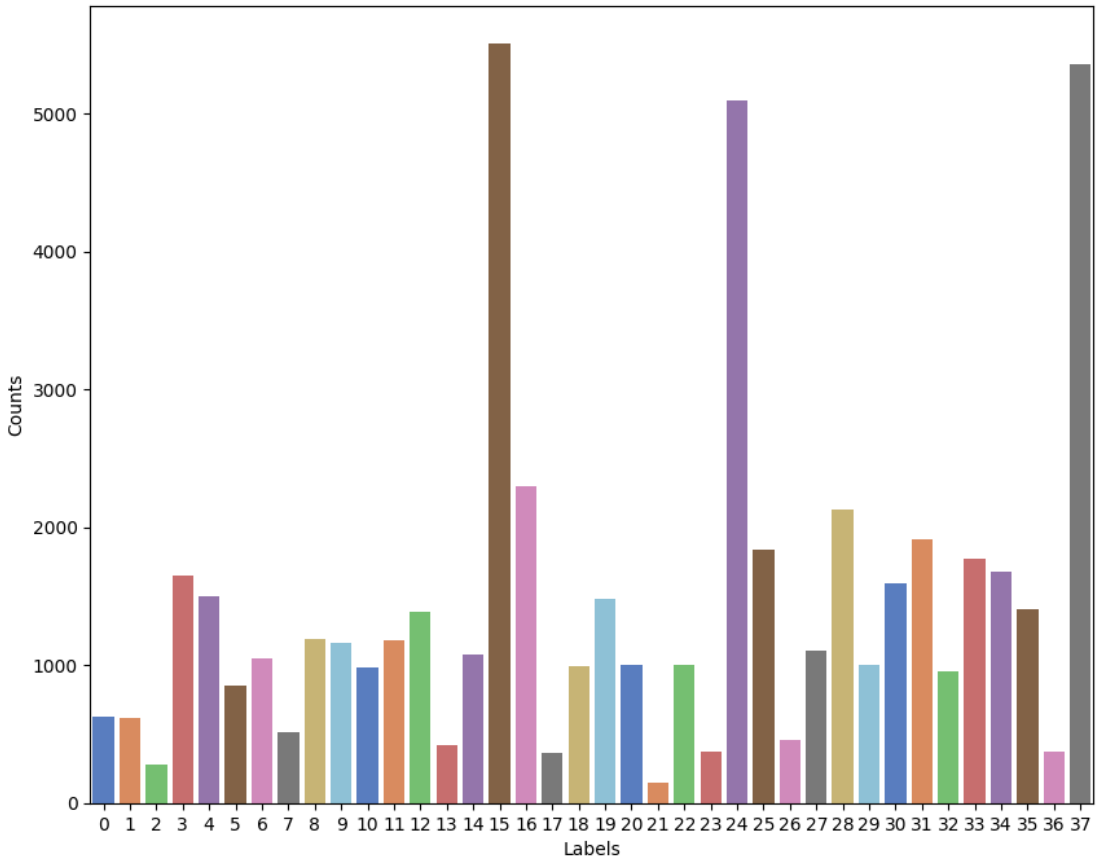

Figure 4.2: Class Distribution Before Class Reduction

26. Soybean_healthy
27. Squash_powdery_mildew
28. Strawberry_healthy
29. Strawberry_leaf_scorch
30. Tomato_bacterial_spot
31. Tomato_early_blight
32. Tomato_healthy
33. Tomato_late_blight
34. Tomato_leaf_mold
35. Tomato_septoria_leaf_spot
36. Tomato_spider_mites_two-spotted_spider_mite
37. Tomato_target_spot
38. Tomato_mosaic_virus
39. Tomato_yellow_leaf_curl_virus

4.1.3 Classes used

We have narrowed down these 38 classes to 11 classes, because of hardware constraints.

The narrowed-down classes, totaling 10431 items are:

1. Apple_Apple_scab
2. Apple_Black_rot
3. Apple_Cedar_apple_rust
4. Apple_healthy
5. Blueberry_healthy
6. Cherry_healthy
7. Cherry_Powdery_mildew
8. Corn_Cercospora_leaf_spot_Gray_leaf_spot
9. Corn_Common_rust
10. Corn_healthy
11. Corn_Northern_Leaf_Blight

4.1.4 Item Data Composition

The original PlantVillage data was mildly imbalanced as such, our dataset also suffers from mild data imbalance.

Key:

0. Apple_Apple_scab
1. Apple_Black_rot
2. Apple_Cedar_apple_rust
3. Apple_healthy
4. Blueberry_healthy
5. Cherry_healthy
6. Cherry_Powdery_mildew
7. Corn_Cercospora_leaf_spot_Gray_leaf_spot
8. Corn_Common_rust
9. Corn_healthy
10. Corn_Northern_Leaf_Blight

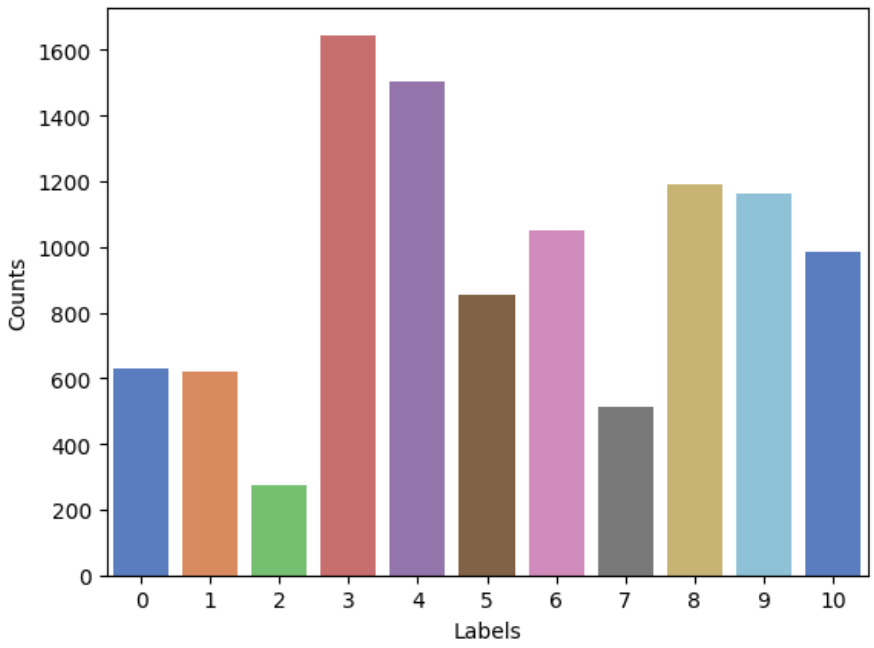

Figure 4.3: Class distribution after class reduction

A potential solution to imbalanced classes is to use either:

1. Downsampling (in this context) means training on a disproportionately low subset of the majority class examples.

2. Upweighting means adding an example weight to the downsampled class equal to the factor by which you downsampled

4.2 Data Processing Techniques

1. Image Augmentation - Importance of Augmentation in Preventing Over fitting
2. Techniques Used (e.g., Rotation, Flipping, Zooming, Cropping)
3. Data Preprocessing for each model VGG16, Xception, MobilenetV2
4. Data Splitting - Training, Validation, and Test Split Ratios, Stratified Splitting to Maintain Class Distribution, Rationale for Chosen Split Ratios
5. Handling Class Imbalance - Techniques for Managing Imbalanced Data - (e.g., Oversampling, Undersampling) Impact on Model Training and Evaluation

4.2.1 Image Augmentation

A technique used to artificially increase the training set by creating modified copies of a dataset using existing data. This technique is very useful when an image dataset is small. This also helps expose our model to different aspects of the training data and reduce overfitting.

We implement this process by adding an Augmentation Layer to our classifier models. Augmentation Layer are only active during training and not during model's inference or evaluation.

Our Augmentation Layer includes: RandomFlip, RandomRotation, Saturation, Brightness

4.2.2 Data Splitting

The total of 10431 images are split 80:10:10, which is:
80% in training, i.e. 8344 images for actual training.
10% for testing, i.e. 1044 images for testing.
10% for validation, i.e. 1044 images for validating the result.

Data Splitting is important as it helps prevent overfitting, ensure robustness and generalization.

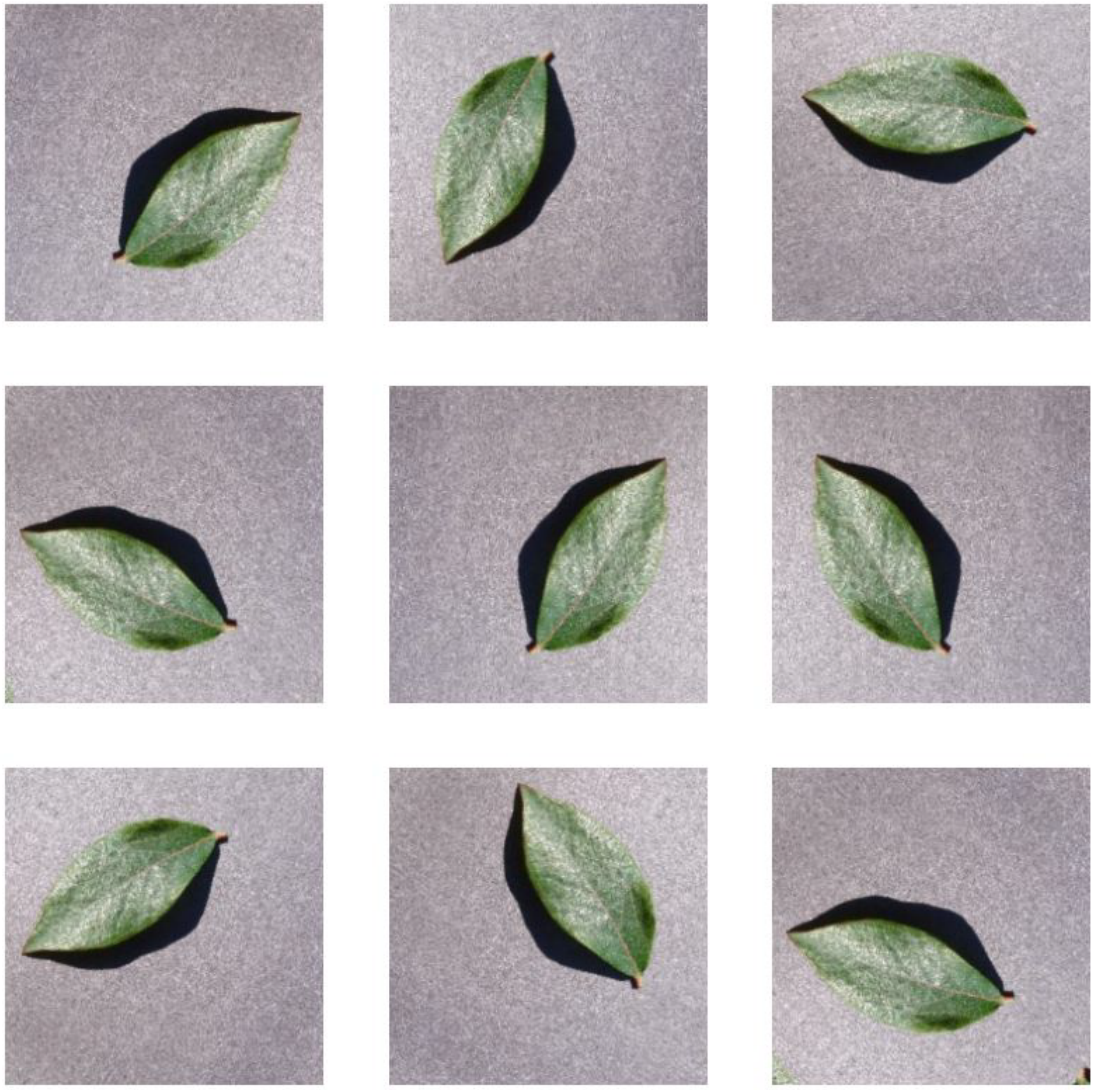

Figure 4.4: Image Augmentation

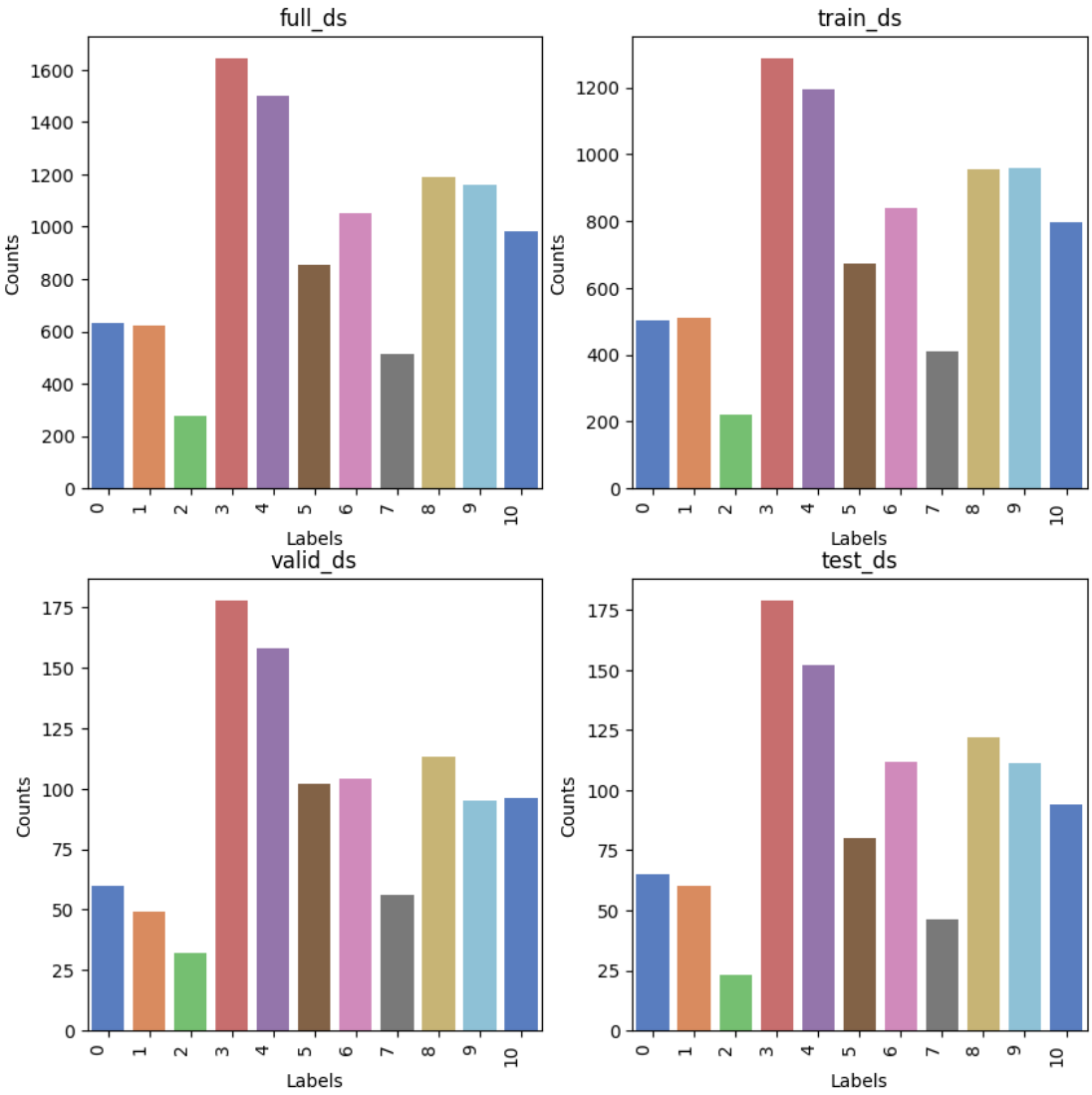

Figure 4.5: Train, Test, Validation Split

Chapter 5

Methodology

In this chapter, we provide the detail procedures, steps that we have done in our research and how our results are obtained.

5.1 Building pre-trained classifier model

For building our model for performing transfer learning, we take the following steps.
We first build our data preprocessing pipeline which involves batching our images inorder to perform prediction, inference, prediction faster, resizing our image datasets to a desired shape that can be accepted by our model. After that, we load our pre-trained model by removing their top layers and added a single Fully Connected Layers (FCN) layer which have which have size equal to the number of classes that we want to perform classifications on.

For building our pre-trained classifier model, with MobileNetV2, refer fig5.1 as base model, we first download the pretrained MobileNetV2 model with weights trained on **ImageNet datasets** and remove their top layers. Then, we add a **Global Average Pooling 2D Layer** on the top of the downloaded pretrained model to flatten the 7x7x1280 features map extracted by the base model, by taking the average of each 7 x 7 features map. Now, we add a single **FCN Layer** on top

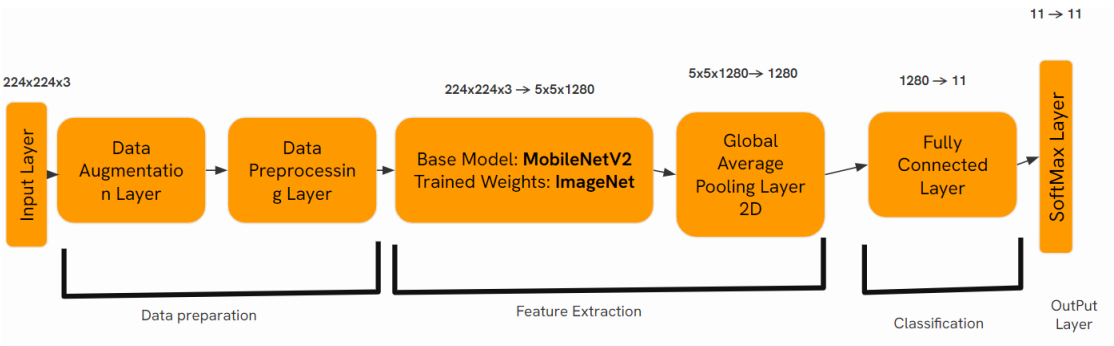

Figure 5.1: MobileNetV2 Based Model

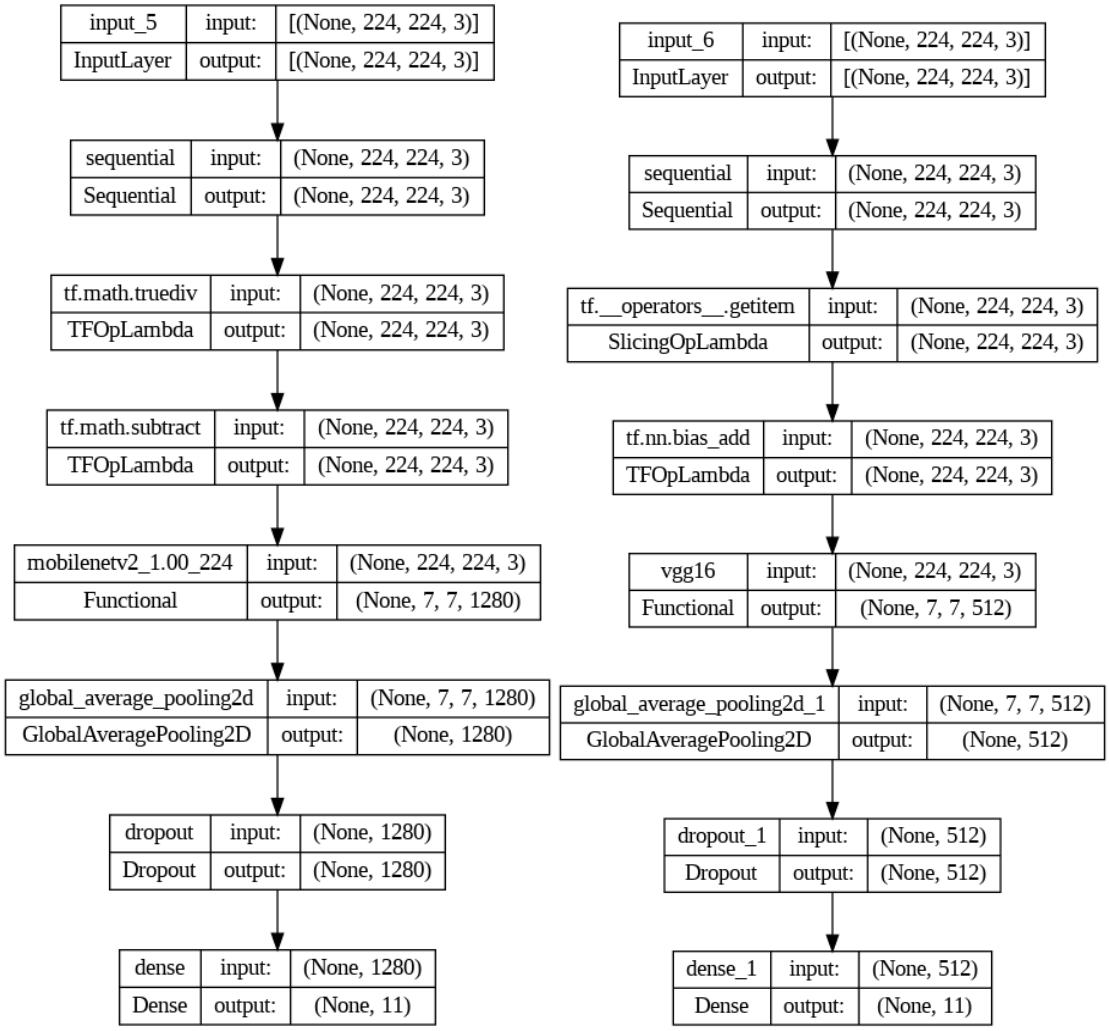

Figure 5.2: From left to right: MobilNetV2 Base Model, VGG-16 Base Model

of Global Avg Pool 2D layer as the last layer. The number of neuron units in this FCN Layer is equal to 11 units as we are planning to classify 11 classes,

Similarly, we build other pretrained model using VGG-16 and Xception as base model, using the same procedures as above. The detailed description of each model layer is shown in fig 5.2 and 5.3.

5.2 Transfer Learning

After building our MobileNetV2, VGG16, Xception base model, we start performing transfer learning and fine-tuning our models using our datasets. So, we first split our datasets on **80:10:10** ratios corresponding to training, testing and validation datasets size.

Then, we perform transfer learning by using the following approach, we freeze the weights of the entire base model, (i.e VGG16. Xception, MobileNetV2 part

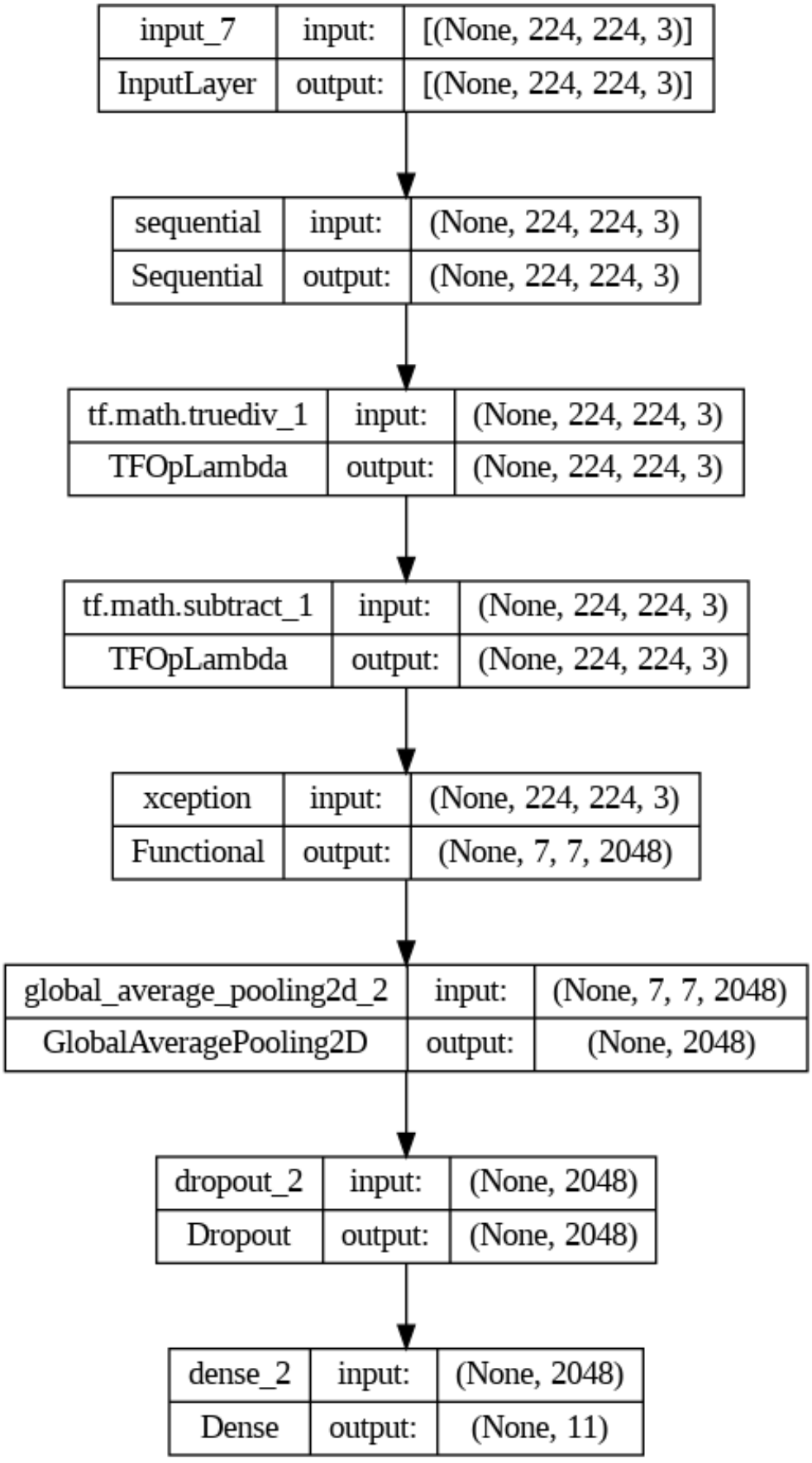

Figure 5.3: Xception Base Model

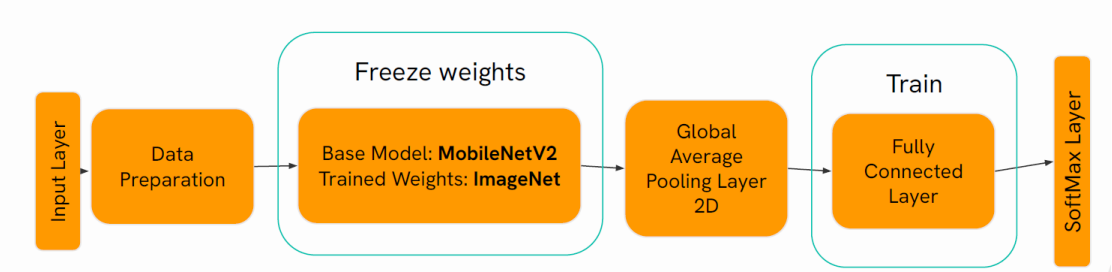

Figure 5.4: Transfer Learning

of our model), then we start training only the FCN layer of our models. Here, Freezing is a technique to prevents the weights in a given layer from being updated during training In this way, we trained our model without destroying the weights (knowledge) that it has learned previously (learned from **ImageNet datasets**) So, in transfer learning phase of the training, we just used our base model as a feature extractor and trained only the last top layer (FCN layer) that we added, to perform the classification task using the features extracted by the base model.

5.2.1 Model Training

Parameters:

- **Epochs**: 10
- **Learning Rate**: 10^{-4}
- **Loss Function**: Sparse Categorical Cross Entropy
- **Optimizer**: Adam
- **Metrics**: Accuracy

In the transfer learning phase, we observe that while training our model, MobileNetV2 base model converse fastest as compared to other models. So, we can say that MobileNetV2 is an excellent feature extractor as compared to the other 2 models that we used.

5.3 Fine Tuning

Parameters

- **Epochs**: 10
- **Learning Rate**: 10^{-5}
- **Loss Function**: Sparse Categorical Cross Entropy
- **Optimizer**: RMSProp

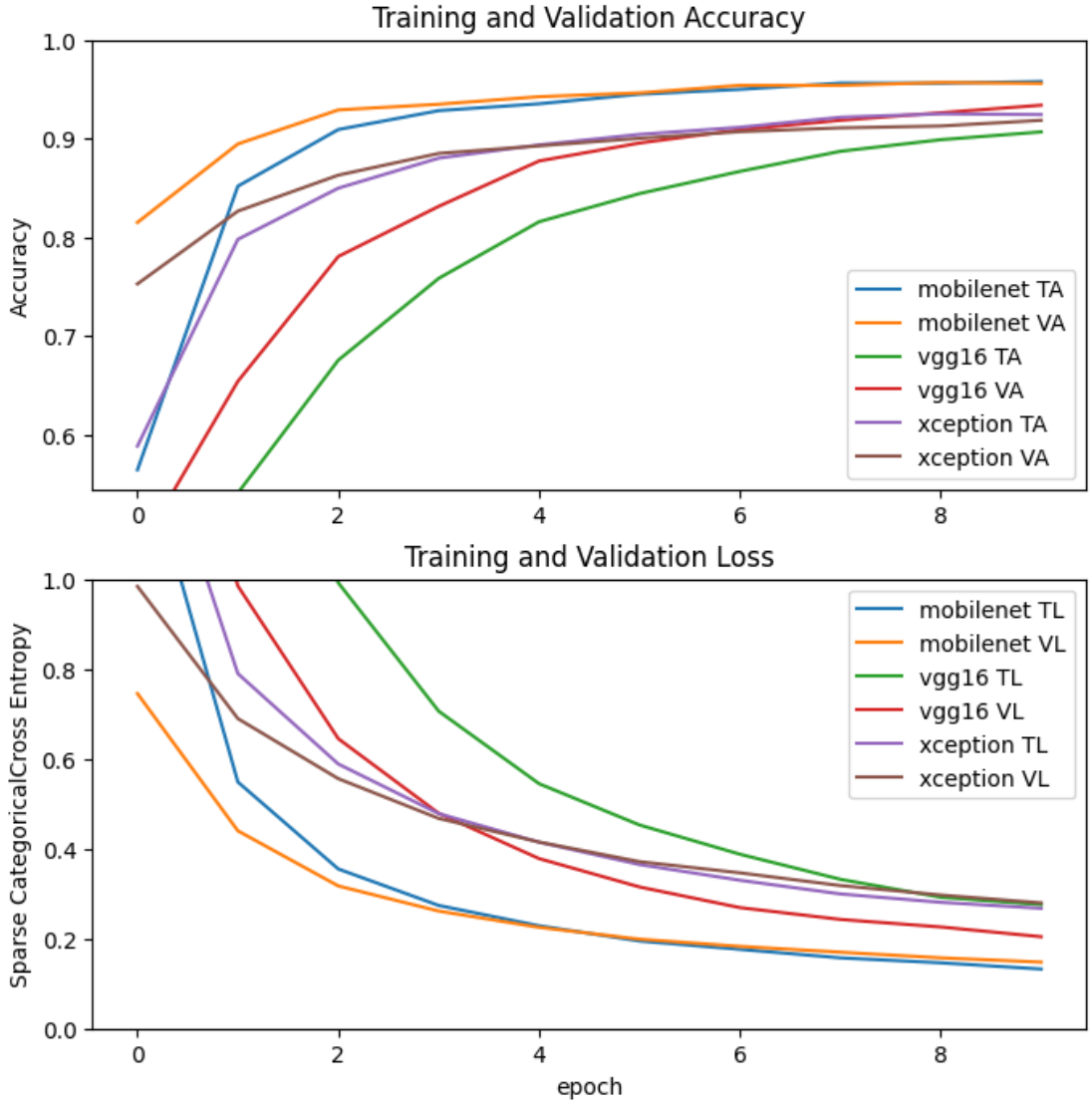

Figure 5.5: Transfer Learning Training Graph

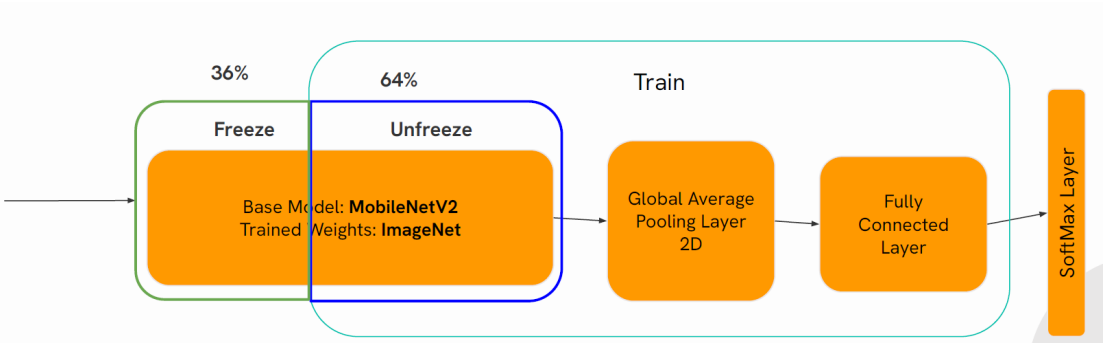

Figure 5.6: Fine Tuning

- **Metrics**: Accuracy

We perform the fine-tuning phase of our model just after transfer learning phase, this phase of training is performed mainly to further increase the performance of our model on our particular training datasets.

In transfer learning, we were only training the top layers of our base model. The weights of the pre-trained network were not updated during training. One way to increase performance even further is to train (or "fine-tune") the weights of the top layers of the pre-trained model alongside the training of the classifier we added. The training process will force the weights to be tuned from generic feature maps to features associated specifically with the dataset.

Fine tuning of our model on Plant Village Datasets is perform in the following ways, we first unfreeze 64% of the top layers of our base models and freeze the remaining 36% of our model weights. Then, we again trained our model with our datasets with the unfreeze weights and the last FCN layer.

While fine tuning, we should try to fine-tune a small number of top layers rather than the whole base model layers. The reason why we unfreeze 64% top layer of our base model and not unfreeze the entire base model weights is because in most convolutional networks, the higher up a layer is, the more specialized it is. The first few layers learn very simple and generic features that generalize to almost all types of images. As you go higher up, the features are increasingly more specific to the dataset on which the model was trained. The goal of fine-tuning is to adapt these specialized features to work with the new dataset, rather than overwrite the generic learning.

5.3.1 Model Training

During the fine-tuning phase of our model, we observe from the training curve graph, that the VGG-16 base model converges faster and outperforms other other models. So we can inferred that VGG-16 model can generalized well on specific datasets using its learned knowledge.

The success of VGG16 indicates that the pre-trained weights on ImageNet were highly beneficial and effectively transferable to the Plant Village dataset. VGG16's architecture, with its deep layers, is adept at extracting intricate and hierarchical features, which are crucial for distinguishing between different plant diseases.

The ability of VGG16 to effectively adapt pre-trained features to new data highlights the power of transfer learning and fine-tuning in achieving high accuracy and robust model performance.

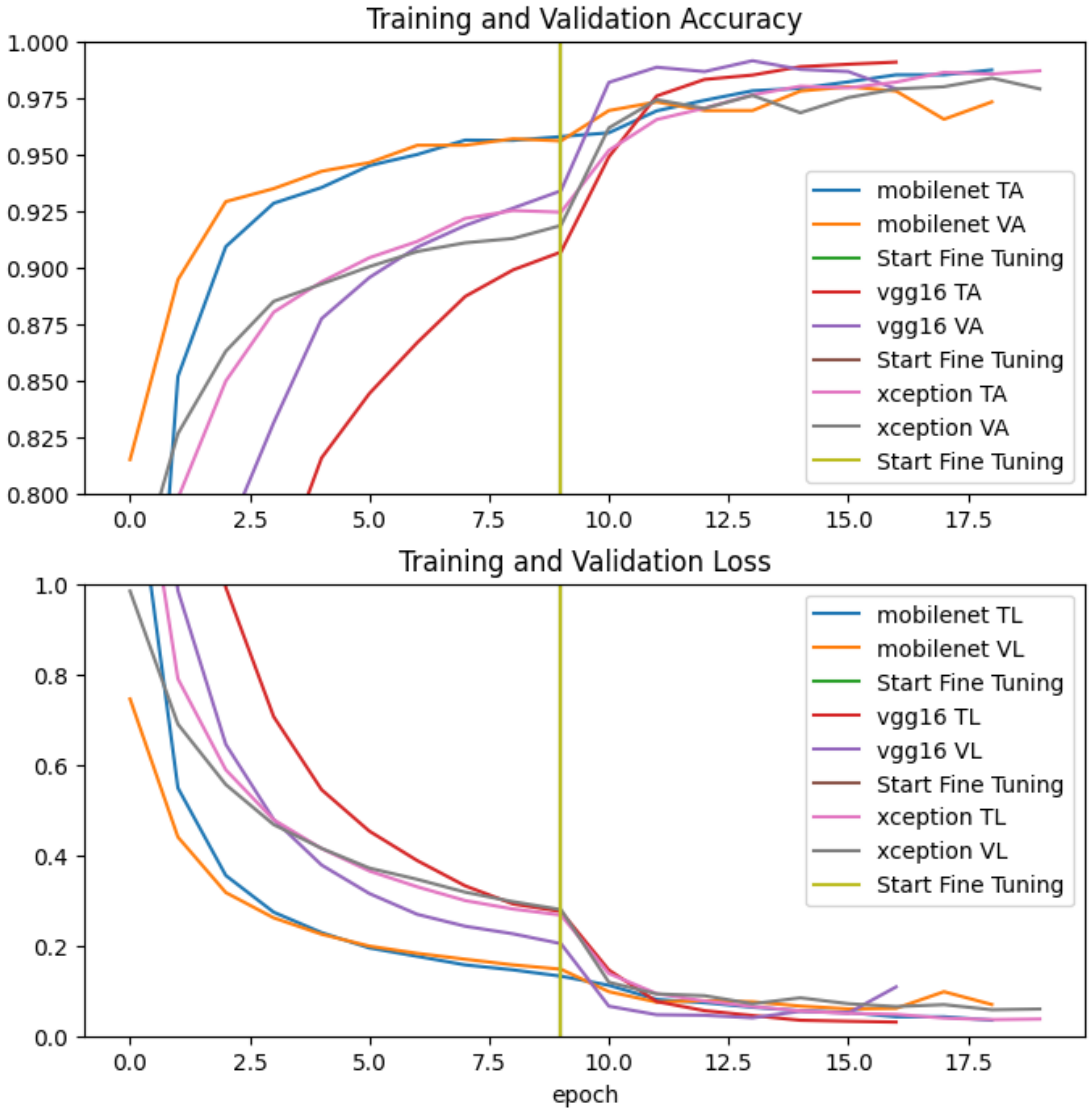

Figure 5.7: Fine Tuning Training Graph

5.4 CNN10L ConvNet Model

We build our CNN10L Model using the specifications define in 3.7 and 3.8

5.4.1 Model Training

We trained our CNN10L model using **K Fold Cross Validation Method**.

So, in this technique, we first divide our dataset into train and test split in ratio of 90:10. Then, further divide our training set into K folds. After that, we perform K fold Cv Training of our model. In the first phase (i.e. K = 1), we take the last fold as validation set and remaining as training set and perform training. Then in second K fold training phase (i.e. K = 2), we take the last second fold as validation set and remaining folds as training sets and again train our models. Similarly , we continue this process for K times taking K^{th} fold as validation step. The steps for this training is shown clearly in the figure 3.15

Parameters:

- **K** = 5 folds
- **Epochs:** 30 (with Early Stopping Condition)
- **Learning Rate:** 10^{-3}
- **Loss Function:** Sparse Categorical Cross Entropy
- **Optimizer:** Adam
- **Metrics:** Accuracy

The training curve for our CNN10LL model for 5 K folds is shown in fig5.8 and 5.9

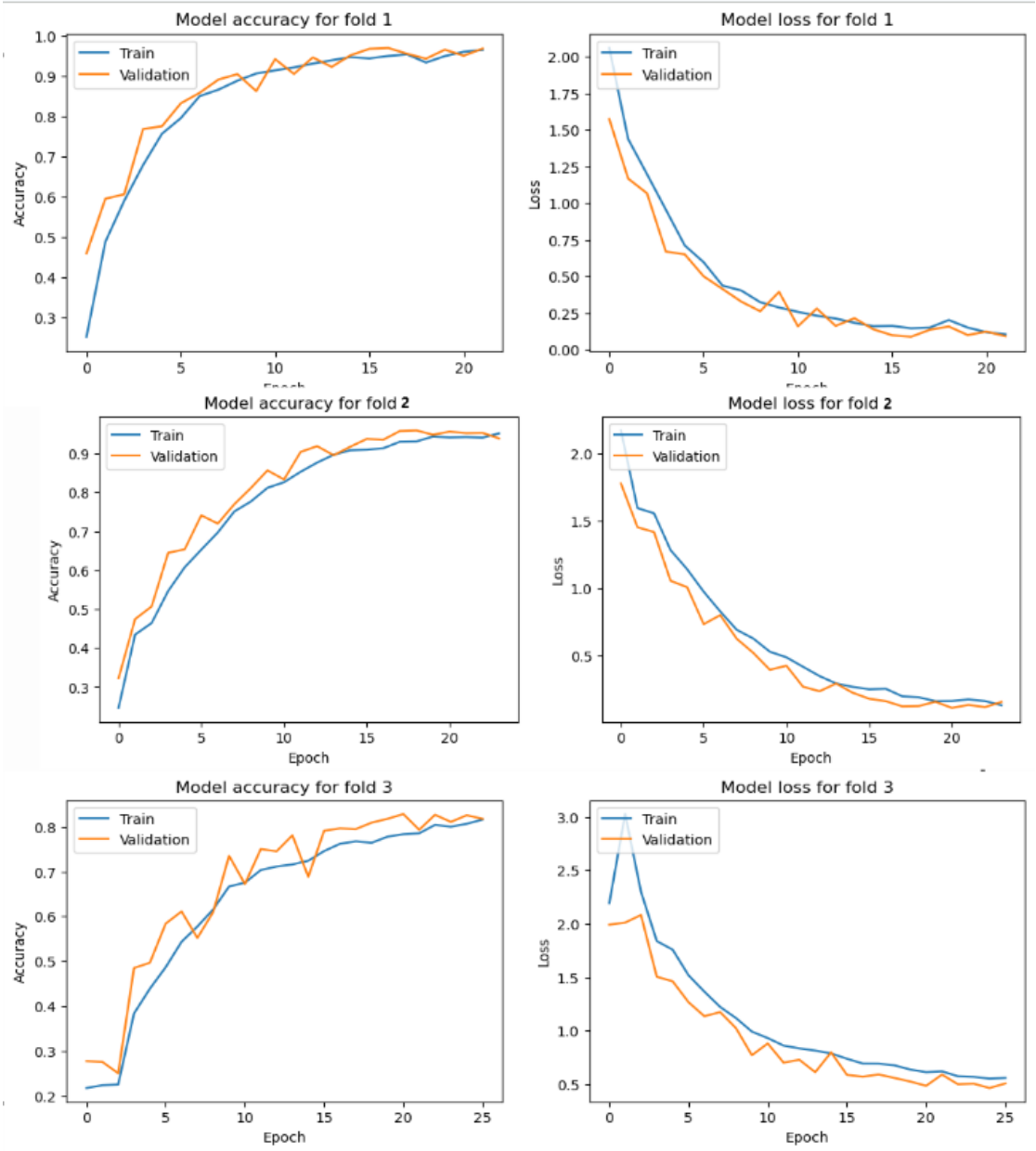

Figure 5.8: CNN10L Training for fold 1 to fold 3

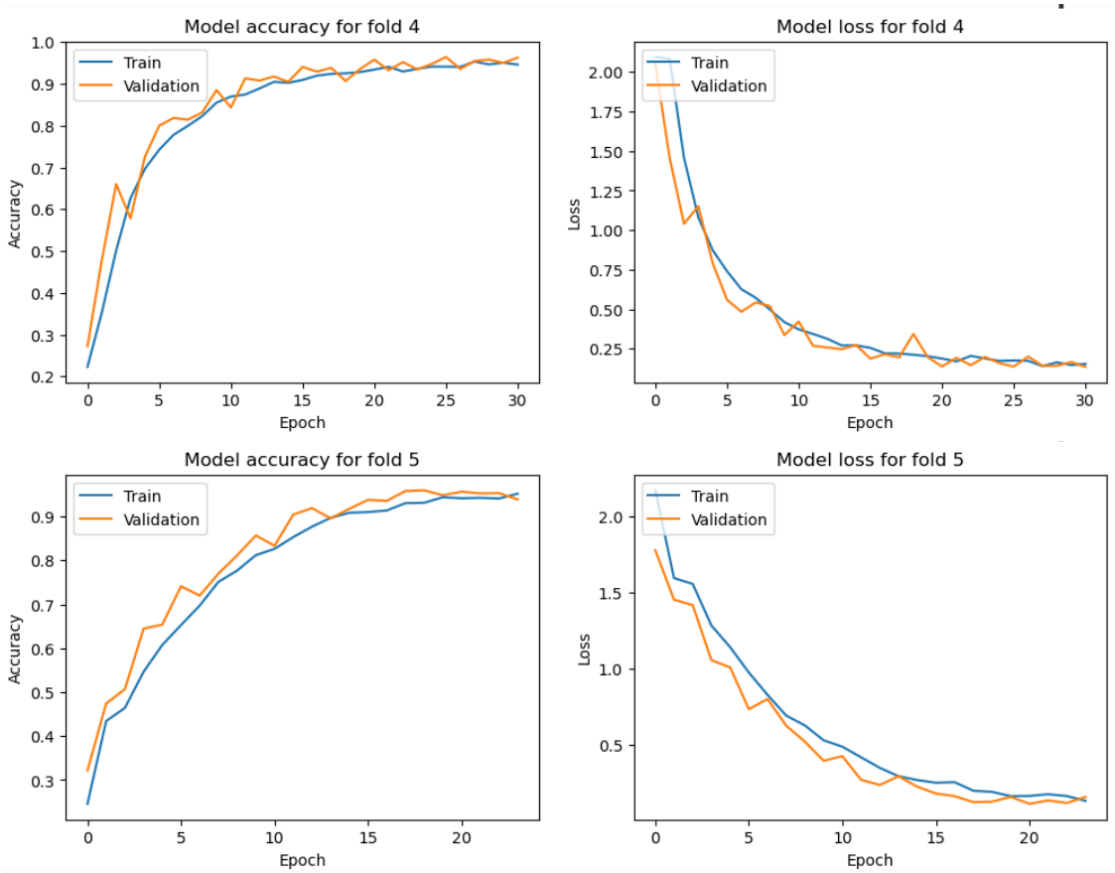

Figure 5.9: CNN10L Training for fold 4 to fold 5

5.5 Classical ML model as Classifier

Since, classical ML classifier models such as SVM, KNN, Random forest, XBoost, etc cannot extract features directly from an image, we used a pretrained MobileNetV2 as a feature extractor for training these models.

So, we first download a pretrained MobileNetV2 model (trained on **ImageNet**) and remove its top layer so that it gives the last feature maps as an output instead of the 1000 classes outputs. Then, we input an image to this MobileNetV2, which give out a 7 x 7 x 1280 features map. we perform a global average pooling 2d on this output feature map to produce a 1280 features vector (7 x 7 x 1280 to 1 x 1 x 1280).

In this way, we perform feature extraction for all the images in our dataset to convert it into 1280 feature vectors from an image of shape 256 x 256 x 3, then train our traditional classifier model on this extracted features

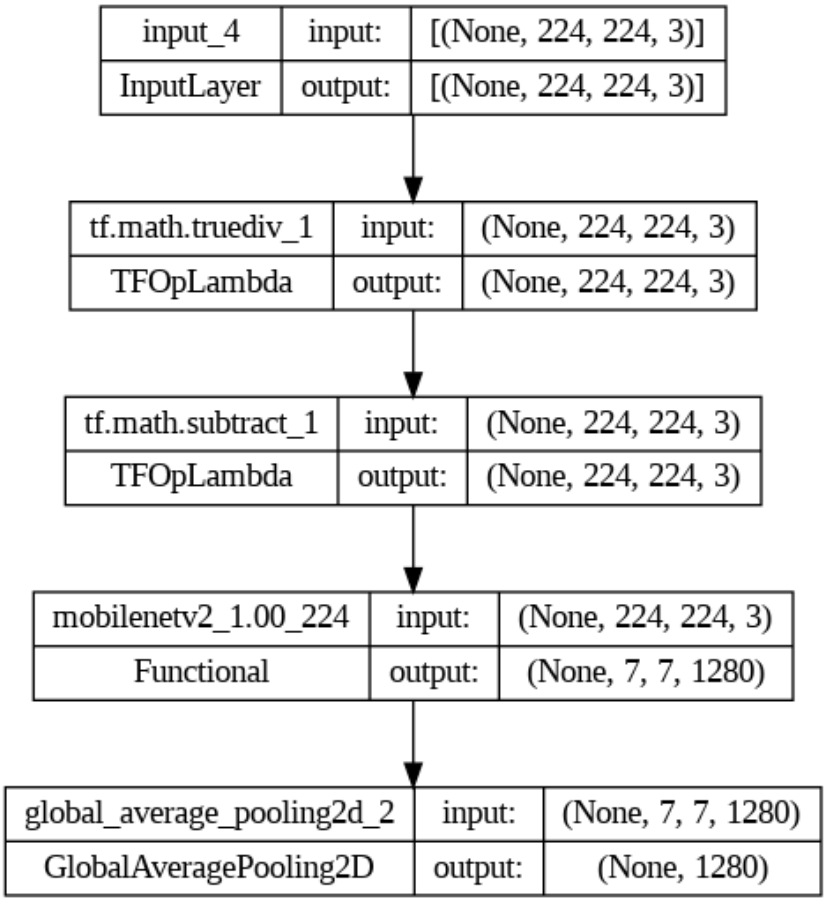

Figure 5.10: MobileNetV2 Feature Extractor used for training Classic ML Model

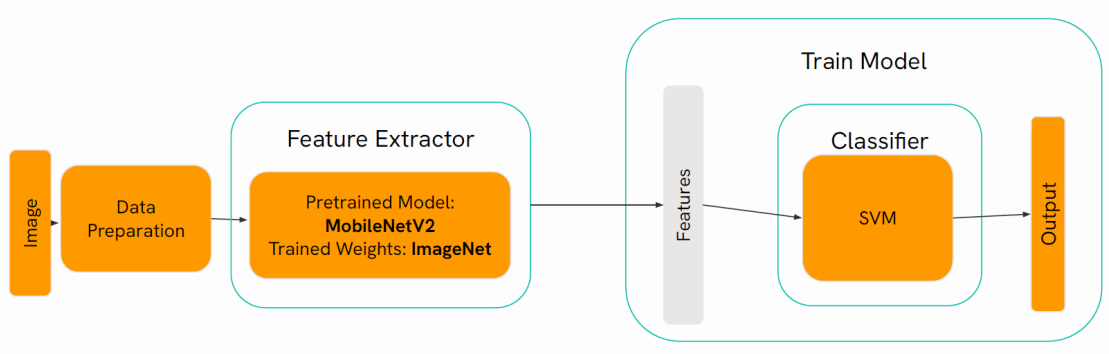

Figure 5.11: MobilenetV2 + SVM

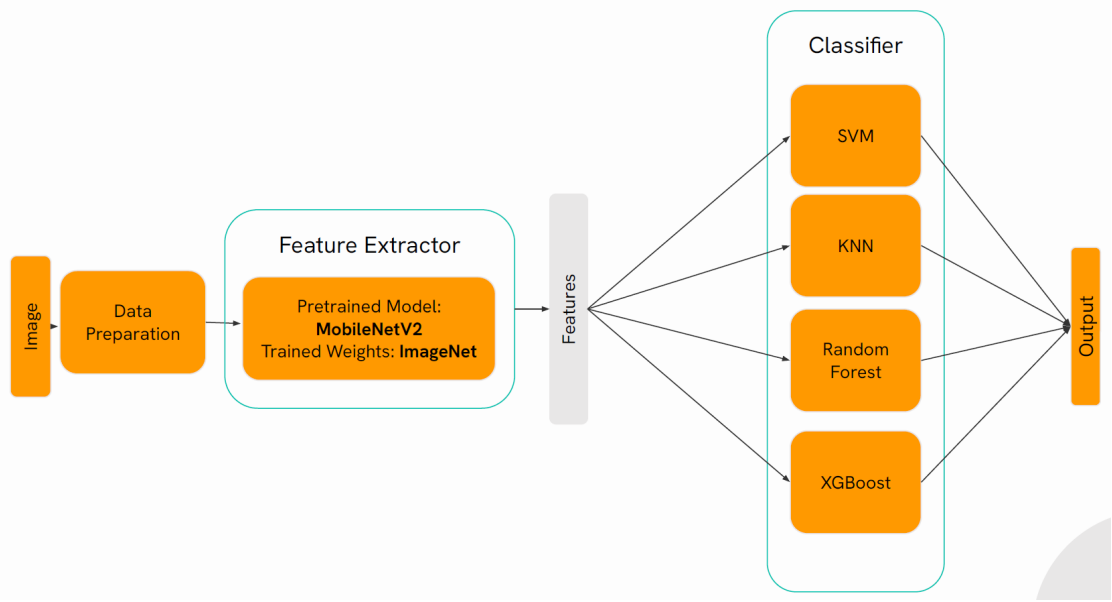

Figure 5.12: MobileNetV2 + Traditional Classifier Models

Chapter 6

Result

This chapter presents the results of our comparative study on various image classification models using the Plant Village dataset. We detail the performance metrics of each model, highlighting the strengths and weaknesses observed through their evaluation.

6.1 Comparative Analysis of Models

In this chapter, we explain the performance of different models that we used in our research with the help of graphs, tables, etc

6.1.1 Pretrained Model Accuracy and Loss Comparison After Transfer Learning

After performing transfer learning of our pretrained Convnet models, we observe that MobileNetV2 outperforms all other models with an accuracy of ***(0.9670, 0.95589, 0.9693)*** on training, validation and testing datasets, respectively. But, the performance difference between the models is very low, they all perform very well with very little difference.

As we have already mentioned in methodology section 5.2, that in transfer learning we are not touching any weights of the base model only the last FCN layer is trained, So, we can inferred from this observation that MobileNetV2 is an excellent feature extractor for generally all classes, even for narrow domain. This is the reason, why we used MobileNetV2 as an feature extractor in combination with other traditional classifeir instead of other convnet model (eg: VGG16, Xception)

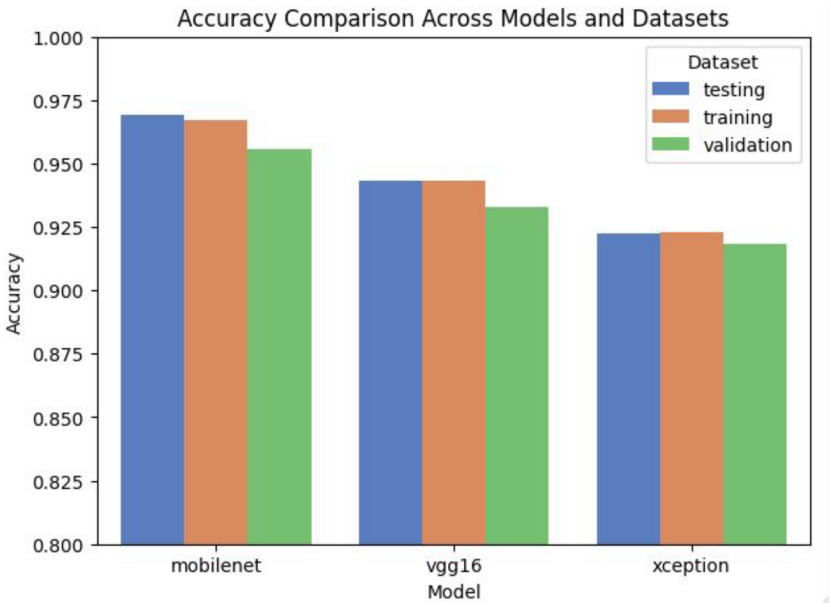

Figure 6.1: Pretrained Model Accuracy After Transfer Learning

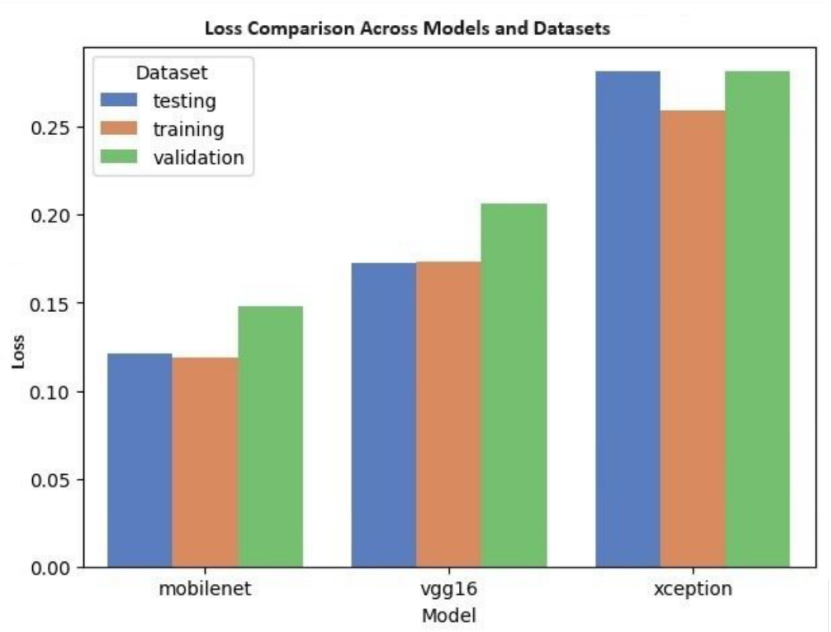

Figure 6.2: Pretrained Model Loss After Transfer Learning

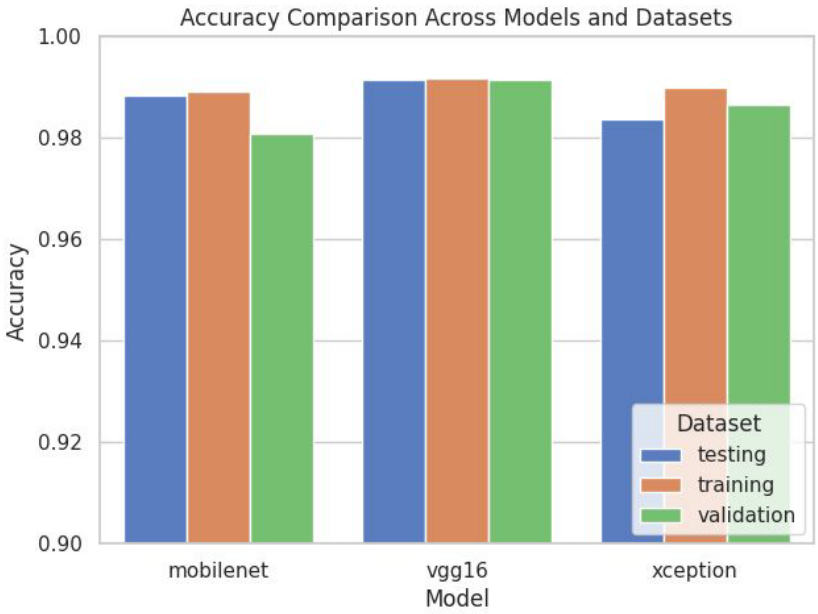

Figure 6.3: Pretrained Model Accuracy After Fine Tuning

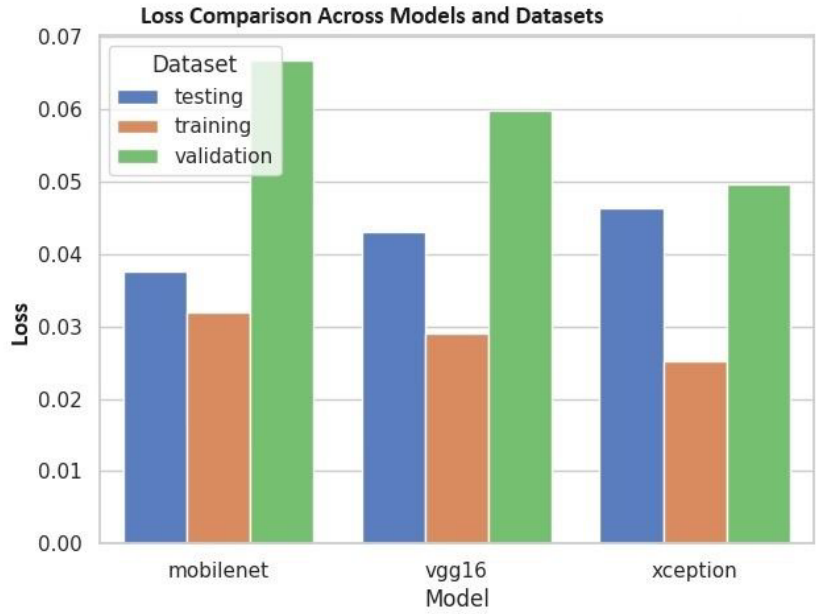

Figure 6.4: Pretrained Model Loss after Fine Tuning

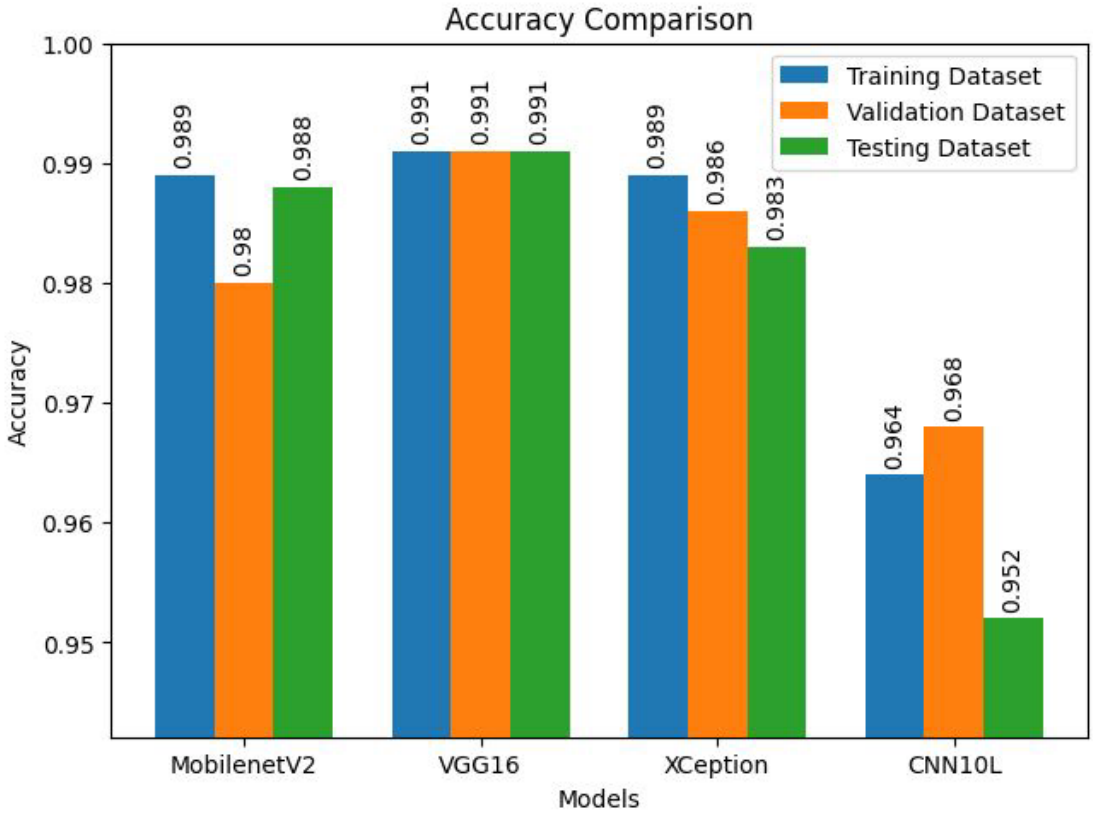

Figure 6.5: Neural Network Model Accuracy Comparisons

6.1.2 Pretrained Model Accuracy and Loss Comparison After Fine Tuning

When performing fine tuning the models, we observe that VGG16 base model, outperform other models even MobileNet. However, the difference in model accuracy with these 3 models, is very low, they only differ by about 1% in their accuracy. So, we can say they perform in par with each other.

6.1.3 Model Comparison with our CNN10L Model

When comparing our CNN10L model (model that we trained from scratch), against the three pre-trained deep learning models: MobileNetV2, VGG16, and Xception, we observed that the CNN10L model perform badly in all classification evaluation metrics but not by much. In term of accuracy metric, CNN10L get (0.9648, 0.9681, 0.952107) in comparison to the best performing model (0.9917, 0.9913, 0.9913) in training, validation and testing datasets, respectively.

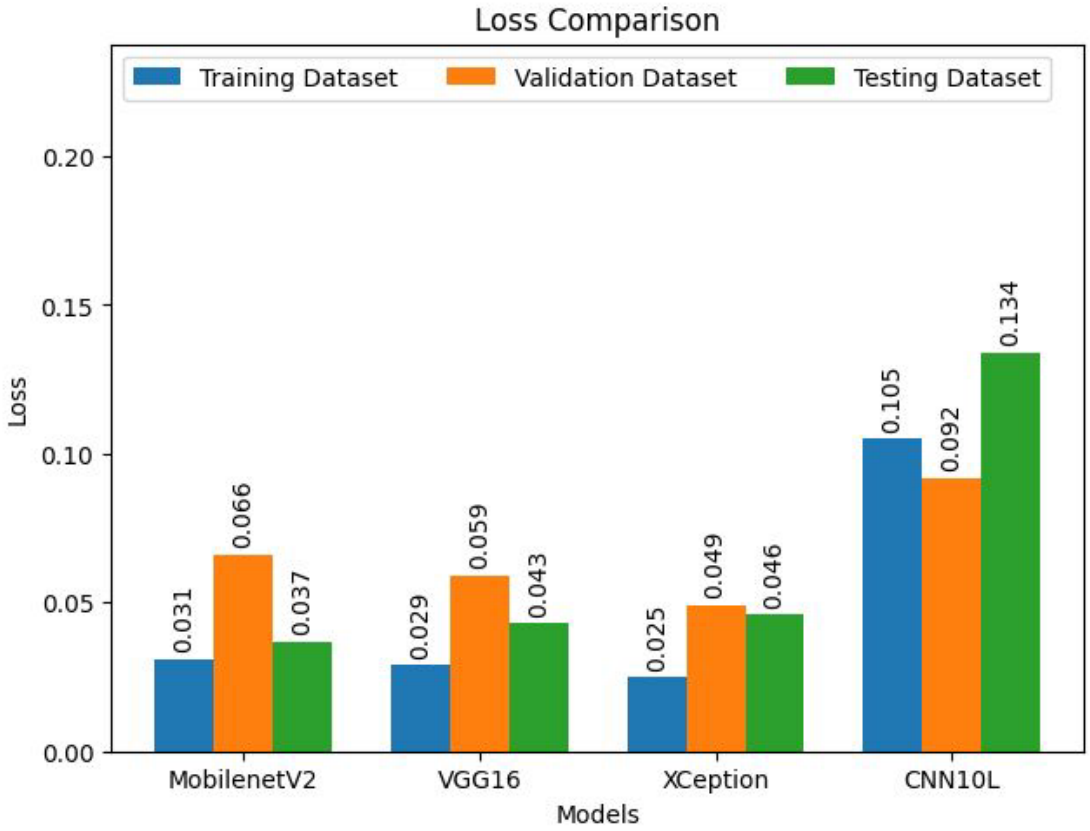

Figure 6.6: Neural Network Model Loss Comparisons

6.1.4 Comparison of all model using different evaluation metrics

In this section, we perform comparison of all the models that we used.

The highest accuracy model from our comparison was VGG16, which is a pretrained model, with 0.99 accuracy. Our custom model got an accuracy rating of 0.95. The least accurate model was a traditional model utilizing RandomForest yielding an accuracy rating of 0.84 in accuracy of testing dataset.

After these rigorous comparisons between the various models using different evaluation matrices, the consensus is that pre-trained deep learning models have a noticeable increase in performance and accuracy, over model trained from scratch and traditional models. Pre-trained models also have some increase in performance and accuracy over our custom trained model, CNN10L.

From this observation, we inferred that, the knowledge that is learned by pretrained model from large datasets like ImageNet is actually helpful in performing classification in narrow domain tasks as our scratch model, CNN10L is easily outperform by the 3 pretrained base model in term of accuracy, precision, recall and F1-score.

Among traditional classifier models, we observe that SVM and XGBoost perform the best and Random Forest is the worst-performing one. Even though CNN10L outperform traditional classifier, the difference in metrics score is very small (3% difference in accuracy metric) with the best performing traditional classifier model (SVM and XGBoost). So, we can say that these classical models are in par with neural network models and we should not completely ignore them.

Model	Measure	Classes											Accuracy
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
MobileNetV2	**Precision**	**1.00**	**1.00**	**0.96**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.86**	
	Recall	**1.00**	**1.00**	**0.96**	**1.00**	**1.00**	**1.00**	**1.00**	**0.88**	**1.00**	**1.00**	**0.99**	
	F1-Score	**1.00**	**1.00**	**0.98**	**1.00**	**1.00**	**1.00**	**1.00**	**0.88**	**1.00**	**1.00**	**0.92**	**0.99**
	Support	**63**	**62**	**26**	**164**	**150**	**85**	**105**	**65**	**119**	**116**	**85**	**1040**
VGG16	**Precision**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.71**	**1.00**	**1.00**	**1.00**	
	Recall	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.87**	
	F1-Score	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.83**	**1.00**	**1.00**	**0.93**	**0.99**
	Support	**63**	**62**	**27**	**164**	**150**	**85**	**105**	**36**	**119**	**116**	**113**	**1040**
Xception	**Precision**	**0.98**	**1.00**	**1.00**	**1.00**	**0.98**	**0.99**	**0.97**	**0.88**	**0.99**	**1.00**	**0.97**	
	Recall	**0.98**	**1.00**	**0.93**	**0.97**	**1.00**	**1.00**	**1.00**	**0.94**	**1.00**	**1.00**	**0.93**	
	F1-Score	**0.98**	**1.00**	**0.96**	**0.98**	**0.99**	**0.99**	**0.99**	**0.91**	**1.00**	**1.00**	**0.95**	**0.98**
	Support	**63**	**62**	**29**	**169**	**147**	**84**	**102**	**48**	**118**	**116**	**102**	**1040**
CNN10L	**Precision**	**0.90**	**0.97**	**1.00**	**0.98**	**0.94**	**1.00**	**0.98**	**0.80**	**1.00**	**0.86**	**1.00**	
	Recall	**0.89**	**0.98**	**0.92**	**0.98**	**0.97**	**0.96**	**0.98**	**0.77**	**1.00**	**0.89**	**0.99**	
	F1-Score	**0.89**	**0.98**	**0.96**	**0.98**	**0.95**	**0.98**	**0.98**	**0.78**	**1.00**	**0.88**	**1.00**	**0.95**
	Support	**81**	**62**	**25**	**166**	**137**	**111**	**82**	**56**	**114**	**94**	**116**	**1044**
SVM	**Precision**	**0.96**	**0.98**	**0.98**	**0.91**	**0.92**	**0.92**	**0.84**	**0.92**	**0.96**	**0.87**	**0.88**	
	Recall	**0.98**	**0.99**	**0.95**	**0.96**	**0.73**	**0.98**	**0.63**	**0.94**	**0.83**	**0.92**	**0.93**	
	F1-Score	**0.97**	**0.99**	**0.96**	**0.93**	**0.81**	**0.95**	**0.72**	**0.93**	**0.89**	**0.89**	**0.90**	**0.93**
	Support	**99**	**147**	**100**	**100**	**15**	**212**	**100**	**190**	**95**	**177**	**167**	**1402**
XGBoost	**Precision**	**0.96**	**0.98**	**0.98**	**0.90**	**0.92**	**0.92**	**0.84**	**0.92**	**0.96**	**0.87**	**0.88**	
	Recall	**0.98**	**0.99**	**0.95**	**0.96**	**0.73**	**0.98**	**0.63**	**0.94**	**0.83**	**0.92**	**0.93**	
	F1-Score	**0.97**	**0.99**	**0.96**	**0.93**	**0.81**	**0.95**	**0.72**	**0.93**	**0.89**	**0.89**	**0.90**	**0.90**
	Support	**99**	**147**	**100**	**100**	**15**	**212**	**100**	**190**	**95**	**177**	**167**	**1402**
RandomForest	**Precision**	**0.96**	**0.94**	**0.94**	**0.87**	**1.00**	**0.76**	**0.75**	**0.76**	**0.84**	**0.69**	**0.79**	
	Recall	**0.87**	**0.98**	**0.92**	**0.89**	**0.40**	**0.95**	**0.12**	**0.90**	**0.61**	**0.81**	**0.86**	
	F1-Score	**0.91**	**0.96**	**0.93**	**0.88**	**0.57**	**0.84**	**0.21**	**0.82**	**0.71**	**0.74**	**0.82**	**0.84**
	Support	**99**	**147**	**100**	**100**	**15**	**212**	**100**	**190**	**95**	**177**	**167**	**1402**
KNN	**Precision**	**1.00**	**0.94**	**0.92**	**0.96**	**0.82**	**0.84**	**0.85**	**0.96**	**0.79**	**0.82**	**0.70**	
	Recall	**0.92**	**0.99**	**0.98**	**0.91**	**0.93**	**0.96**	**0.52**	**0.81**	**0.78**	**0.86**	**0.95**	
	F1-Score	**0.96**	**0.96**	**0.95**	**0.93**	**0.87**	**0.90**	**0.65**	**0.88**	**0.78**	**0.84**	**0.81**	**0.88**
	Support	**99**	**147**	**100**	**100**	**15**	**212**	**100**	**190**	**95**	**177**	**167**	**1402**

Table 6.1: Performance metrics (rounding to 2 decimal places) for various models.

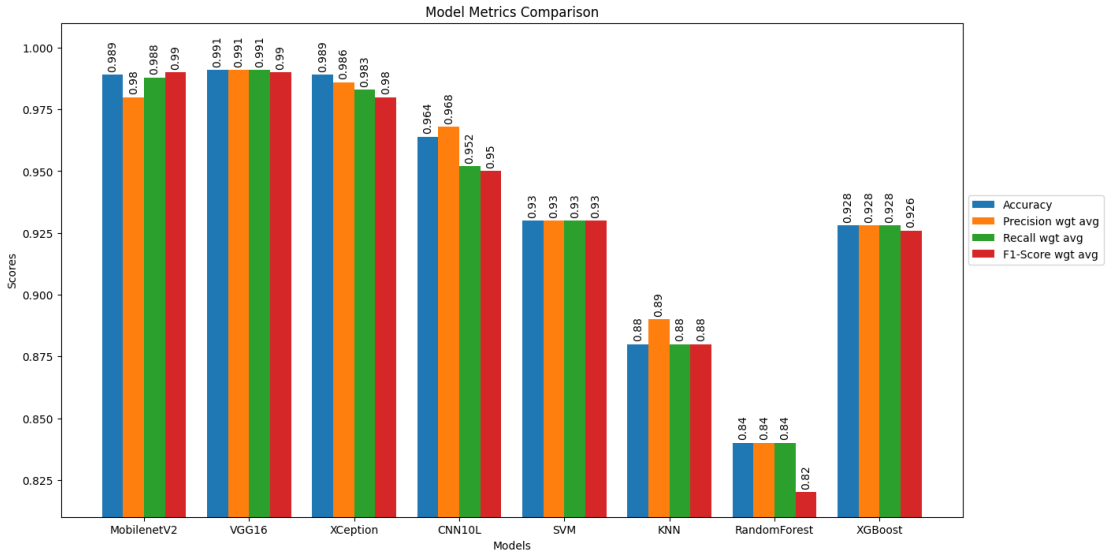

Figure 6.7: All Model Evaluation Metrics

6.1.5 Confusion Matrix Produced by Models on Testing Datasets

MobileNetV2 Base Model

VGG16 Base Model

Xception Base Model

CNN10L Model

MobileNetV2 + SVM Model

MobileNetV2 + KNN Model

MobileNetV2 + Random Forest Model

MobileNetV2 + XGBoost Model

6.1.6 Model Prediction

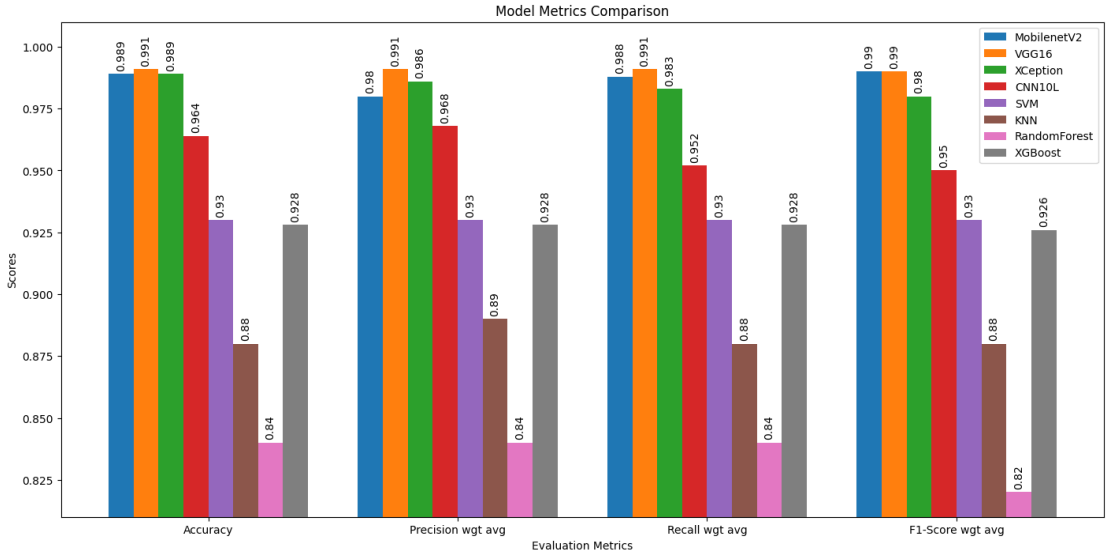

Figure 6.8: All Model Evaluation Metrics Group By Metrics Types

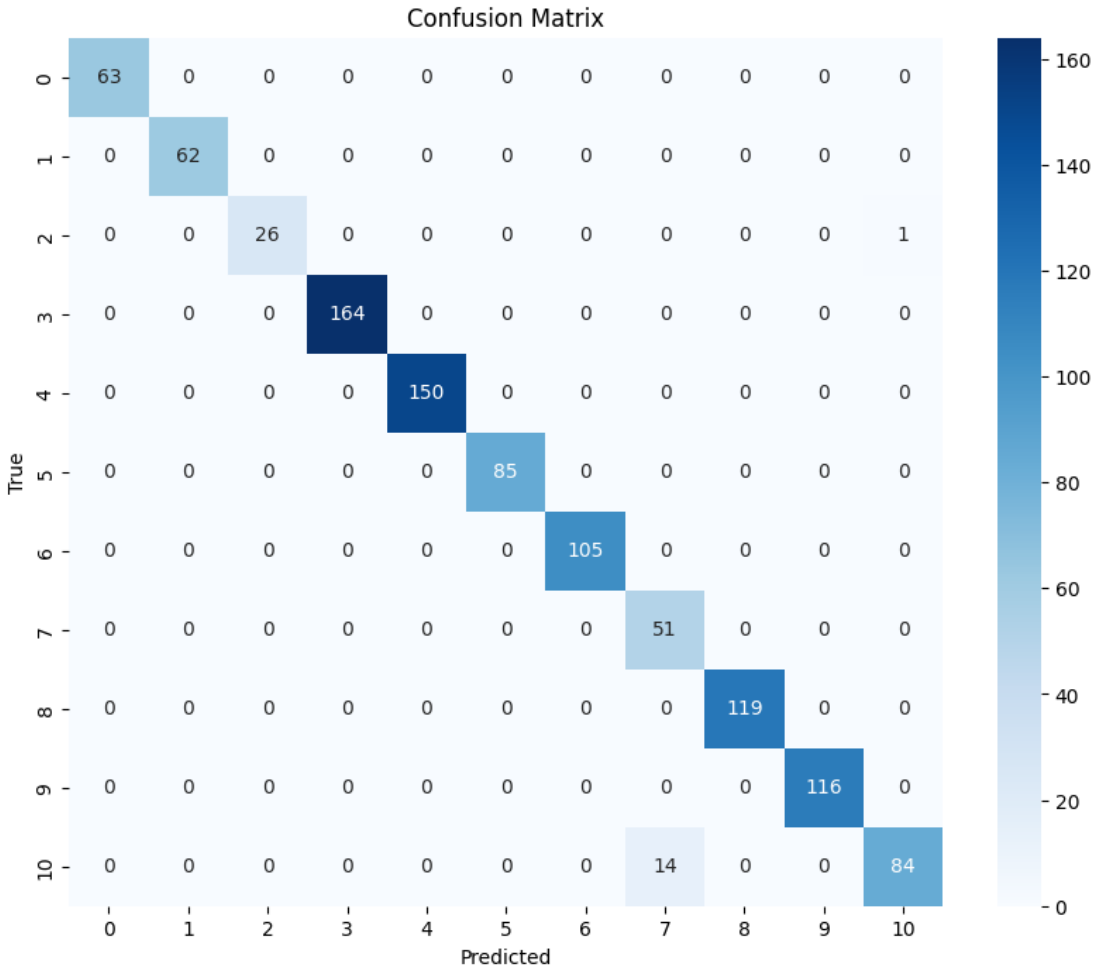

Figure 6.9: Confusion Matrix of MobileNetV2

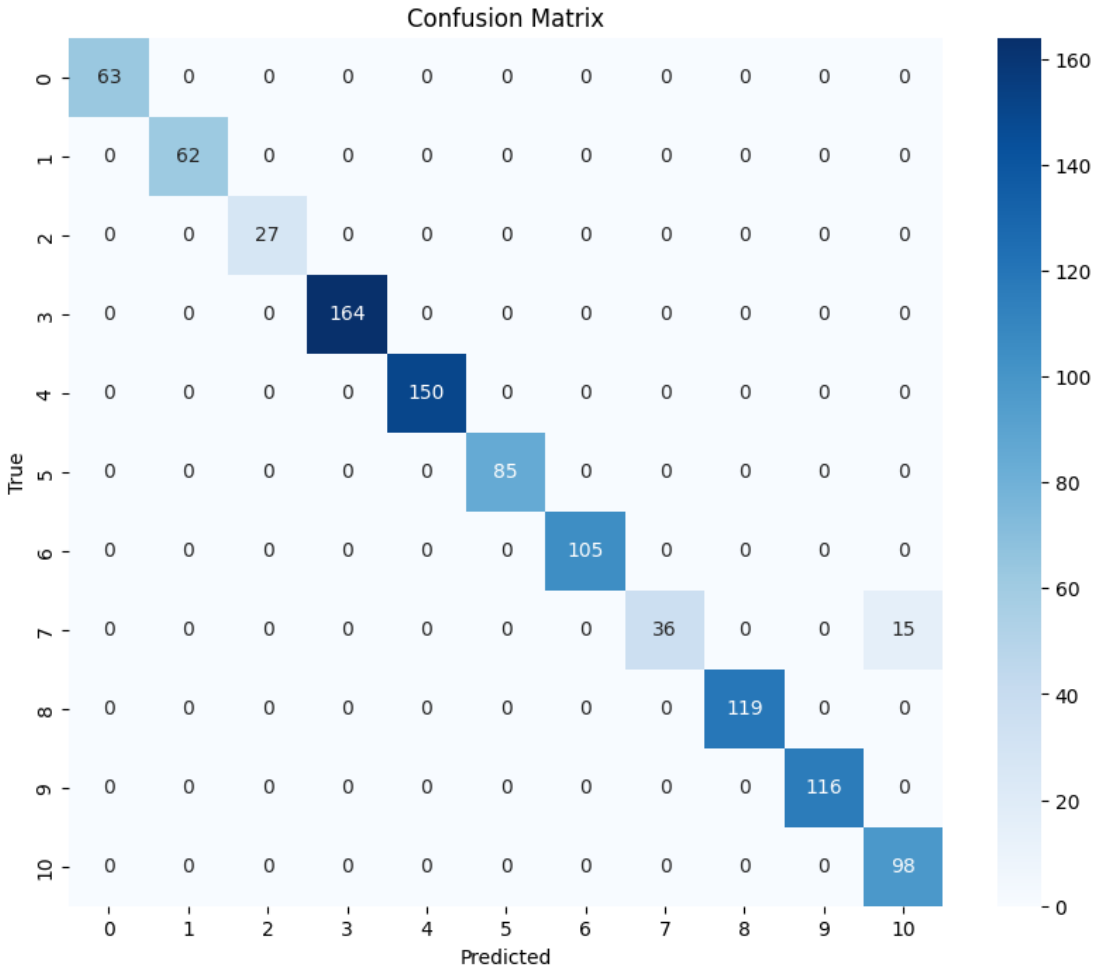

Figure 6.10: Confusion Matrix of VGG16

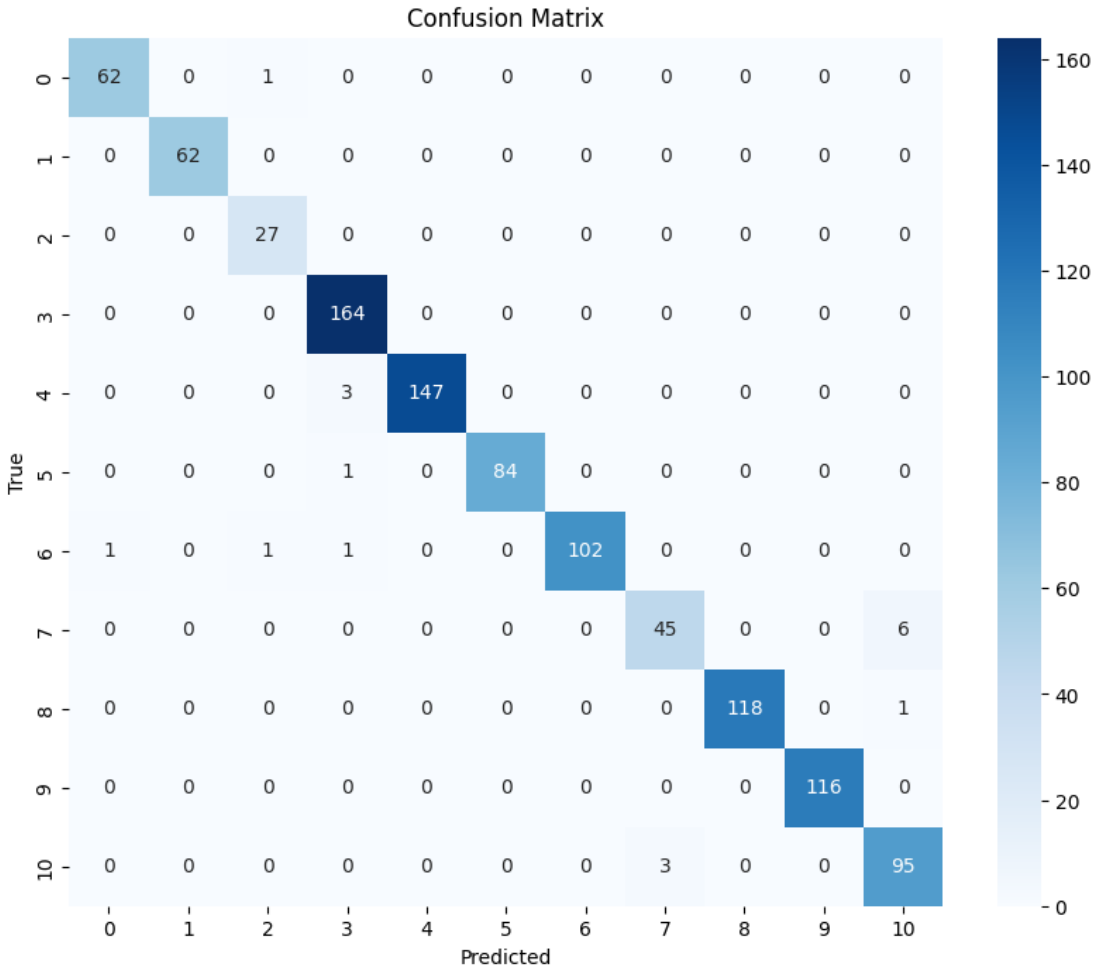

Figure 6.11: Confusion Matrix of Xception

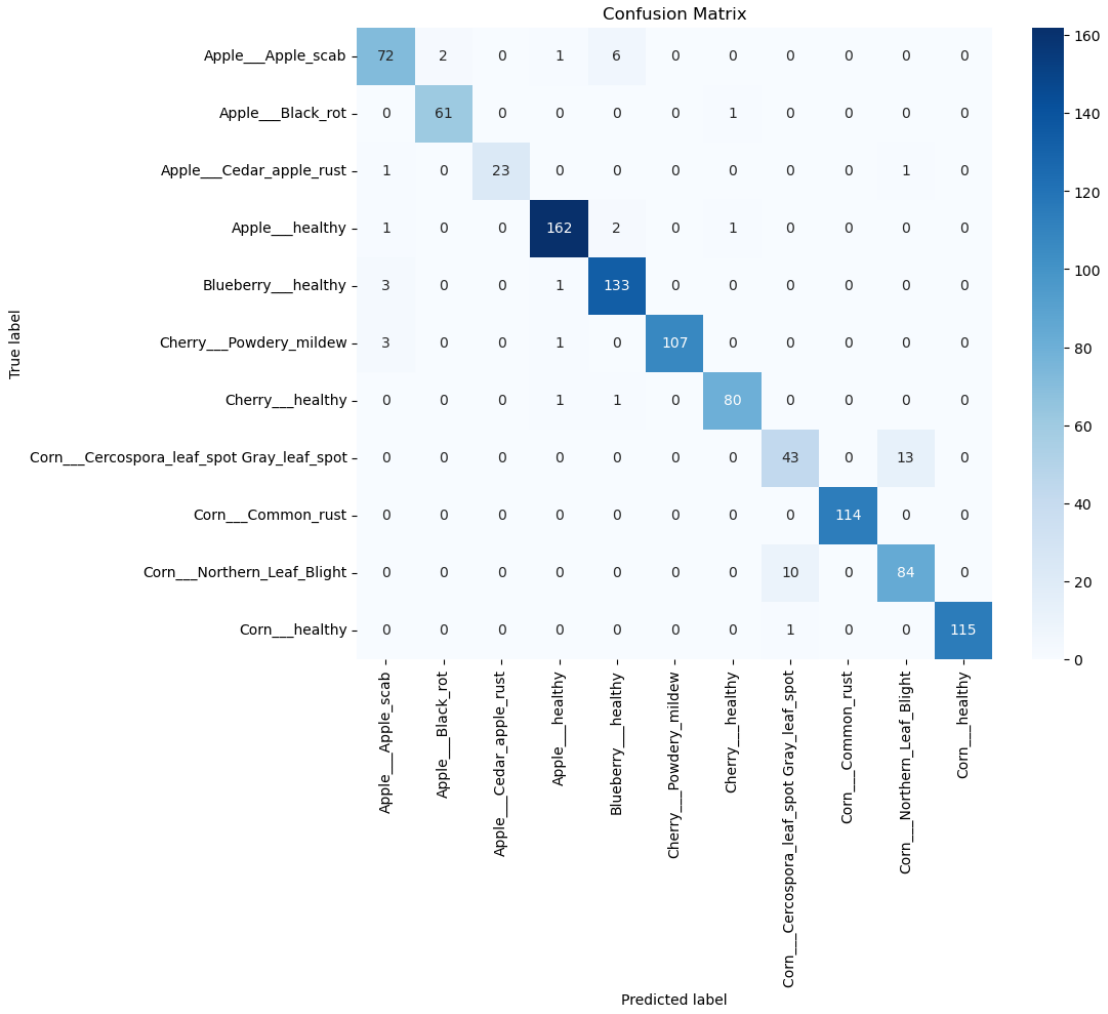

Figure 6.12: Confusion Matrix of CNN10L Model

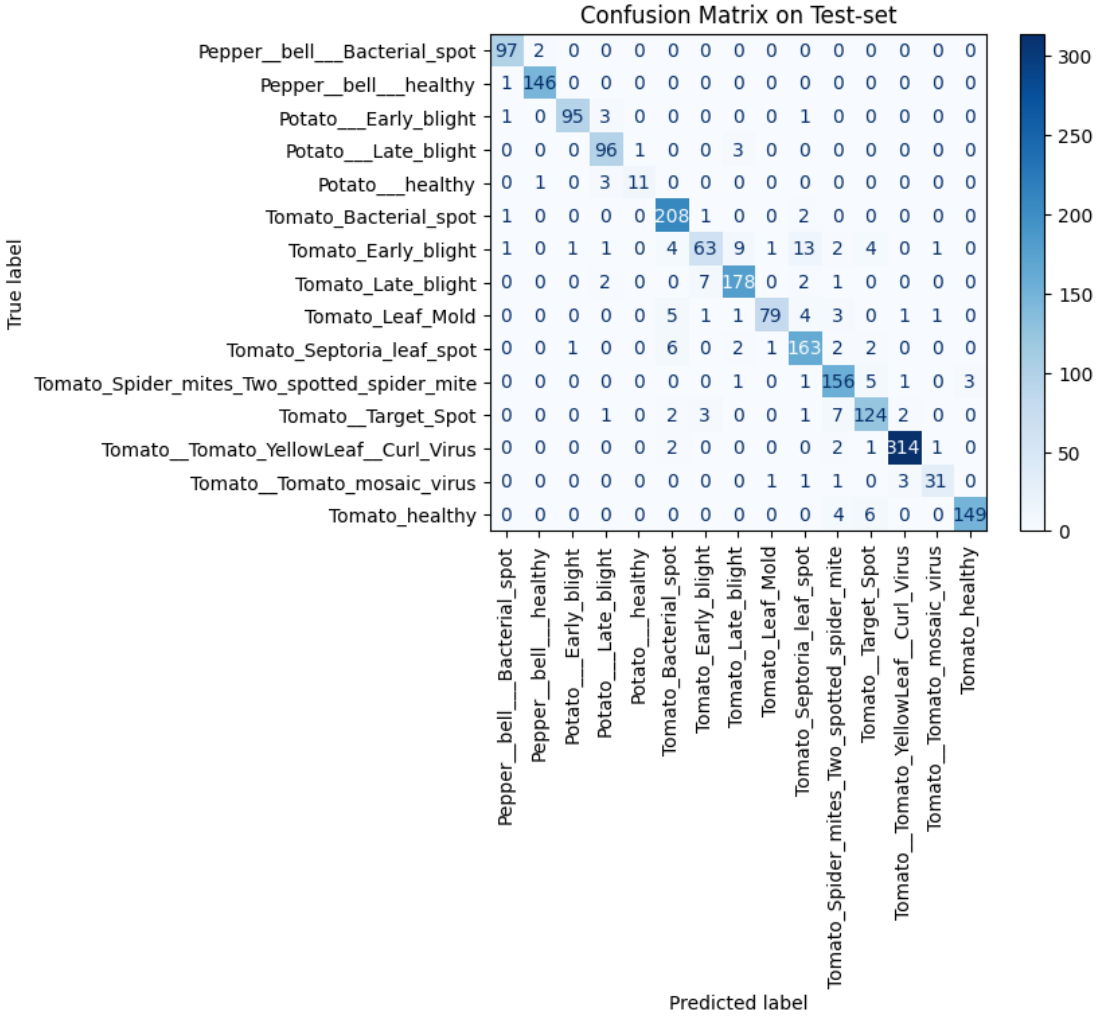

Figure 6.13: Confusion Matrix of MobileNetV2 + SVM Model

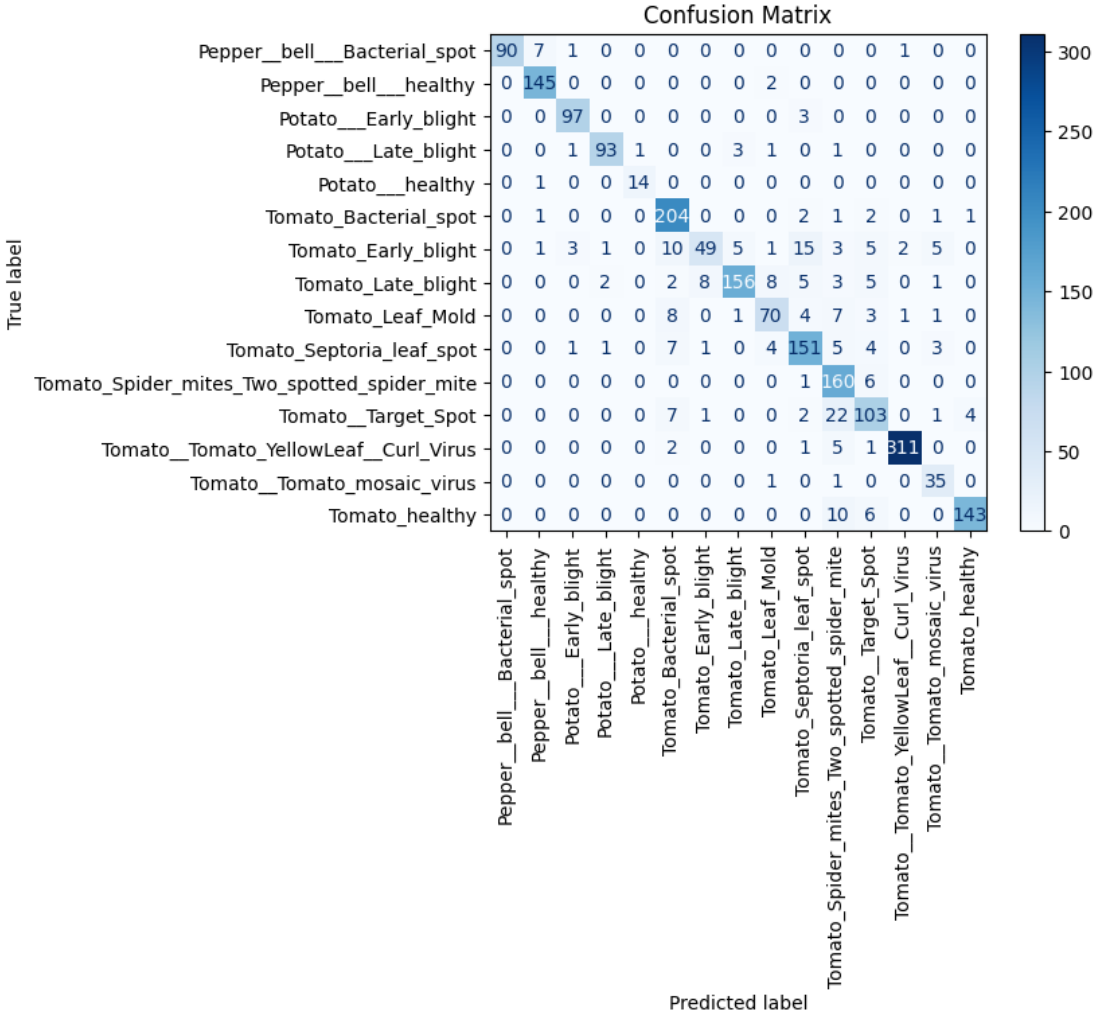

Figure 6.14: Confusion Matrix of MobileNetV2 + KNN

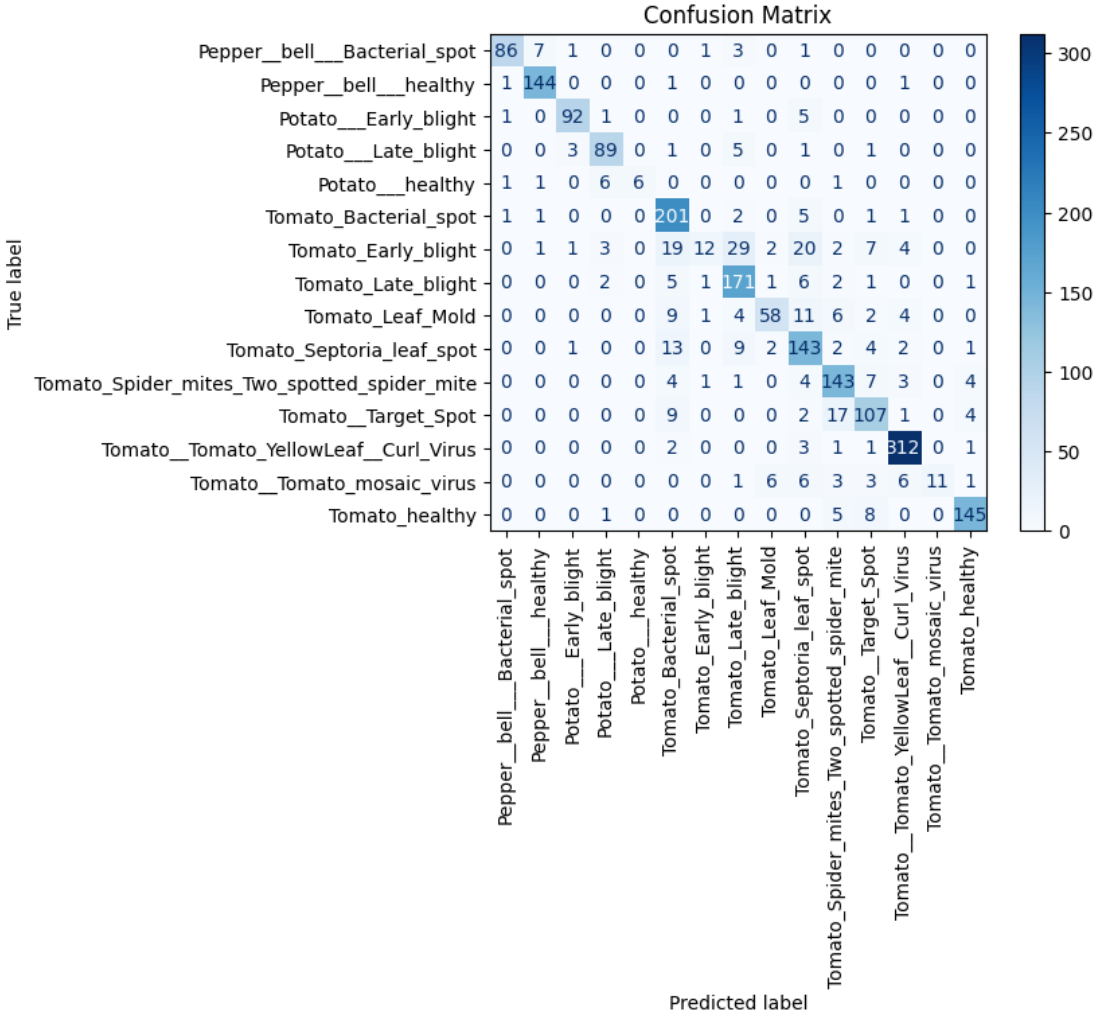

Figure 6.15: Confusion Matrix of MobileNetV2 + Random Forest

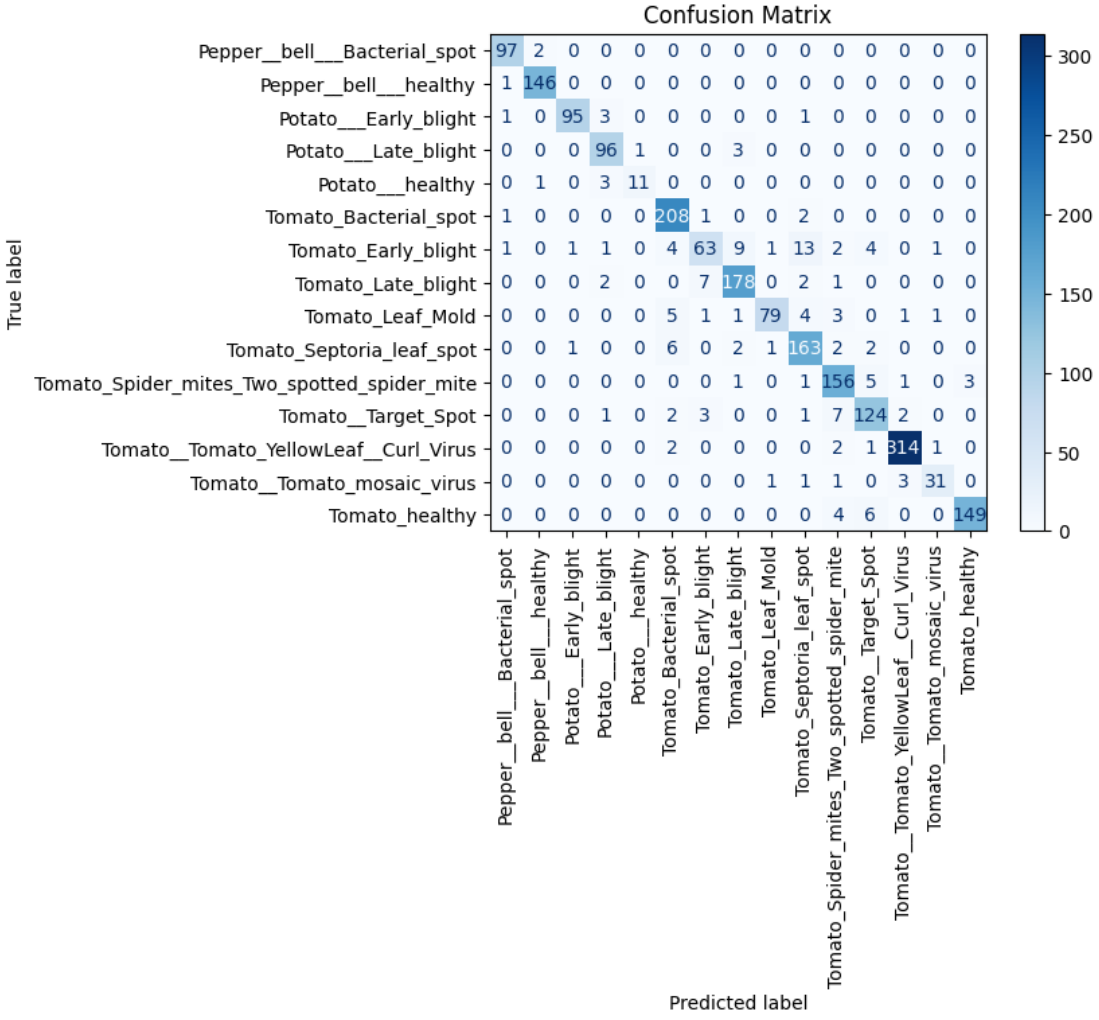

Figure 6.16: Confusion Matrix of MobileNetV2 + XGBoost

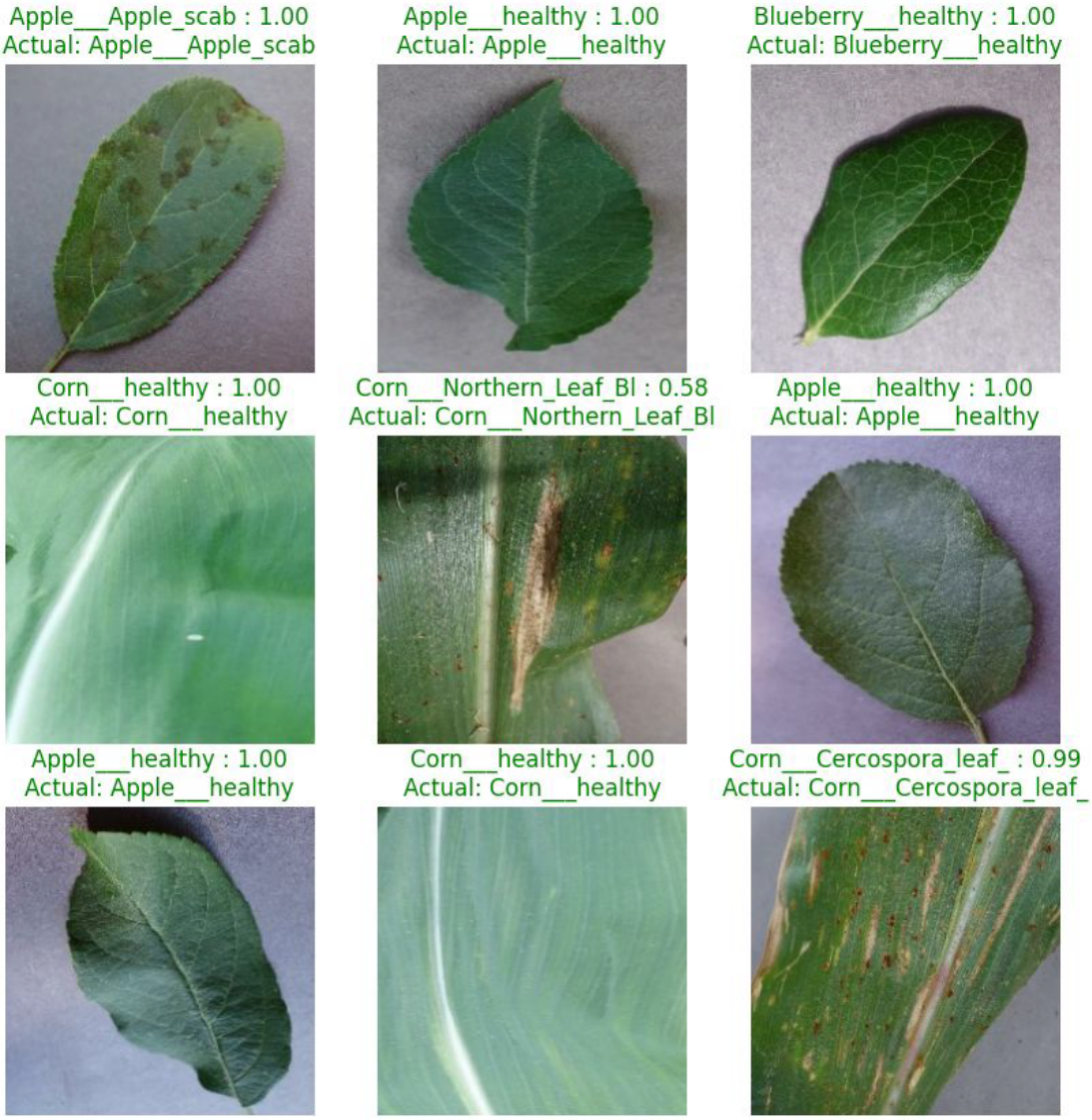

Figure 6.17: VGG16 Model Prediction

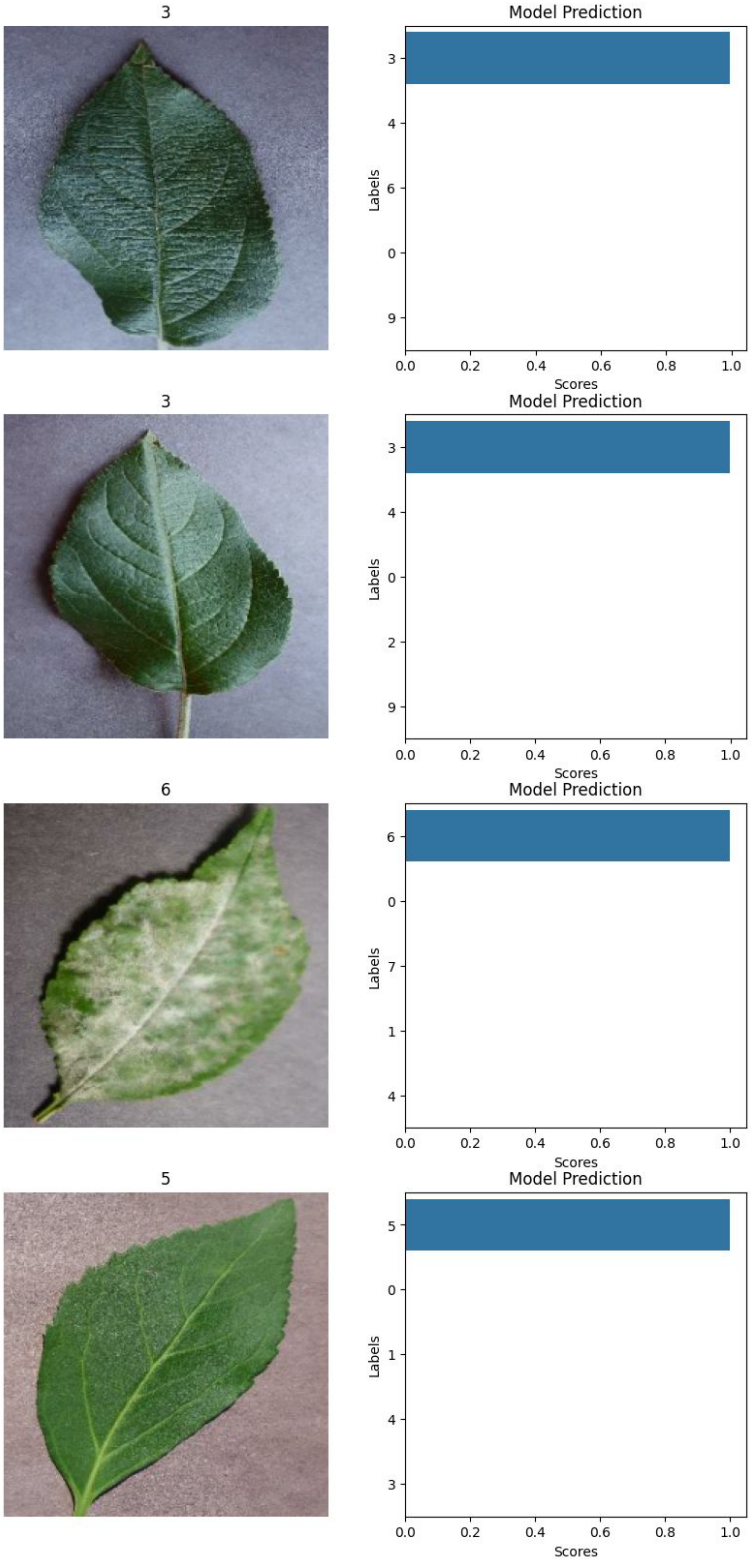

Figure 6.18: VGG-16 Model Prediction - Sample1

Chapter 7

Model Interpretation and Visualisation

This chapter delves into the interpretability of our models, offering insights into the decision-making processes of the neural networks. By visualizing model filters, feature maps, and utilizing techniques such as PCA and t-SNE, we aim to elucidate how different models perceive and classify plant disease images. Furthermore, we explore advanced interpretability methods, including saliency maps and nearest neighbor analysis, to provide a comprehensive understanding of the underlying mechanisms driving model predictions. Through these analyses, we aim to not only compare the effectiveness of different models but also enhance the transparency and explainability of our best-performing model (VGG16 Base Model).

1. Visualizing CNN Layers - Techniques for Visualization (Filter, Feature Maps)
2. Visualizing features map with a masked image and a normal one
3. Visualizing features map of images of different classes
4. Other Model Interpretation Techniques - PCA, T-SNE, Nearest Neighbors, Saliency Maps

7.1 Model Interpretation

Model interpretation at heart, is to find out ways to understand model decision making policies better. This is to enable fairness, accountability and transparency which will give humans enough confidence to use these models in

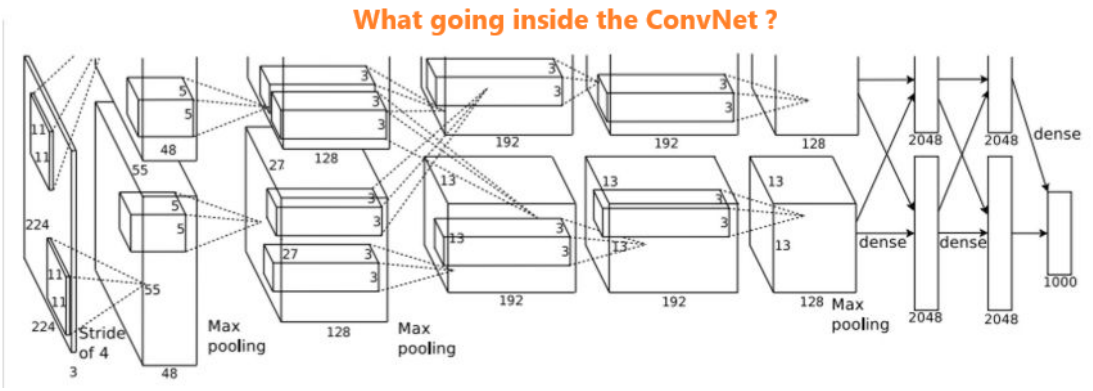

Figure 7.1: Importance of Model Interpretation

real-world problems which a lot of impact to business and society. Hence, there are techniques which have existed for a long time now, which can be used to understand and interpret models in a better way. These can be grouped under the following two major categories.

1. Exploratory analysis and visualization techniques like clustering and dimensionality reduction.

2. Model performance evaluation metrics like precision, recall, accuracy, ROC curve and the AUC (for classification models) and the coefficient of determination (R-square), root mean-square error, mean absolute error (for regression models).

In our research, we'll be focussing more on visualisation part such as visualising feature space using dimensionality reduction technique, feature map visualisation, etc.

Model Interpretation is very important to reduce biases in our training data. For Example: in saliency map fig 7.33, we are using a dog breed classifier to predict the dog classes but the model instead of focusing on the dog features to classify wolf or not, it actually focus on the background of the picture and predict wolf if the background contain snow and not-wolf if no snow is present. So, using appropriate model interpretation techniques (here, using a saliency map) we can find such biases in our model and get a glimpse of how the model is making the predictions without taking our model as a complete black box.

7.1.1 Understanding Machine Learning Model Interpretation

Machine Learning has seen widespread industry adoption only in the last couple of years. Hence, model interpretation as a concept is still mostly theoretical and subjective. Any machine learning model at its heart has a response function which tries to map and explain relationships and patterns between the independent (input) variables and the dependent (target or response) variable(s).

When a model predicts or finds our insights, it takes certain decisions and choices. Model interpretation tries to understand and explain these decisions taken by the response function i.e., the what, why and how. The key to model interpretation is transparency, the ability to question, and the ease of understanding model decisions by humans. The three most important aspects of model interpretation are explained as follows.

1. What drives model predictions? We should have the ability to query our model and find out latent feature interactions to get an idea of which features might be important in the decision-making policies of the model. This ensures fairness of the model.

2. Why did the model take a certain decision? We should also be able to validate and justify why certain key features were responsible in driving certain decisions taken by a model during predictions. This ensures accountability and reliability of the model.

3. How can we trust model predictions? We should be able to evaluate and validate any data point and how a model takes decisions on it. This should be demonstrable and easy to understand for key stakeholders that the model works as expected. This ensures transparency of the model.

Interpretability also popularly known as human-interpretable interpretations (HII) of a machine learning model is the extent to which a human (including non-experts in machine learning) can understand the choices taken by models in their decision-making process (the how, why and what).

When comparing models, besides model performance, a model is said to have a better interpretability than another model if its decisions are easier to understand by a human than the decisions from the other model. [35]

7.1.2 The Importance of Machine Learning Model Interpretation

When tackling machine learning problems, data scientists often have a tendency to fixate on model performance metrics like accuracy, precision and recall and so on (This is important no doubt!). This is also prevalent in most online competitions around data science and machine learning. However, metrics only tell a part of the story of a model's predictive decisions. Over time, the performance might change due to model concept drift caused by various factors in the environment. Hence, it is of paramount importance to understand what drives a model to take certain decisions.

Some of us might argue if a model is working great why bother digging deeper? Always remember that when solving problems in the real-world, for the business to trust your model predictions and decisions, they will keep asking the question, "Why should I trust your model?" and this makes perfect sense. Would you be satisfied with a model just predicting and taking decisions (the what) like if a person has cancer or diabetes, if a person might be a risk to society or even if a customer will churn? Maybe not, we might prefer it more if we could know more about the model's decision process (the why and how). This gives us more transparency into why the model makes certain decisions, what might go wrong in certain scenarios and over time it helps us build a certain amount of trust on these machine learning models. [35]

7.2 Visualising First Layer Filters

In this section, we perform visualization of the first layer filter learned by the VGG16 base model and CNN10L model.The first layer filters are critical as they capture basic features such as edges, textures, and colors, which are foundational for subsequent layers.

Observations of Fine Tune VGG16 Base Model Filter:

- **Edge and Texture Detection:**
  The first layer filters of the fine-tuned VGG16 model prominently detect edges and basic textures. The filters are well-defined, showing a variety of orientations and frequencies, which are essential for capturing the structure of the input images.
  The filters are appear to be less diverge as compare to each other, most of them seems to have the same structure but differ in their orientation, flip,

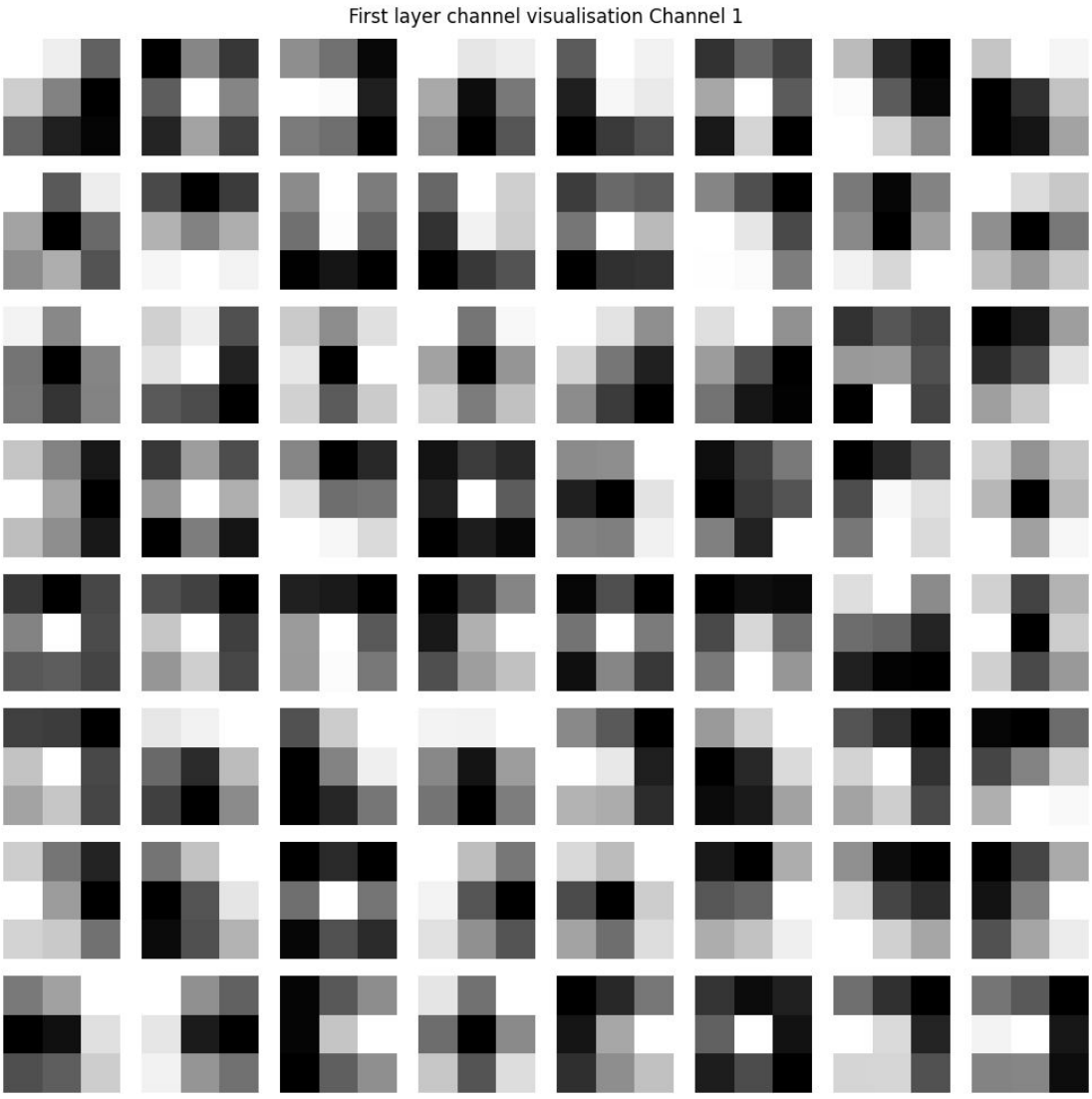

Figure 7.2: VGG16 First Layer First 64 Filters

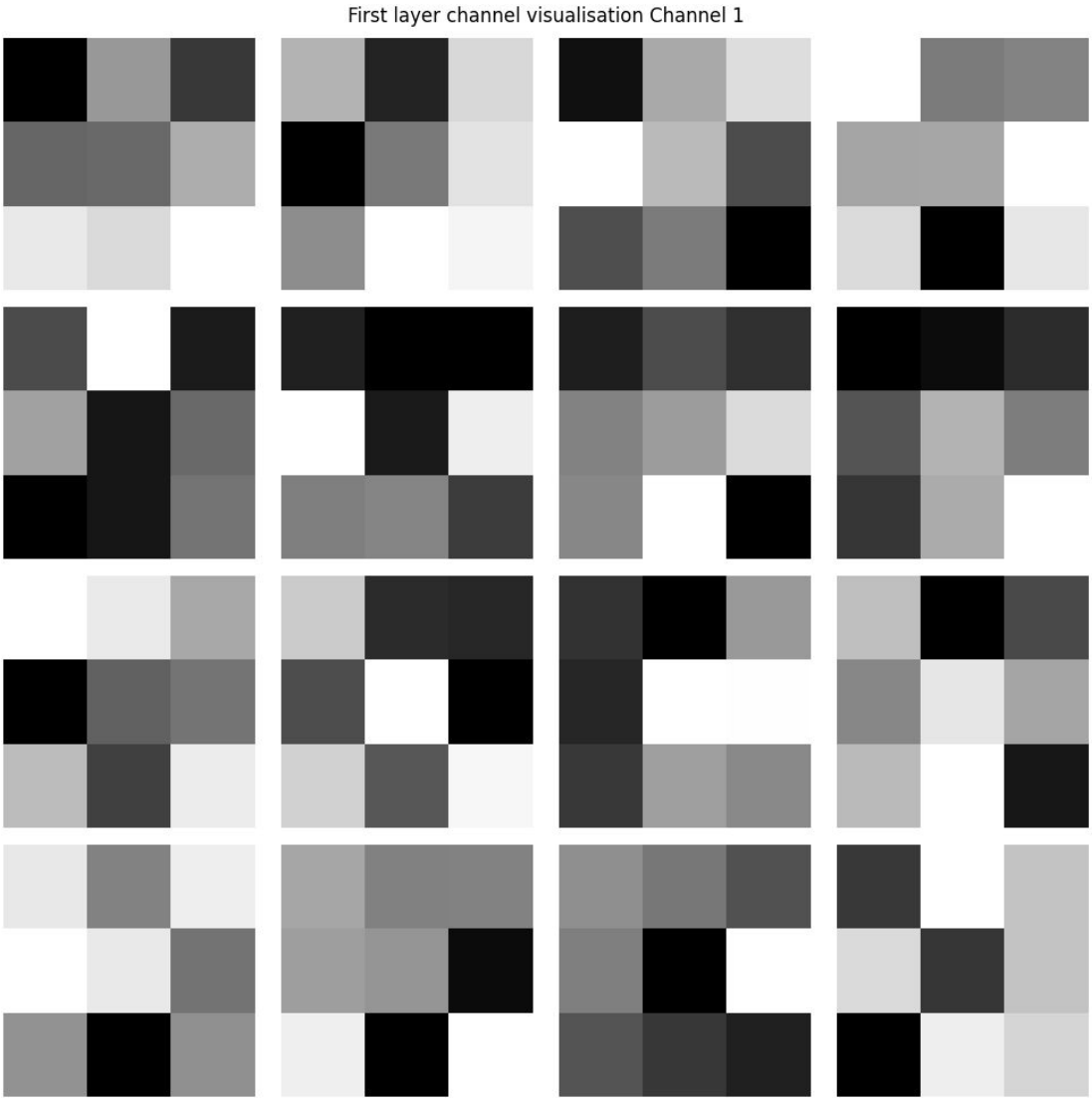

Figure 7.3: CNN10L First Layer 16 Filters

rotate or are complete opposite of each other.
For eg: in fig: 7.2 filter of row-6 and col-5; and filter of row-8 and col-8 are same. Similarly, for filter of (4-row, 4-col) and of (8, 6); (8, 1) and (8, 2) seems to have the same structure but differ in their orientation. Filter of (1, 1), (3, 8) and (6, 8) also have similar structure but have different orientation, many such filters can be pick out from the figure 7.2
These foundational features contribute to the model's ability to accurately classify different diseases by building on these basic patterns in deeper layers.

Observations of Fine Tune CNN10L Model Filter:

- **Edge and Basic Pattern Detection:**
  Similar to VGG16, the first layer filters of the CNN10L model also detect edges, lines, textures, gradients and basic patterns. However, the filters appear more diverse and slightly less defined compared to those of the fine-tuned VGG16 model.
  The filters show a wide range of structure but may not capture as wide a variety of textures and colors as effectively as by VGG16 model.
  From the fig: 7.3, we observe that filter in (3, 3) and in fig: 7.2, filter in (2, 3) seems to have similar structure but differ in their orientation.

- **Specialization:**
  Due to small number of filters present in the first layer and since it is trained only on Plant village datasets, the filter are less structured and specialised on the plant village datasets, so, they'll have low generalisation capability compare to vgg models. The filters of the CNN10L model are somewhat specialized to the specific characteristics of the Plant Village dataset so, even though they do capture basic patterns, their diverse nature of filter suggests that the model might rely more on specific patterns present in the training data, which can limit generalization compared to VGG16.

7.3 Features Map Visualisation

7.3.1 Feature Map Visualisation of Apple Black Rot

To interpret the feature extraction process at different depths of the fine-tuned VGG16 model, we visualized the feature maps at layers 2, 5, 9, 13, and 17. These layers represent increasing levels of abstraction, from basic edge detection to complex pattern recognition.

Observation:

Figure 7.4: Apple Scab

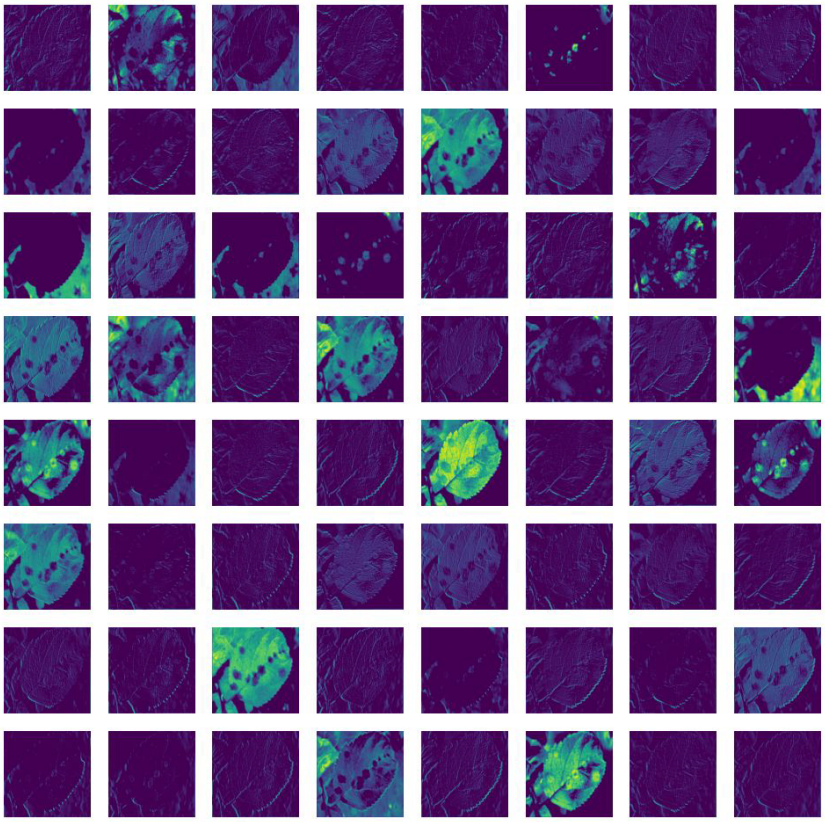

Figure 7.5: Feature Map of Apple Scab in Layer 2 produced by fine-tuned VGG16 base model

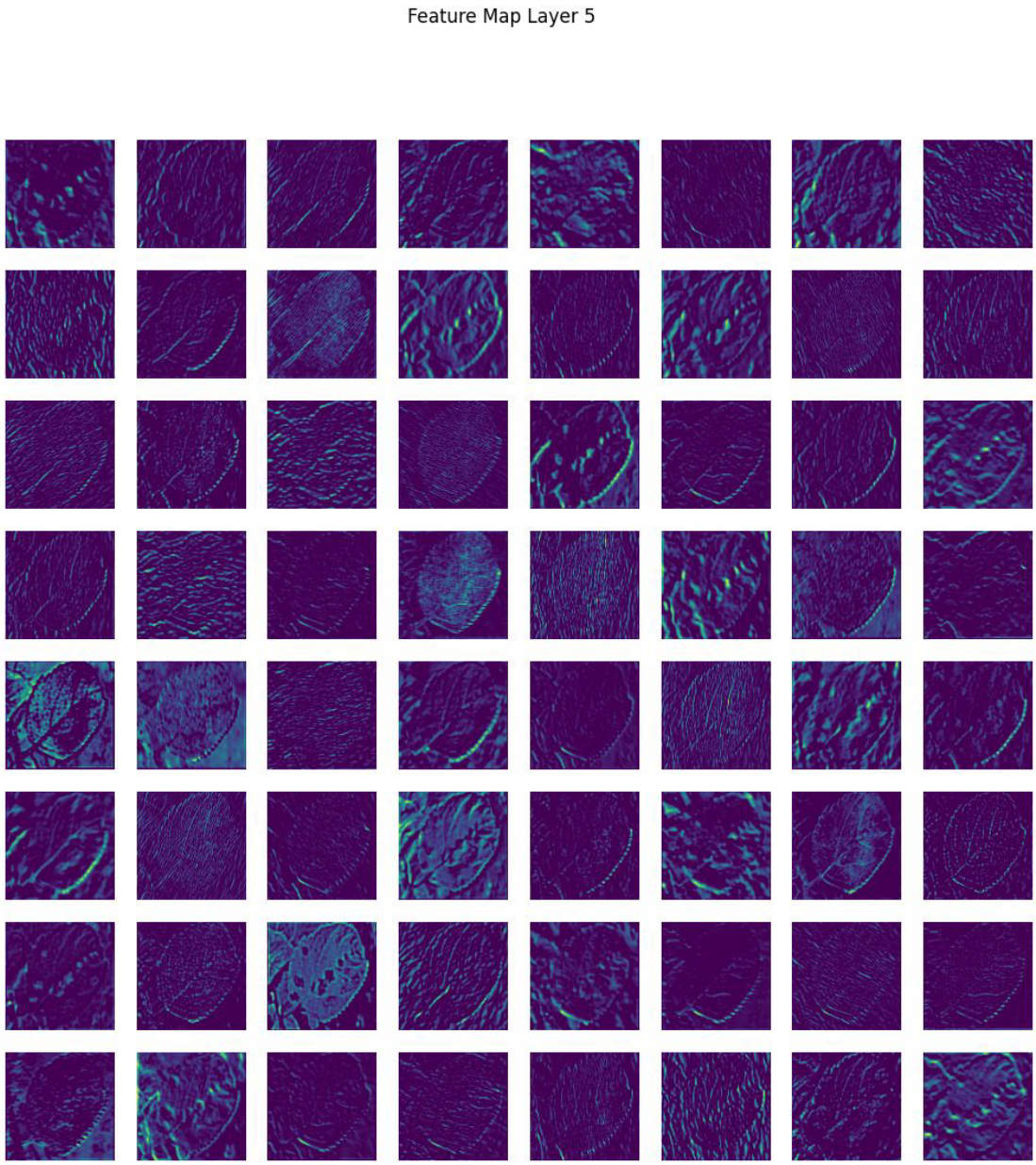

Figure 7.6: Feature Map of Apple Scab in Layer 5 produced by fine-tuned VGG16 base model

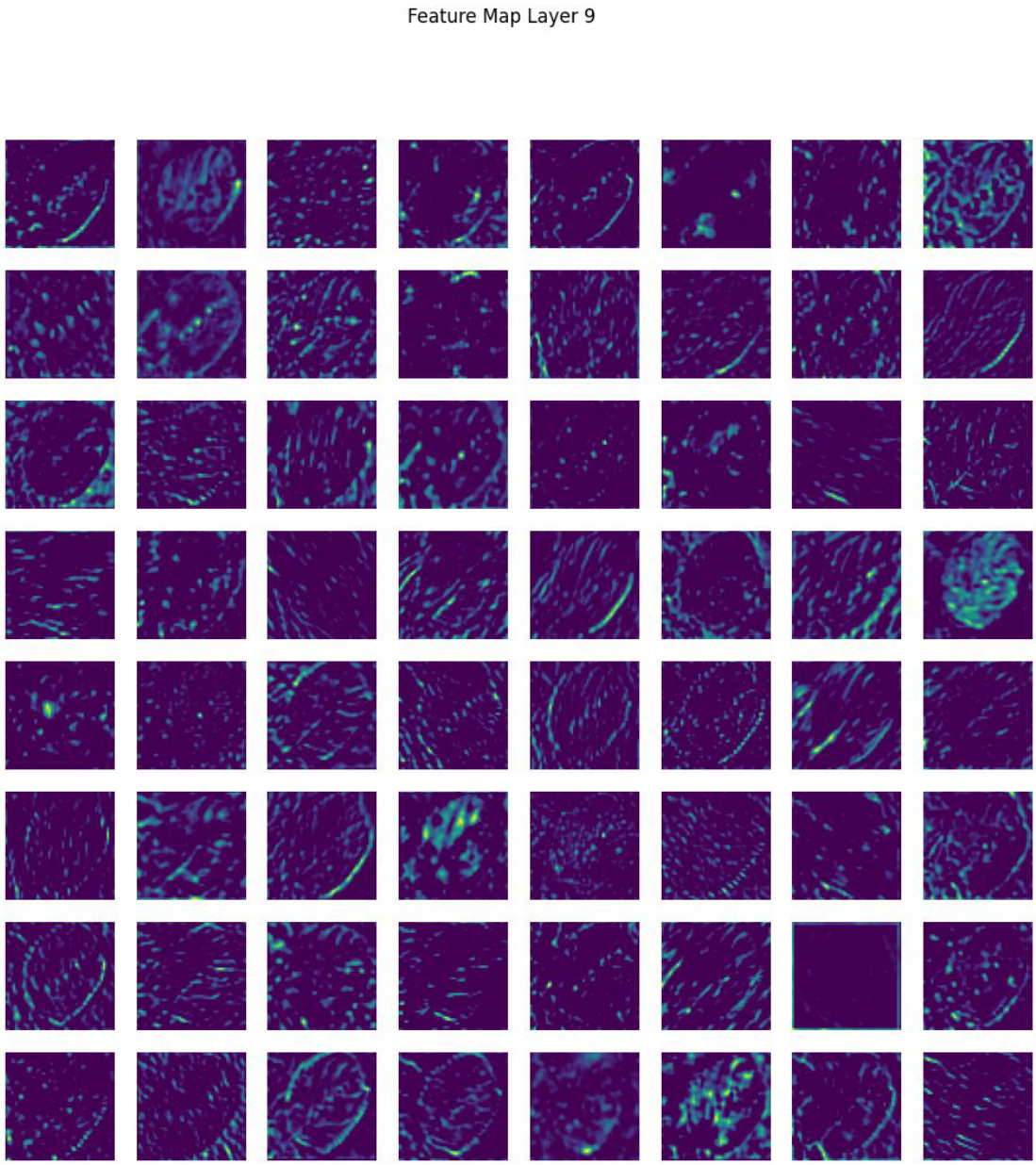

Figure 7.7: Feature Map of Apple Scab in Layer 9 produced by fine-tuned VGG16 base model

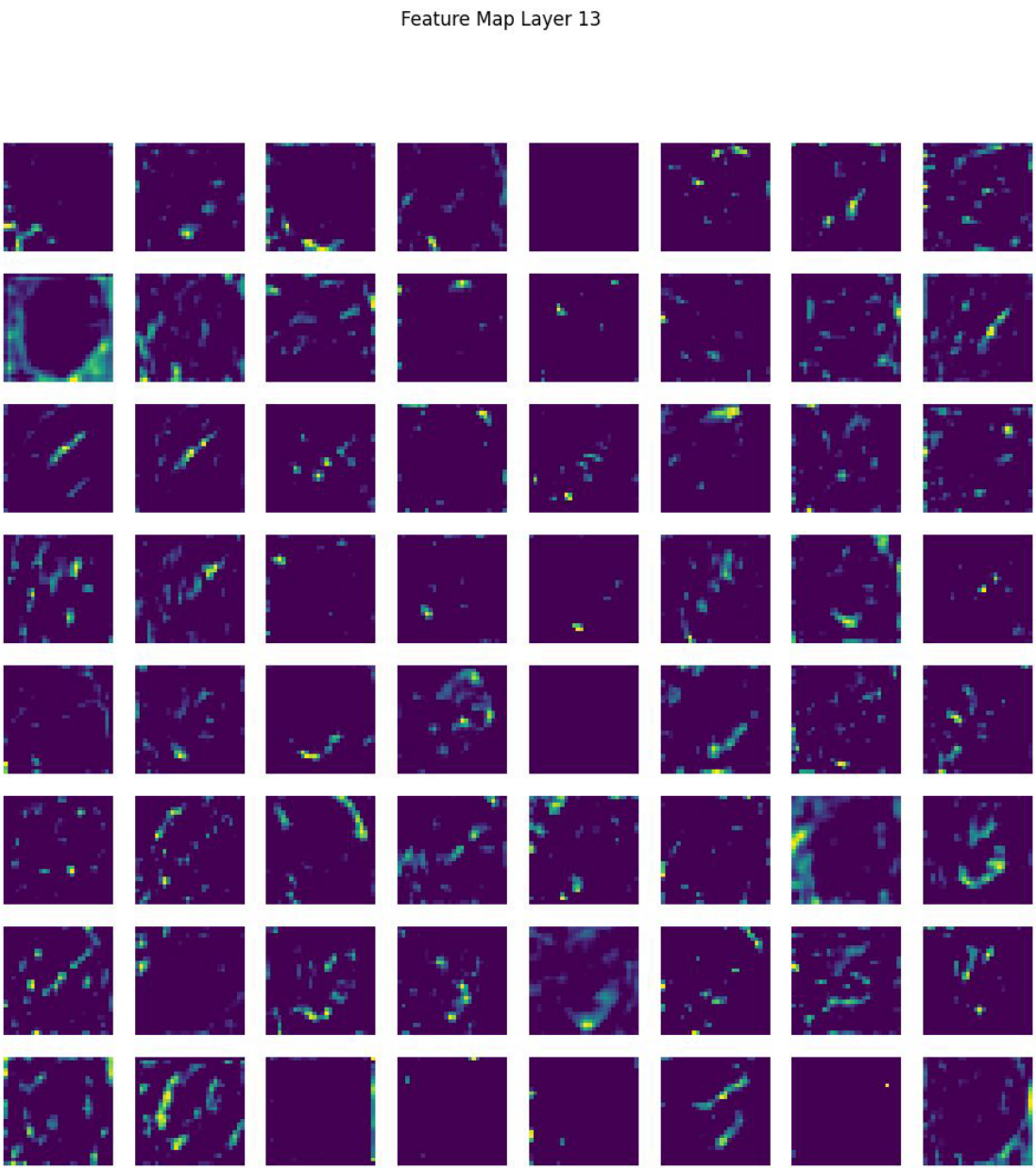

Figure 7.8: Feature Map of Apple Scab in Layer 13 produced by fine-tuned VGG16 base model

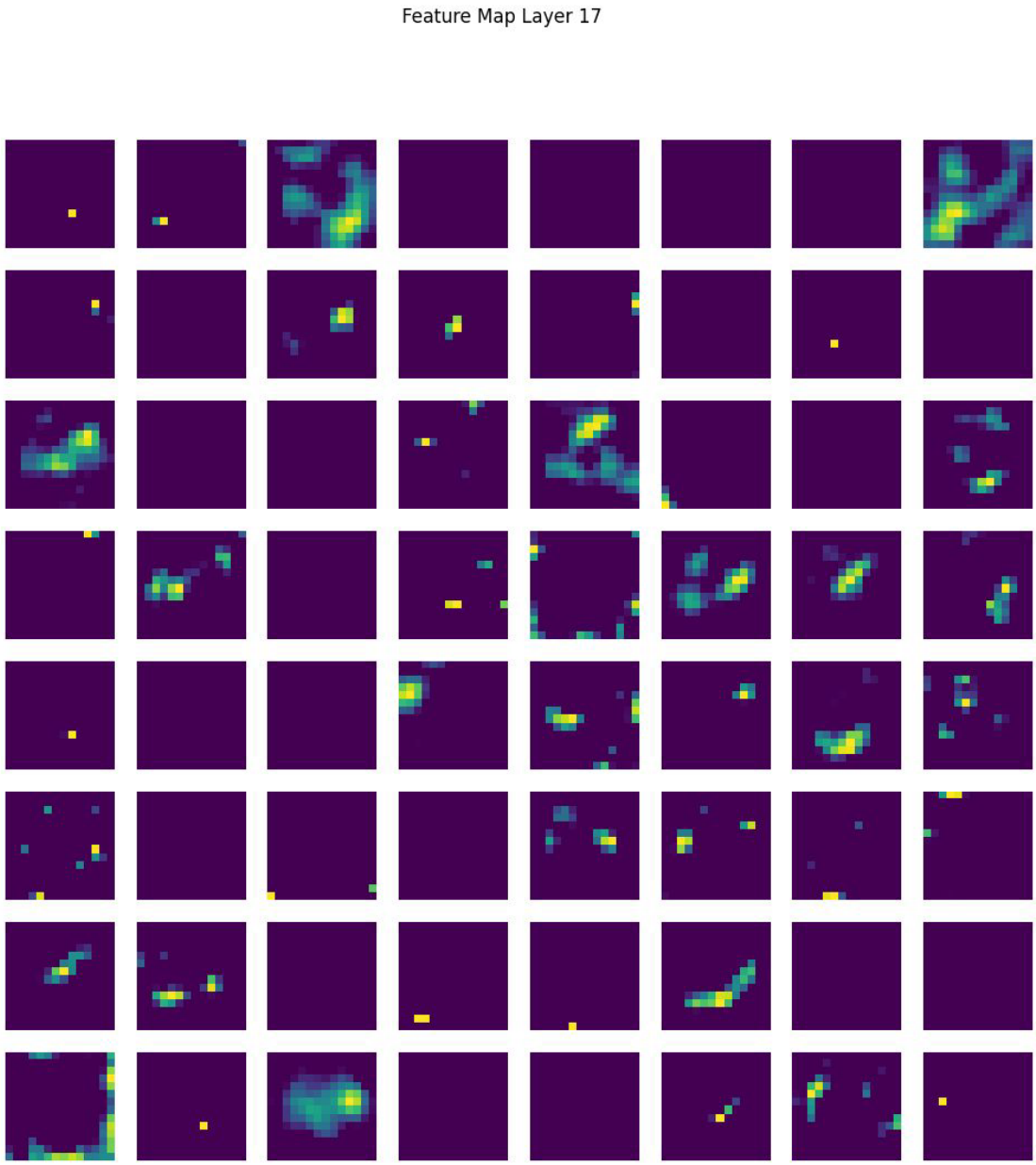

Figure 7.9: Feature Map of Apple Scab in Layer 17 produced by fine-tuned VGG16 base model

- **Layer2**
  Basic Edge and Texture Detection: The feature maps at this early layer primarily capture simple edges and textures. The activation patterns highlight basic structures in the input images, such as contours and edges of leaves. This layer serves as the foundation for more complex feature extraction in subsequent layers.

- **Layer5**
  Enhanced Texture and Shape Recognition: By layer 5, the feature maps show more complex textures and shapes. The model begins to detect patterns such as leaf veins and surface textures, which are more intricate than basic edges. These features are crucial for distinguishing between different types of diseases that affect the texture and surface appearance of leaves.

- **Layer9**
  Intermediate Patterns and Details: The feature maps at layer 9 capture intermediate patterns and details. The activations are more abstract, representing combinations of edges, textures, and shapes detected in earlier layers. This layer contributes to the model's ability to recognize more detailed and specific features of diseased leaves.

- **Layer13**
  Complex Structures and Patterns: At layer 13, the feature maps show complex structures and patterns, indicating a high level of abstraction. The activations capture specific disease characteristics such as spots, discolorations, and irregular shapes. This depth allows the model to differentiate between subtle differences in disease presentations.

- **Layer17**
  High-Level Feature Integration: The feature maps at layer 17 exhibit high-level feature integration. The activations represent very abstract and complex features that combine all previous layers' information. This layer is critical for final classification, as it integrates all detected patterns and structures to make a decision about the disease class.

7.3.2 Feature Map Comparison of Apple Scab and Apple Black Rot

In this section, to gain deeper insights into the feature extraction capabilities of our fine-tuned VGG16 model, we visualized and compared the feature maps for two distinct diseases: apple scab and apple black rot. This comparison aims to

highlight how the model distinguishes between different disease characteristics at various layers.

Observations:

- **Apple Scab:**
  The feature maps for apple scab reveal strong activations around the edges and surface spots characteristic of the disease. At intermediate layers, the model captures the distinct textures and discolorations typical of apple scab. In deeper layers, these features integrate to form a comprehensive representation, focusing on the scab's unique pattern and severity. This indicates that the model effectively learns to recognize the specific visual cues associated with apple scab, enabling precise classification.

- **Apple Black Rot:**
  For apple black rot, the feature maps show activations concentrated around the dark, rotted areas of the leaf. Early layers detect the edges of the rotted regions, while intermediate layers highlight the contrasting textures and colors associated with the rot. In the deepest layers, the model integrates these features to form a distinct representation of black rot, emphasizing the lesion's shape, color, and spread. This demonstrates the model's ability to capture and differentiate the unique visual signatures of black rot, ensuring accurate disease identification.

7.4 Feature Map comparison with Mask Image

To further interpret the feature extraction capabilities of our fine-tuned VGG16 model, we visualized the feature maps using both a normal and a masked image. The masked image had a portion of the leaf covered with a grey mask to understand how the model handles occluded information.

Observation:

- **Normal Image:**
  When visualizing the feature maps of the normal image at various layers (2, 5, 9, 13, and 17), the model consistently captured detailed patterns, textures, and structures of the leaf. The feature maps showed strong activations corresponding to disease-specific areas, such as spots and discolorations, indicating that the model effectively identifies and processes relevant features for accurate classification.

Figure 7.10: Apple Black Rot

- **Masked Image:**
  With the masked image, the feature maps at the same layers exhibited a notable reduction in activation in the masked region. However, the model still maintained strong activations in the unmasked areas. This suggests that while the mask affected the detection of features in the occluded region, the model was still able to extract and rely on the visible features to make informed predictions. This robustness highlights the model's ability to handle incomplete information and still focus on relevant, disease-specific characteristics present in the visible parts of the image.

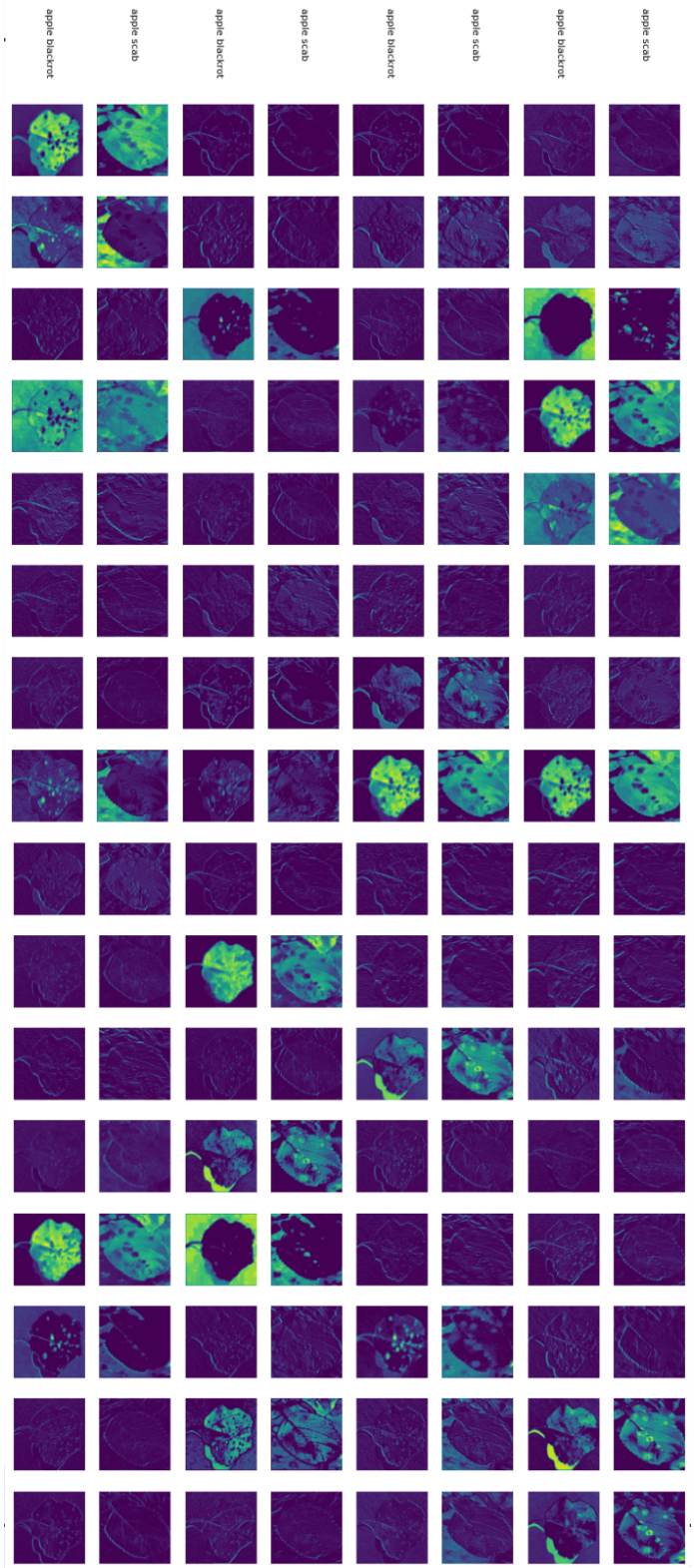

Figure 7.11: VGG16 base model Layer-2, first 64 Feature Map Comparison with Apple Scab and Apple Black Rot image

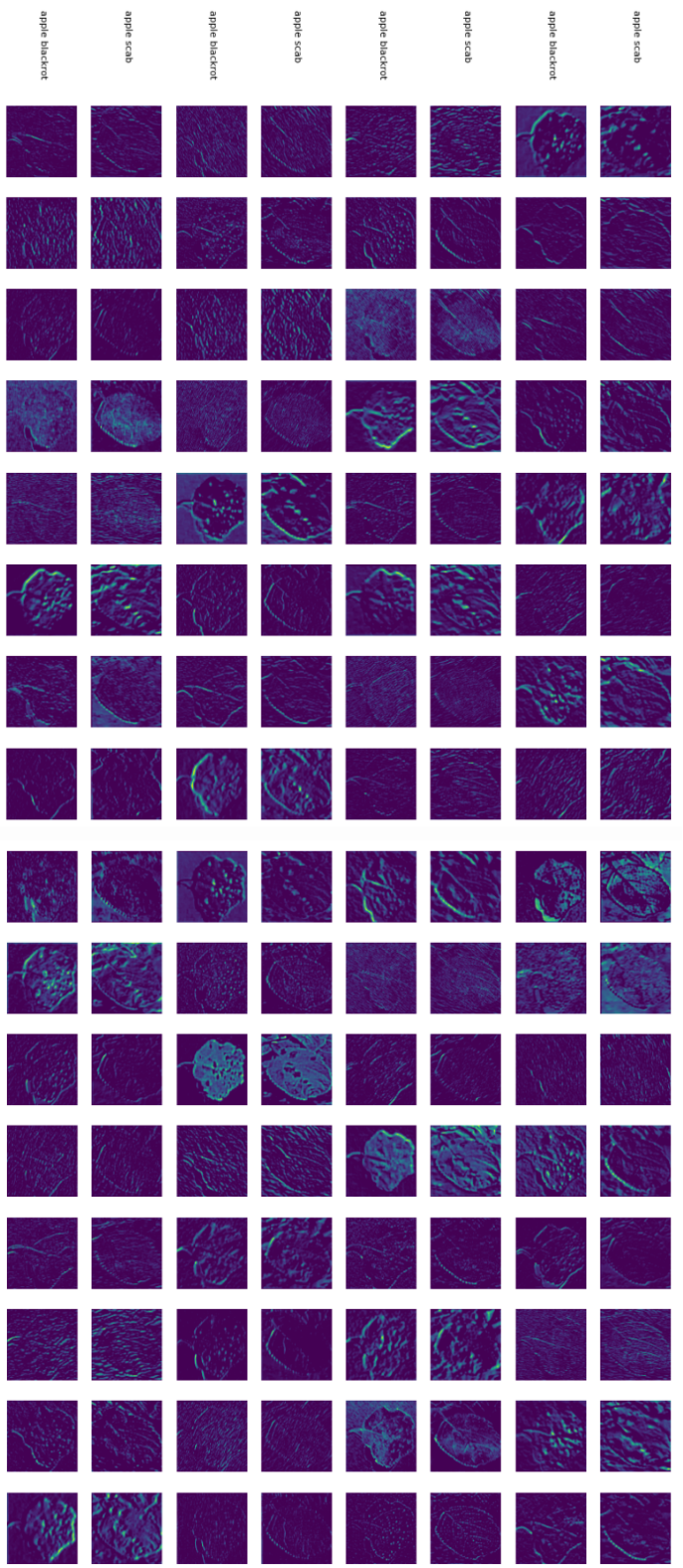

Figure 7.12: VGG16 base model Layer-5, first 64 Feature Map Comparison with Apple Scab and Apple Black Rot image

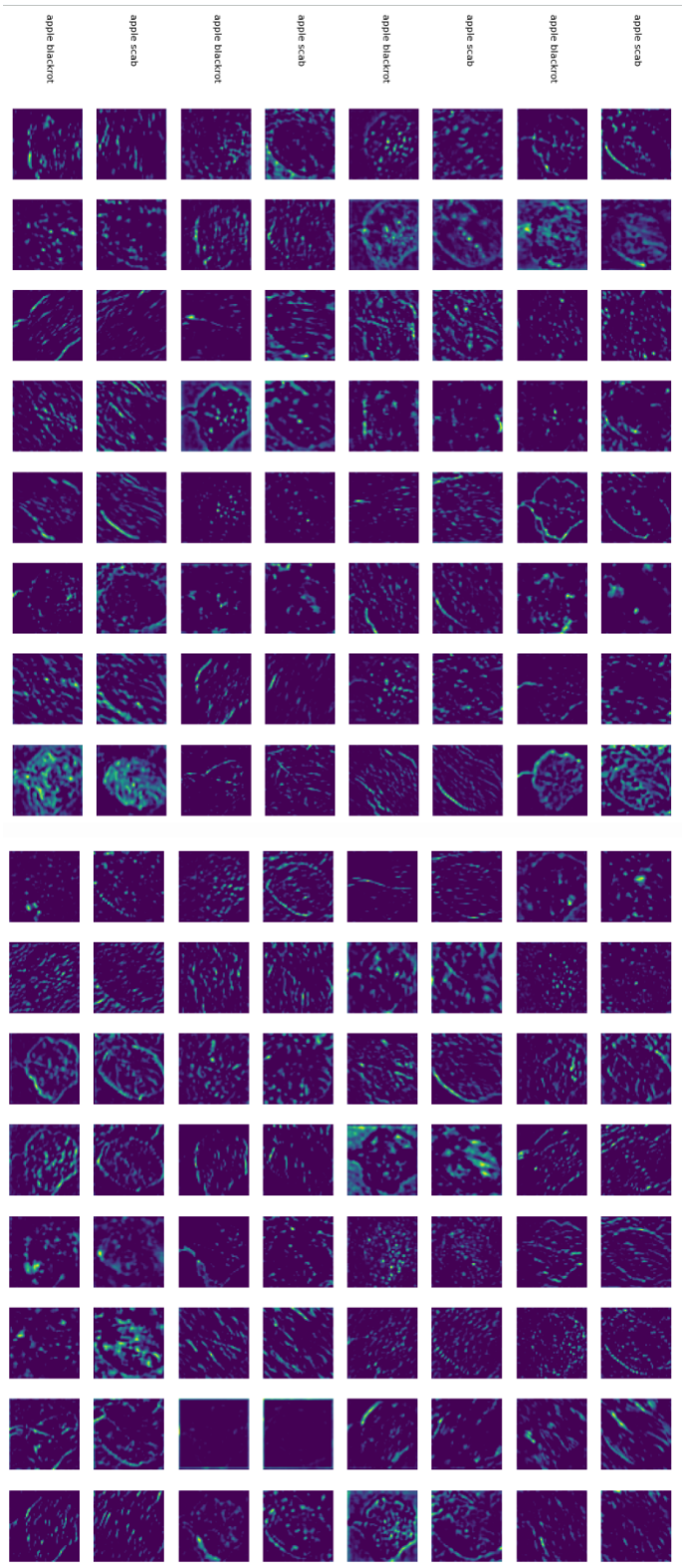

Figure 7.13: VGG16 base model Layer-9, first 64 Feature Map Comparison with Apple Scab and Apple Black Rot image

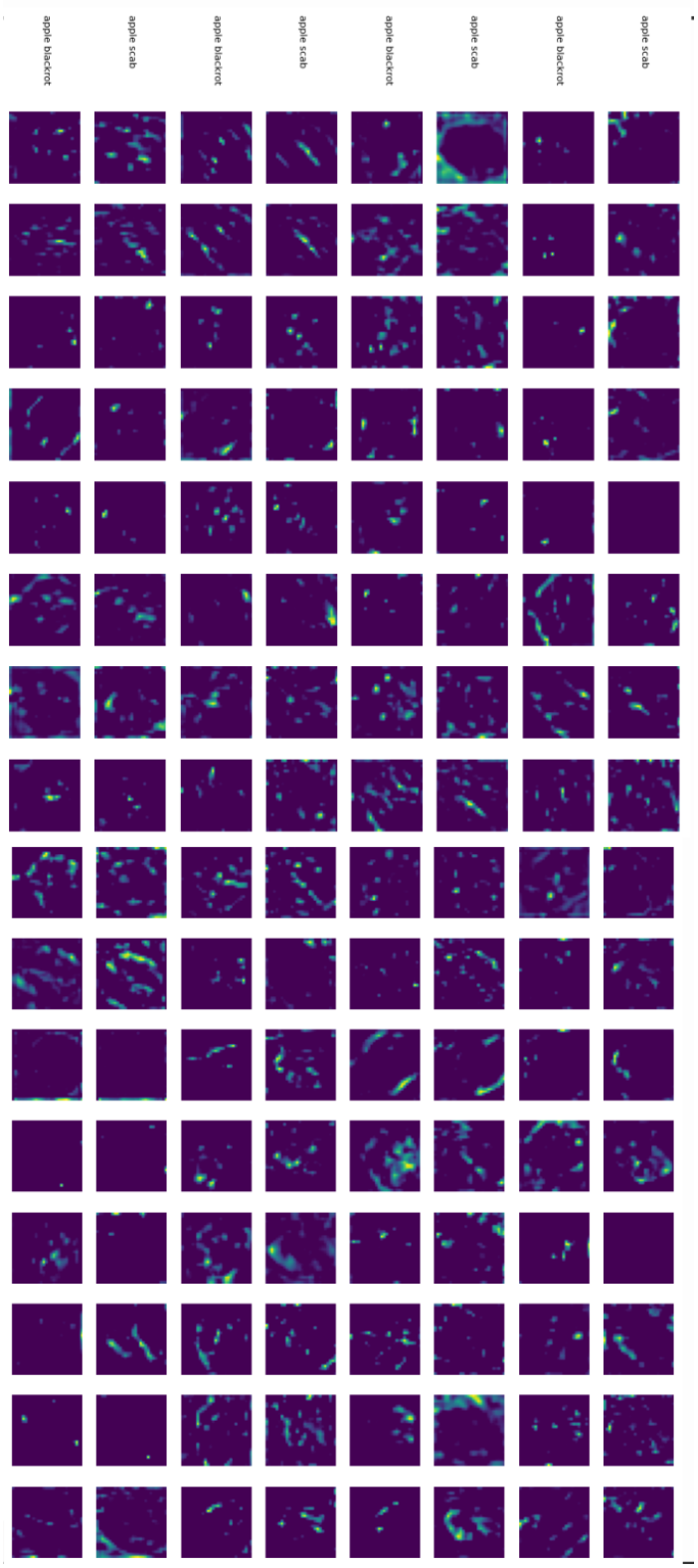

Figure 7.14: VGG16 base model Layer-13, first 64 Feature Map Comparison with Apple Scab and Apple Black Rot image

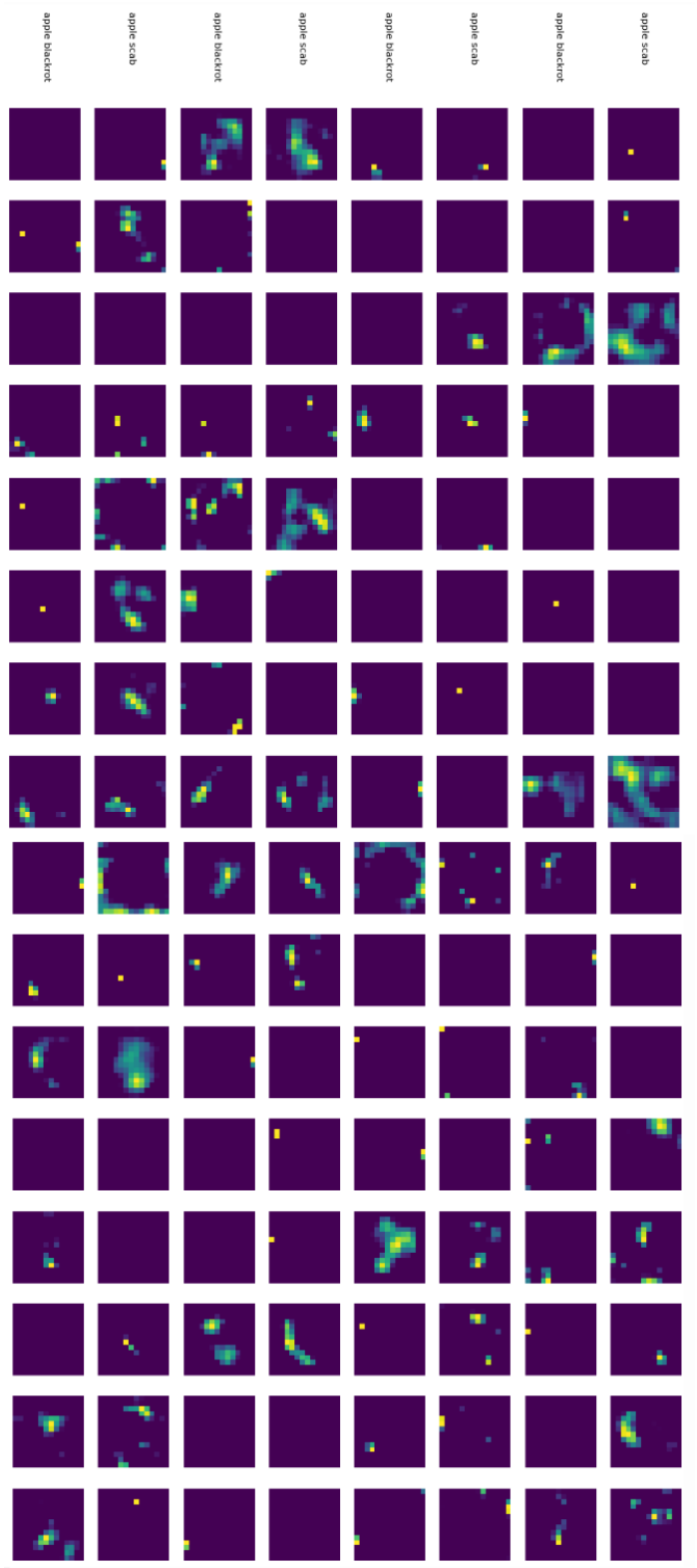

Figure 7.15: VGG16 base model Layer-17, first 64 Feature Map Comparison with Apple Scab and Apple Black Rot image

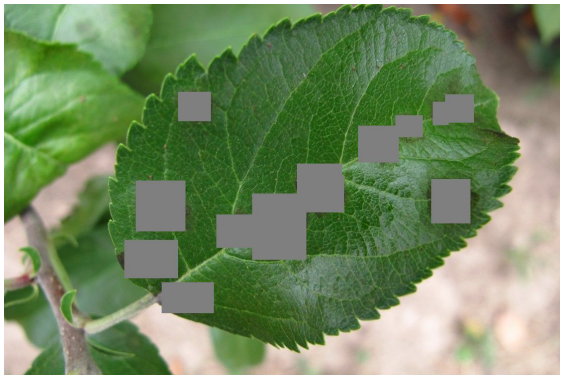

Figure 7.16: Apple Scabs with Masks

7.5 Using PCA to vizualise Feature Space

Principal component analysis (PCA) is a linear dimensionality reduction technique with applications in exploratory data analysis, visualization and data preprocessing.

The data is linearly transformed onto a new coordinate system such that the directions (principal components) capturing the largest variation in the data can be easily identified [36] [37].

7.5.1 Working of Principal Component Analysis (PCA)

1. **Standardize the Data**
   If the features of your dataset are on different scales, it's essential to standardize them (subtract the mean and divide by the standard deviation).

2. **Compute the Covariance Matrix**
   Calculate the covariance matrix for the standardized dataset.

3. **Compute Eigenvectors and Eigenvalues**
   Find the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors represent the directions of maximum variance, and the corresponding eigenvalues indicate the magnitude of variance along those directions.

4. **Sort Eigenvectors by Eigenvalues**
   Sort the eigenvectors based on their corresponding eigenvalues in descending order.

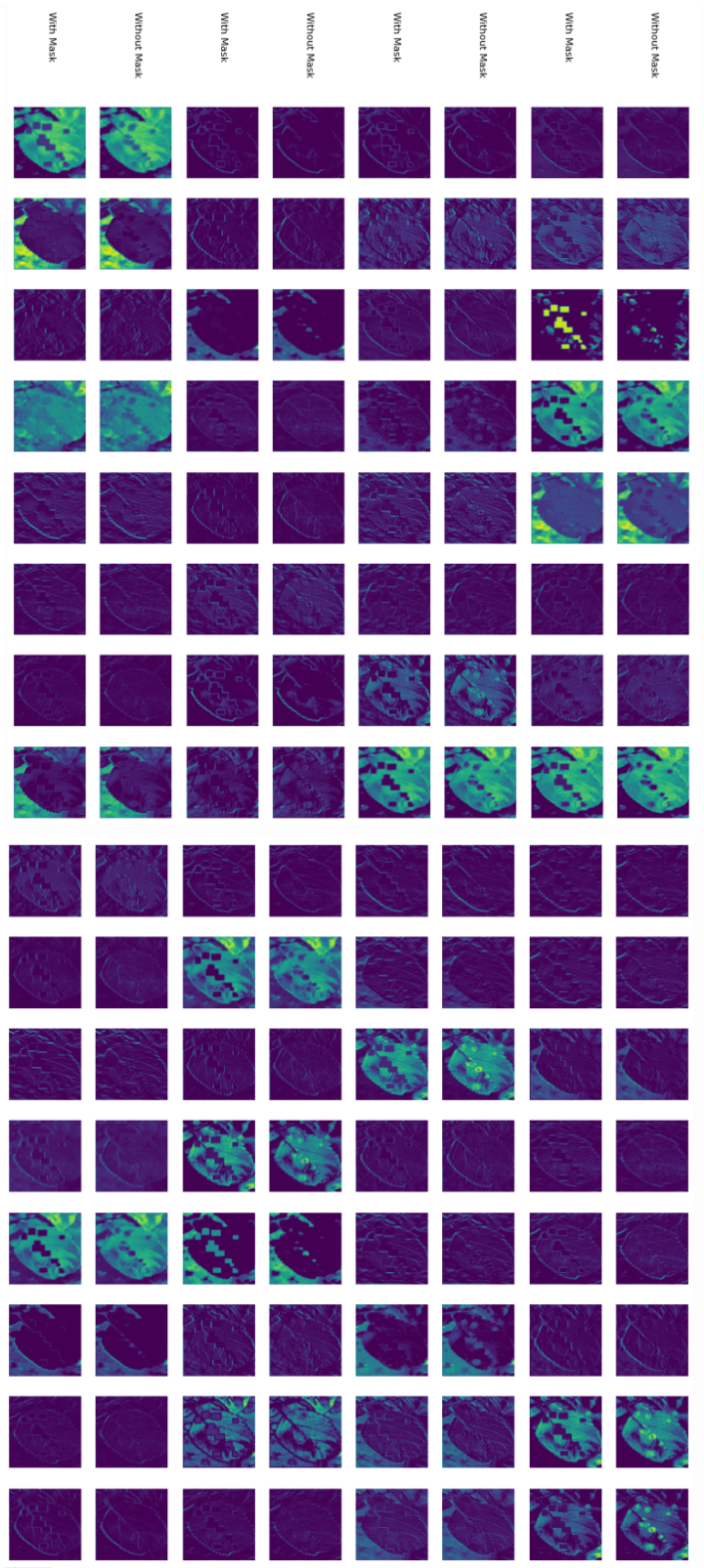

Figure 7.17: VGG16 base model Layer-2, first 64 Feature Map Comparison with Apple Scabs and Mask Apple Scabs

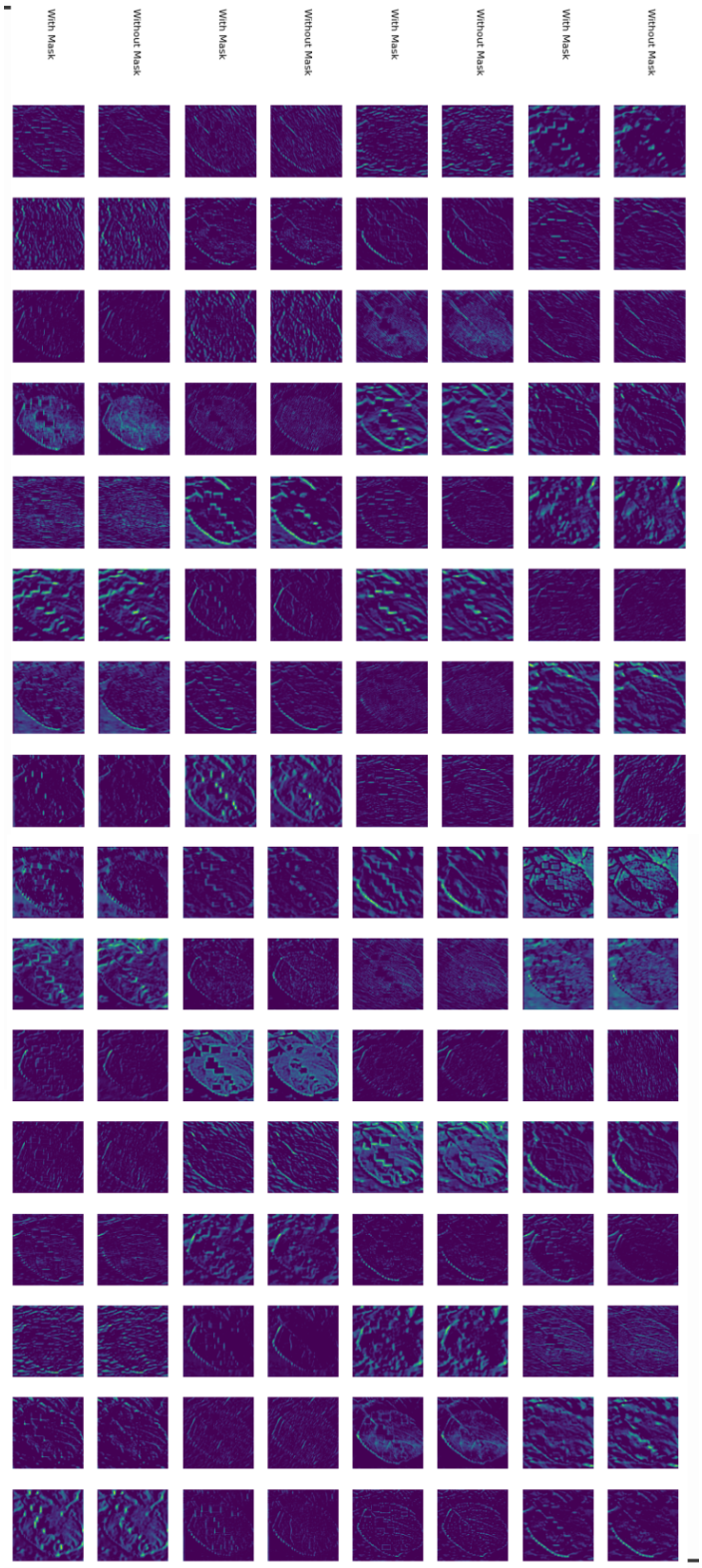

Figure 7.18: VGG16 base model Layer-5, first 64 Feature Map Comparison with Apple Scabs and Mask Apple Scabs

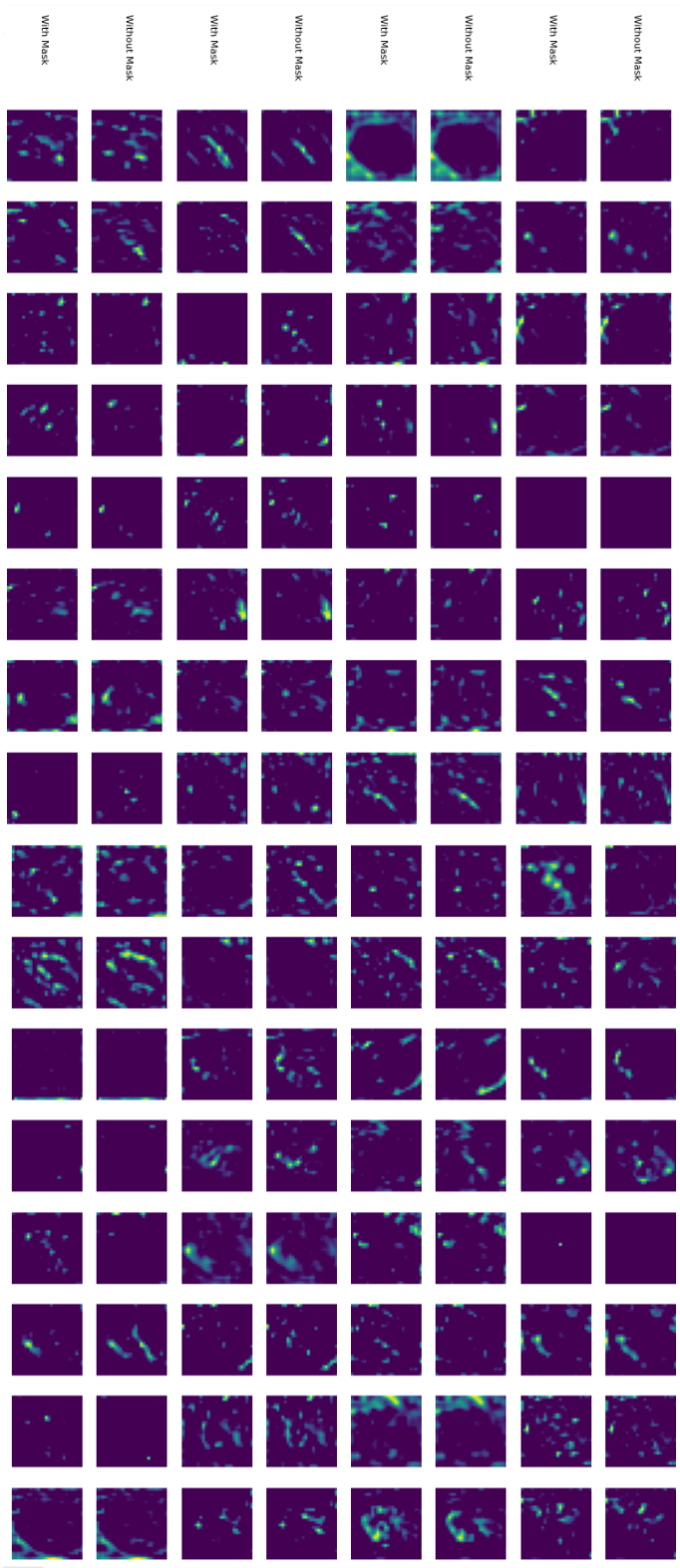

Figure 7.19: VGG16 base model Layer-13, first 64 Feature Map Comparison with Apple Scabs and Mask Apple Scabs

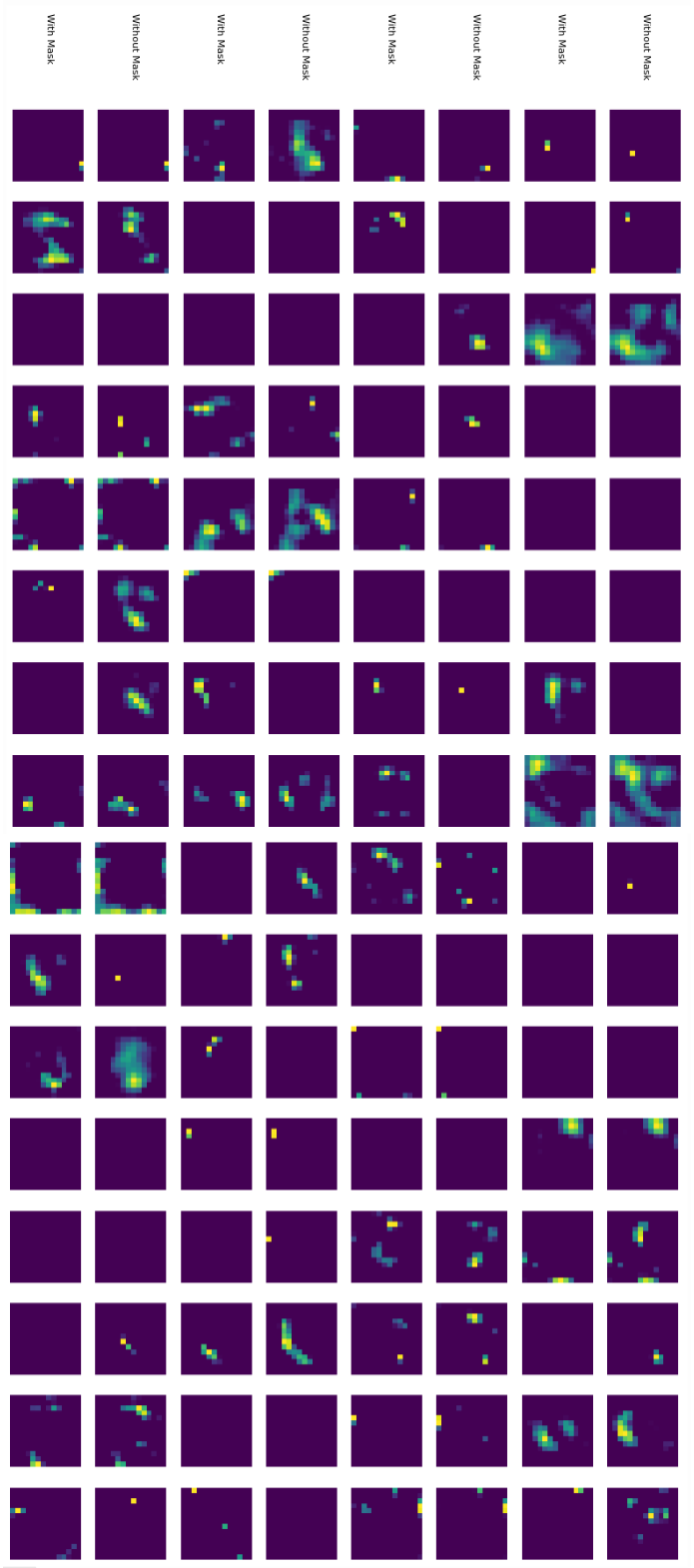

Figure 7.20: VGG16 base model Layer-17, first 64 Feature Map Comparison with Apple Scabs and Mask Apple Scabs

5. **Choose Principal Components**
   Select the top k eigenvectors (principal components) where k is the desired dimensionality of the reduced dataset.

6. **Transform the Data**
   Multiply the original standardized data by the selected principal components to obtain the new, lower-dimensional representation of the data. [37]

7.5.2 PCA Feature Space Visualisation

To analyze the feature representation learned by our models, we visualized the feature space using Principal Component Analysis (PCA). This analysis helps in understanding how well the model distinguishes between different classes based on the extracted features.

7.5.3 Before Fine Tuning

Observations: refer fig7.22

- When applying PCA to the features extracted by the pre-trained VGG16 model, the visualization did not show distinct clusters. Images from different classes were intermixed, indicating that the features were not sufficiently discriminative for different plant diseases.

- Leaves from the same species, regardless of disease, were often grouped together. This suggests that the pre-trained model's features were more influenced by species characteristics rather than disease-specific traits.

7.5.4 After Fine Tuning

Observations: 7.24

- After fine-tuning the VGG16 model on the Plant Village dataset, the PCA visualization showed somewhat clearer clusters. There was a noticeable improvement in the separation of different disease classes, although the clusters were still not very distinct.

Figure 7.21: Legend used in feature space vizualization

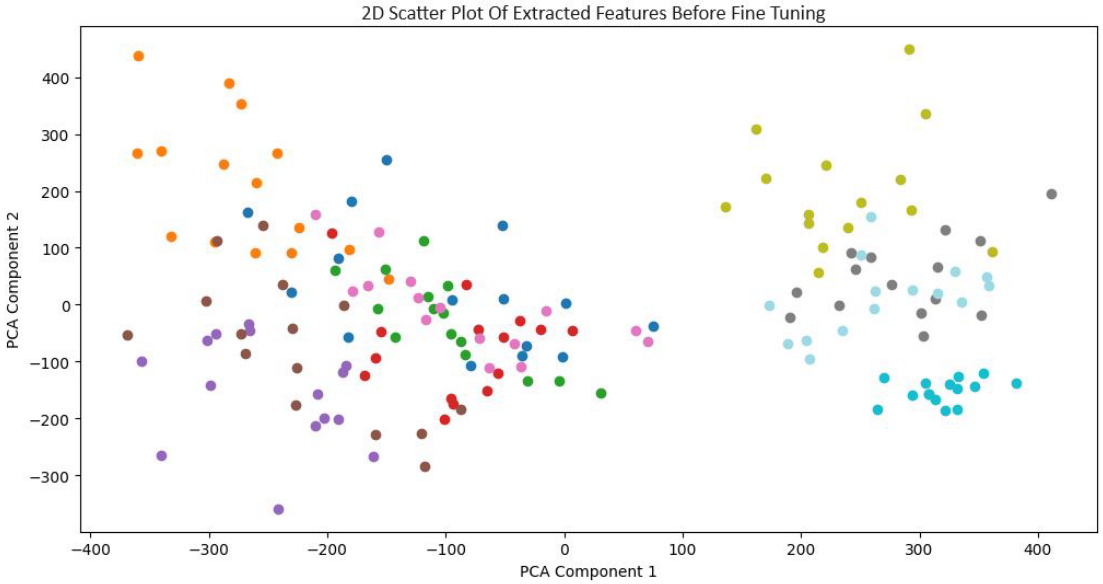

Figure 7.22: VGG16 Feature Space Visualisation Before Fine Tuning Using PCA in 2D

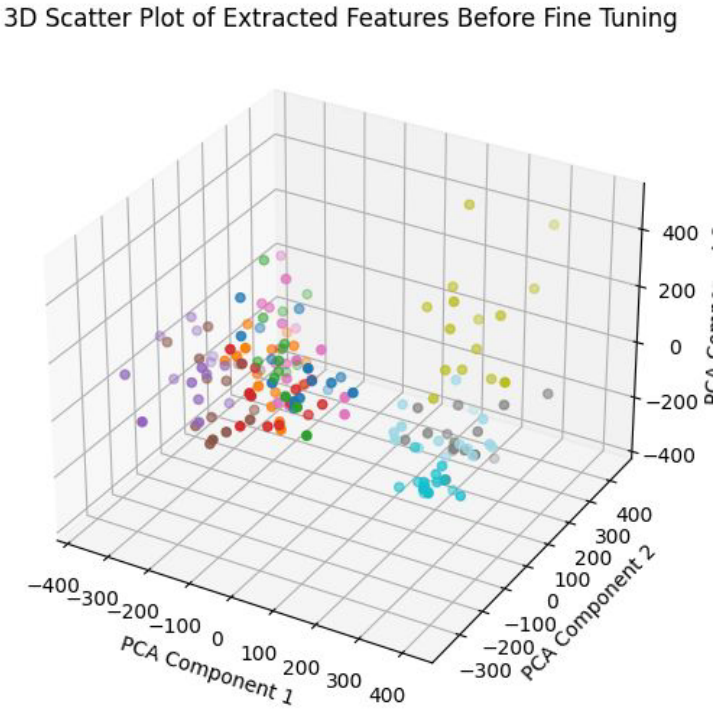

Figure 7.23: VGG16 Feature Space Visualisation Before Fine Tuning Using PCA in 3D

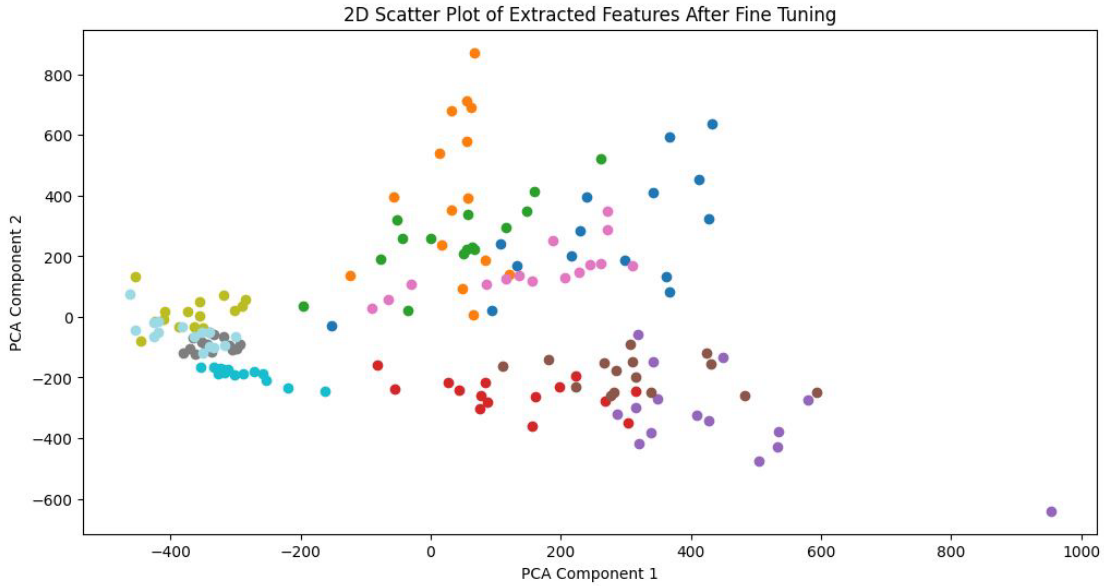

Figure 7.24: VGG16 Feature Space Visualisation After Fine Tuning using PCA in 2D

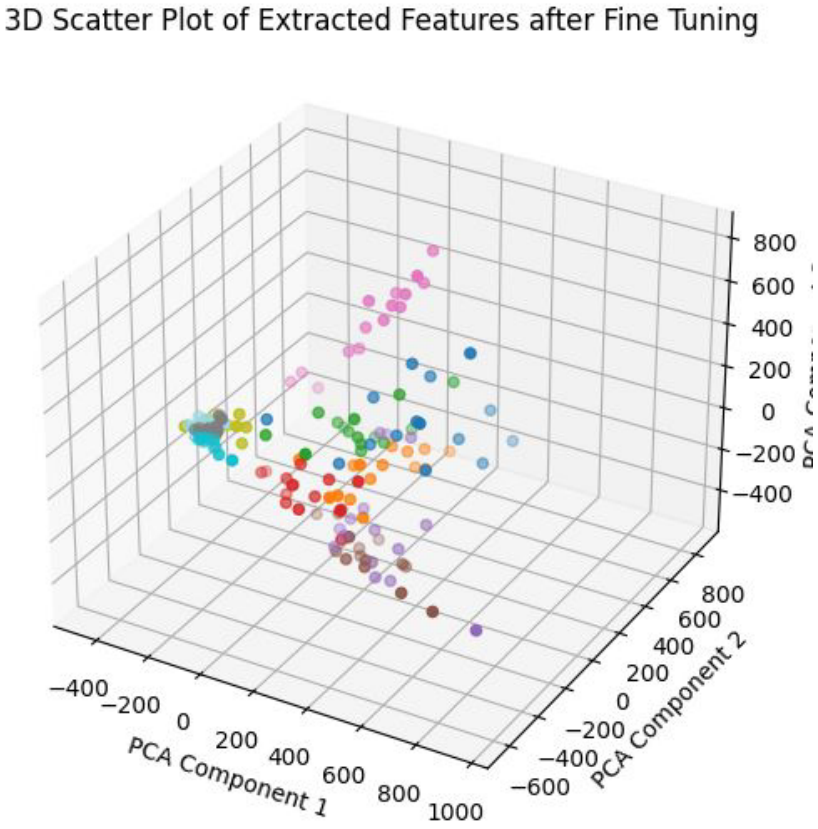

Figure 7.25: Feature Space Visualisation After Fine Tuning using PCA on 3D

7.6 Using t-SNE to vizualise Feature Space

t-distributed stochastic neighbor embedding (t-SNE) is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. It is a nonlinear dimensionality reduction technique for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions [38] [39] [40].

Since, feature space visualisation using PCA, doesn't provide work as expected by our team, we perform the same visualisation technique using t-SNE and get our expected results in the feature space visualisation both in 2D and 3D features space.

7.6.1 Before Fine Tuning

Observations: refer fig7.26

- Using t-SNE on the same features also resulted in poorly defined clusters. The intermixing of different disease classes persisted, further confirming that the pre-trained model did not adequately separate the disease classes in the feature space.

7.6.2 After Fine Tuning

Observations: fig7.28

- Applying t-SNE to the features from the fine-tuned VGG16 model revealed prominent clusters for each class. The visualization showed that the fine-tuned model could effectively separate different disease classes, with each class forming its own local cluster.
- Unlike the pre-tuned model, the fine-tuned model's feature space demonstrated that leaves were clustered based on disease classes rather than species. Leaves of the same species but different diseases were now separated, indicating that the fine-tuned model learned disease-specific features.

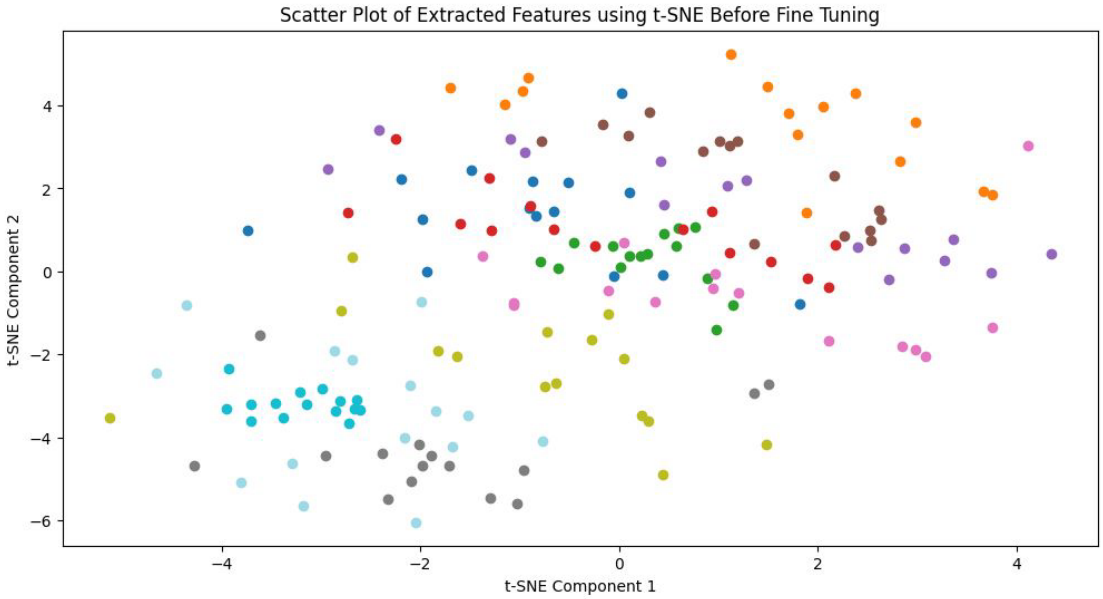

Figure 7.26: VGG16 Feature Space Visualisation Before Fine Tuning using t-SNE in 2D

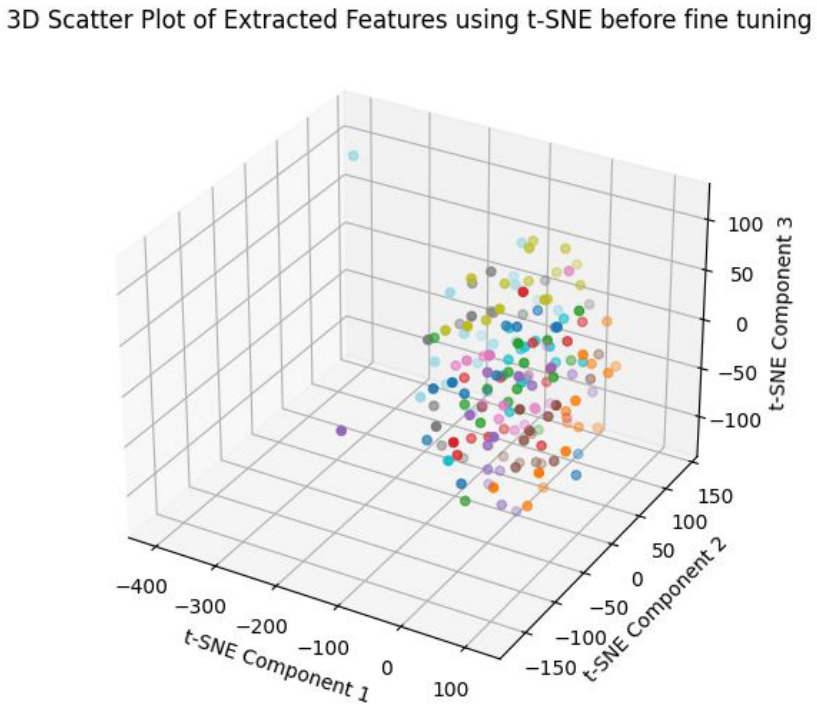

Figure 7.27: VGG16 Feature Space Visualisation Before Fine Tuning using t-SNE in 3D

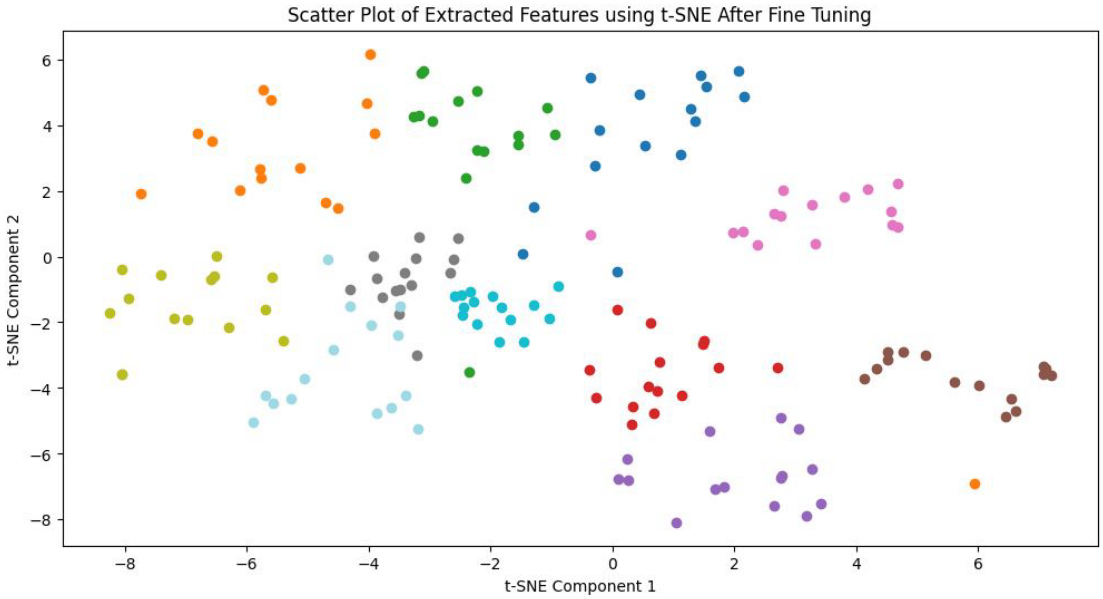

Figure 7.28: VGG16 Feature Space Visualisation After Fine Tuning using t-SNE in 2D

The feature space visualization using PCA and t-SNE highlights the significant improvements achieved through fine-tuning. Before fine-tuning, the pre-trained model's features were not adequately distinguishing between different diseases, often clustering images based on species. After fine-tuning, the model learned more discriminative features, leading to clearer separation of disease classes in the feature space, by forming local cluster and also preserving the information that these images are from the same species as the local cluster of each plant disease from the same plant disease are located closely in the feature space.

For example, this fig 7.24, we observe that the fine tuned VGG16 base model group *Corn* plant species together globally in the feature space and also we observe that individual corn disease are also clustered together locally.

This enhanced representation underscores the effectiveness of fine-tuning in adapting pre-trained models to specific classification tasks, resulting in more accurate and reliable predictions.

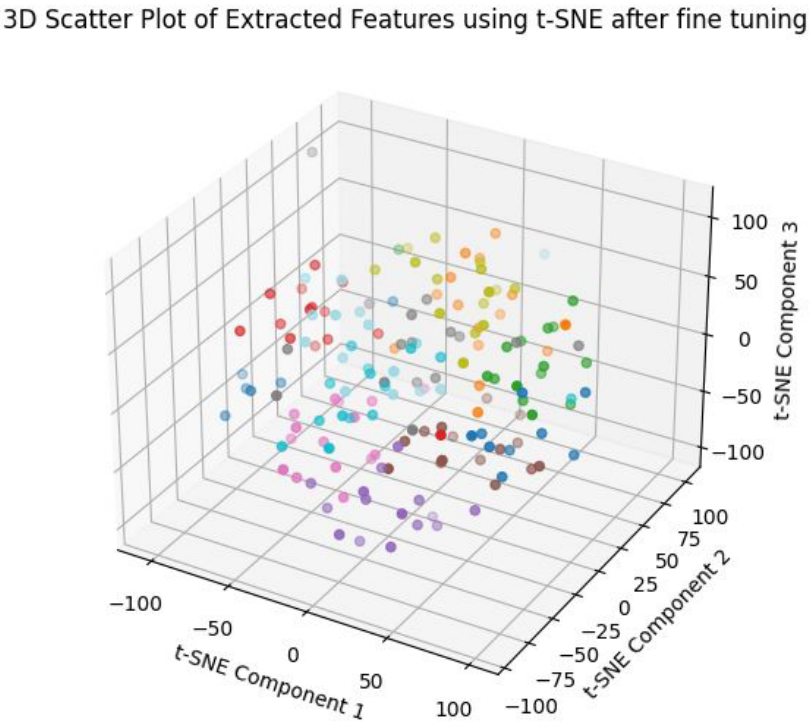

Figure 7.29: VGG16 Feature Space Visualisation After Fine Tuning using t-SNE in 3D

7.7 Visualising Nearest Neighbors in Features Space

In this technique, we used Cosine similarity,

$$cosineSimilarity(A, B) = cos(\theta) = \frac{A * B}{|A| * |B|}$$

to find and visualize the nearest neighbors of a query image in the feature space. The features from our image is extracted using our fine tuned VGG16 base model.

From analysis of nearest neighbor using cosine similarity, we observe that the nearest neighbors data instance in feature space exhibits similar features to the query image. These demonstrate the model's ability to group similar images effectively

The steps we take to perform in this technique are as follows:

Algorithm 1 Find Nearest Neighbors in Feature Space

Input:

- *query_image*: Image for which nearest neighbors are to be found
- *Feature Extractor*: Pretrained model to extract features
- *Image Dataset*: Dataset containing images
- N: Number of nearest neighbors to find

Output:

- *Nearest neighbors*: List of N nearest images to the query image in feature space

1: Extract feature from the *query image* using the *feature extractor* model
2: Similarly, extract features from all image in image dataset
3: Compute cosine similarity with query image features and all other image features
4: From the cosine similarity result find the $\boldsymbol{N}$ largest value and plot the corresponding image on graph

7.7.1 Before Fine Tuning

In this phase we, observe the following:

- When querying the pre-trained VGG16 model with an image of apple black rot, the nearest neighbors included images of healthy apple leaves, apple scab, and other apple diseases.
- This indicates that the feature representations learned from the ImageNet dataset were not sufficiently specialized to distinguish between different types of apple diseases. The model grouped various apple-related images closely, despite differences in disease characteristics.

7.7.2 After Fine Tuning

After Fine Tuning our model, we, observe that:

- After fine-tuning the VGG16 model on the Plant Village dataset, the nearest neighbor results showed a significant improvement.
- When the same query image of apple black rot was used, the nearest neighbors identified by the model were predominantly images of apple black rot, demonstrating a high degree of specificity.
- This improvement suggests that fine-tuning allowed the VGG16 model to learn more precise and relevant features specific to the Plant Village dataset, enhancing its ability to correctly differentiate between similar classes.

The nearest neighbor analysis highlights the effectiveness of fine-tuning in improving model performance. By adapting the pre-trained weights to the specific characteristics of plant disease images, the fine-tuned VGG16 model achieved a better representation of disease-specific features, leading to more accurate and reliable classifications.

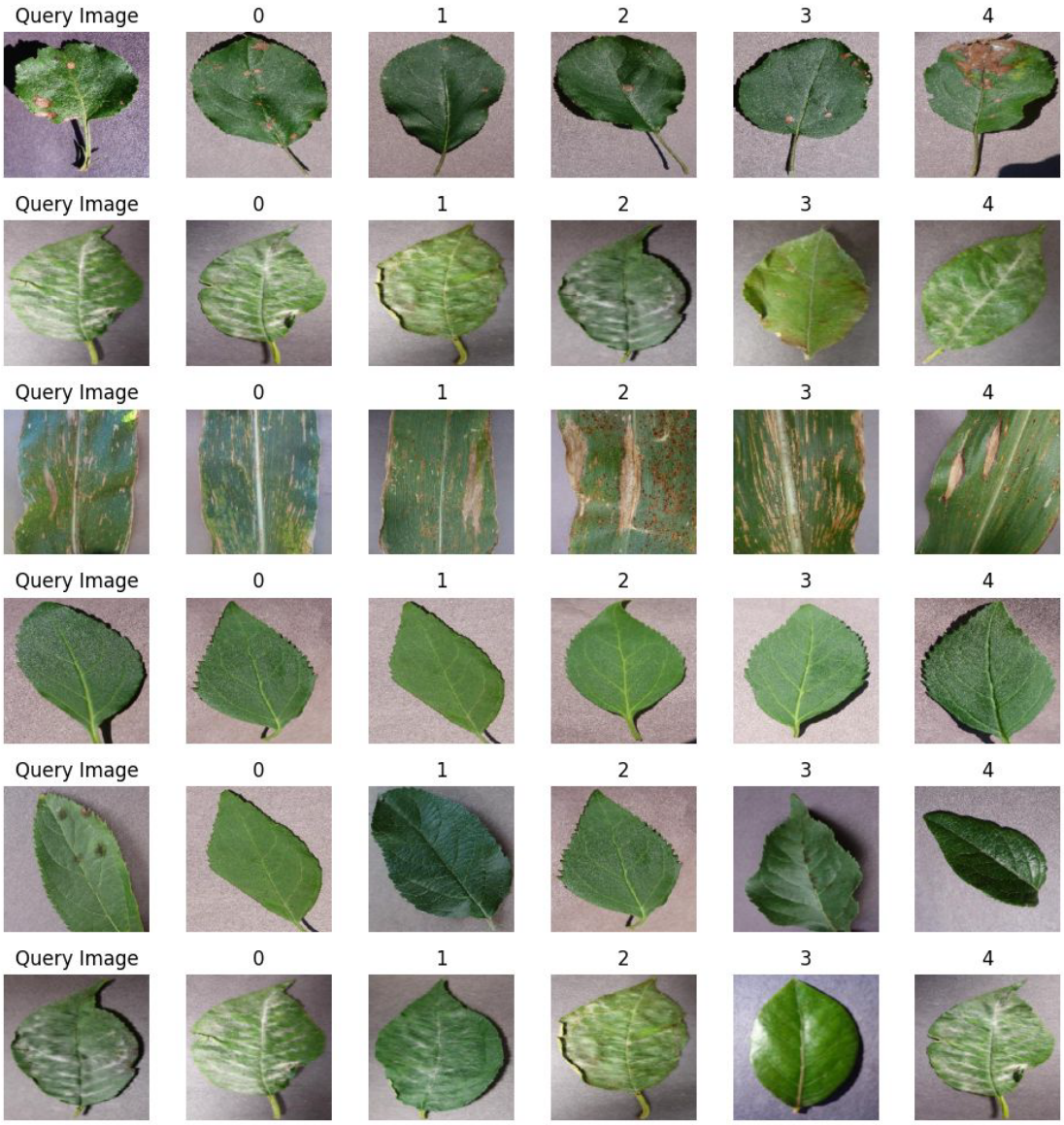

Figure 7.30: Feature Space Nearest Neighbour Before Fine Tuning, where image labeled 0 in each row is the closest to the query image in feature space and labeled 1 is the second closest and so on.

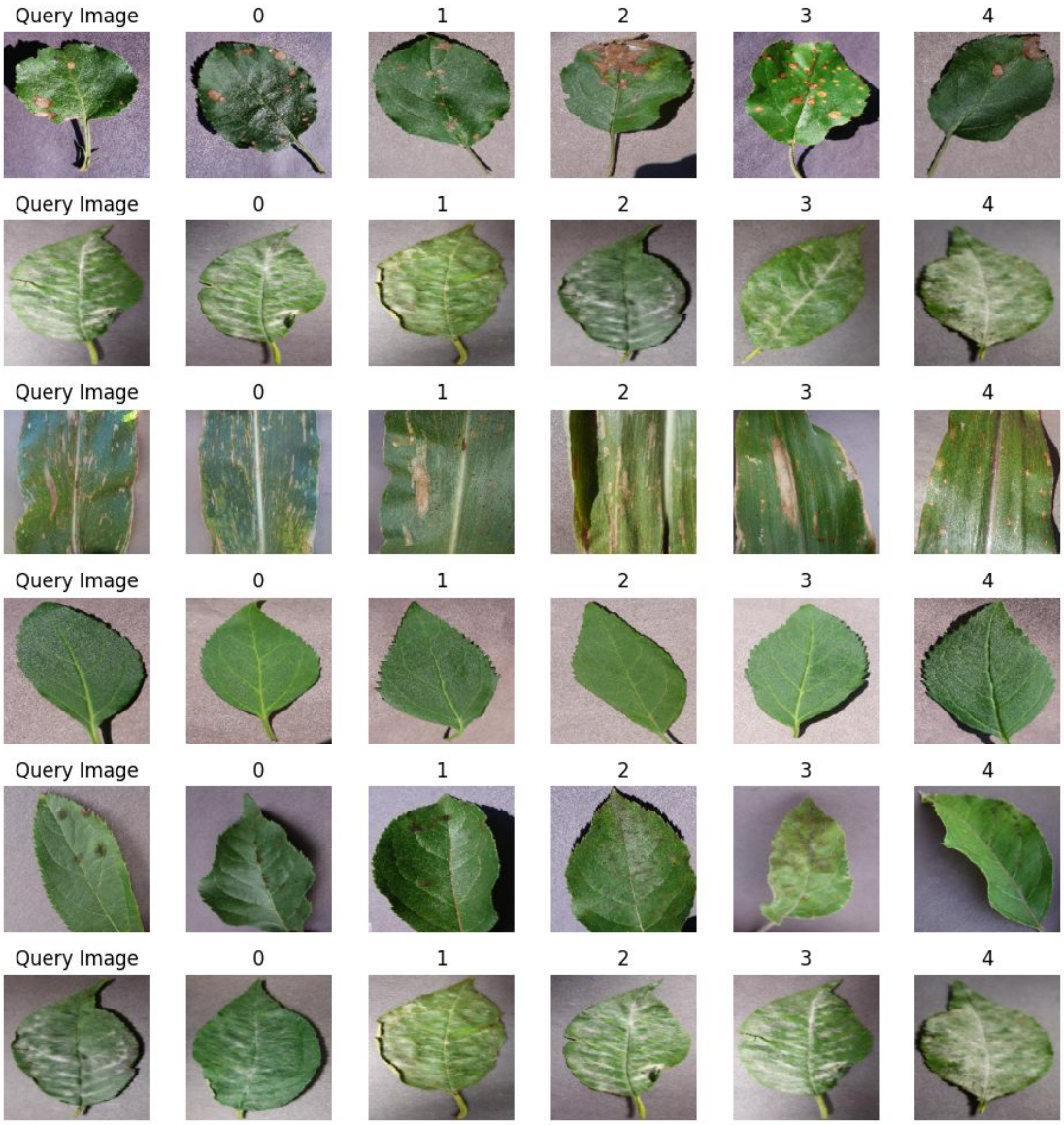

Figure 7.31: Feature Space Nearest Neighbors After Fine Tuning, where image labeled 0 in each row is the closest to the query image in feature space and labeled 1 is the second closest and so on.

7.8 Saliency Maps

In computer vision, a saliency map is an image that highlights either the region on which people's eyes focus first or the most relevant regions for machine learning models. The goal of a saliency map is to reflect the degree of importance of a pixel to the human visual system or an otherwise opaque ML model [41] [42].

> A saliency map highlights the important parts of an image that a machine learning model focuses on when making a prediction. It shows which pixels in the image have the most influence on the model's decision.

7.8.1 Explainable Artificial Intelligence using Saliency Map

Explainable Artificial Intelligence in the context of black box machine learning models: Saliency maps are a prominent tool providing visual explanations of the decision-making process of machine learning models, particularly deep neural networks. These maps highlight the regions in input images, text, or other types of data that are most influential in the model's output, effectively indicating where the model is "looking" when making a prediction. By illustrating which parts of the input are deemed important, saliency maps help in understanding the internal workings of otherwise black box models, thereby fostering trust and transparency. In image classification tasks, for example, saliency maps can identify pixels or regions that contribute most to a specific class decision. One of the most prominent techniques to generate saliency maps is Gradient-weighted Class Activation Mapping (Grad-CAM).

7.8.2 Steps for Generating Saliency Maps

The steps are as follows:

1. Prediction:
   First, we pass an input image through the model to get its prediction. The model outputs a set of scores (or probabilities) for each possible class. For example, if the model is a dog breed classifier, it might output scores for classes like "Labrador", "Poodle", "Bulldog", etc.

2. Identify the Top Class:
   We identify the class with the highest score. This is the class that the model thinks the image most likely belongs to. Let's call this the "top class."

3. Compute the Gradient:
   To understand which parts of the image contributed most to the top class score, we compute the gradient of this score with respect to each pixel in the input image.
   The gradient tells us how much a small change in each pixel would affect the top class score. If changing a pixel significantly increases the score, that pixel is important for the model's prediction.

4. Take the Absolute Value: We take the absolute value of these gradients. This step is important because we care about the magnitude of the impact (how much it affects the score) rather than the direction (whether it increases or decreases the score).

5. Aggregate Across Color Channels:
   If the image has multiple color channels (like RGB), we combine the gradients across these channels. A common approach is to take the maximum gradient value across the channels for each pixel. This gives us a single importance value per pixel.

6. Create the Saliency Map:
   The result is a saliency map, where each pixel's value represents its importance in the model's prediction. High values indicate pixels that had a strong influence on the top class score, while low values indicate less important pixels.

The key idea behind saliency maps is using gradients, which are a core concept in how neural networks learn. During training, gradients show how to adjust weights to reduce the error. Here, we use gradients to see how changes in the input image affect the model's output, highlighting the parts of the image that are most influential. [41] [43]

7.8.3 Observations from Saliency Map

In our observations of saliency maps generated by both a pre-trained model and a fine-tuned model, several notable differences emerges. The pretrained model is effectively good in highlighting regions of images which are of general interest in classification tasks, fig: 7.32, but in specialized task it produces broader and less precise saliency regions. This is likely due to its training on a generalized training datasets, which help them to equip with a wide-ranging but somewhat superficial understanding of features.

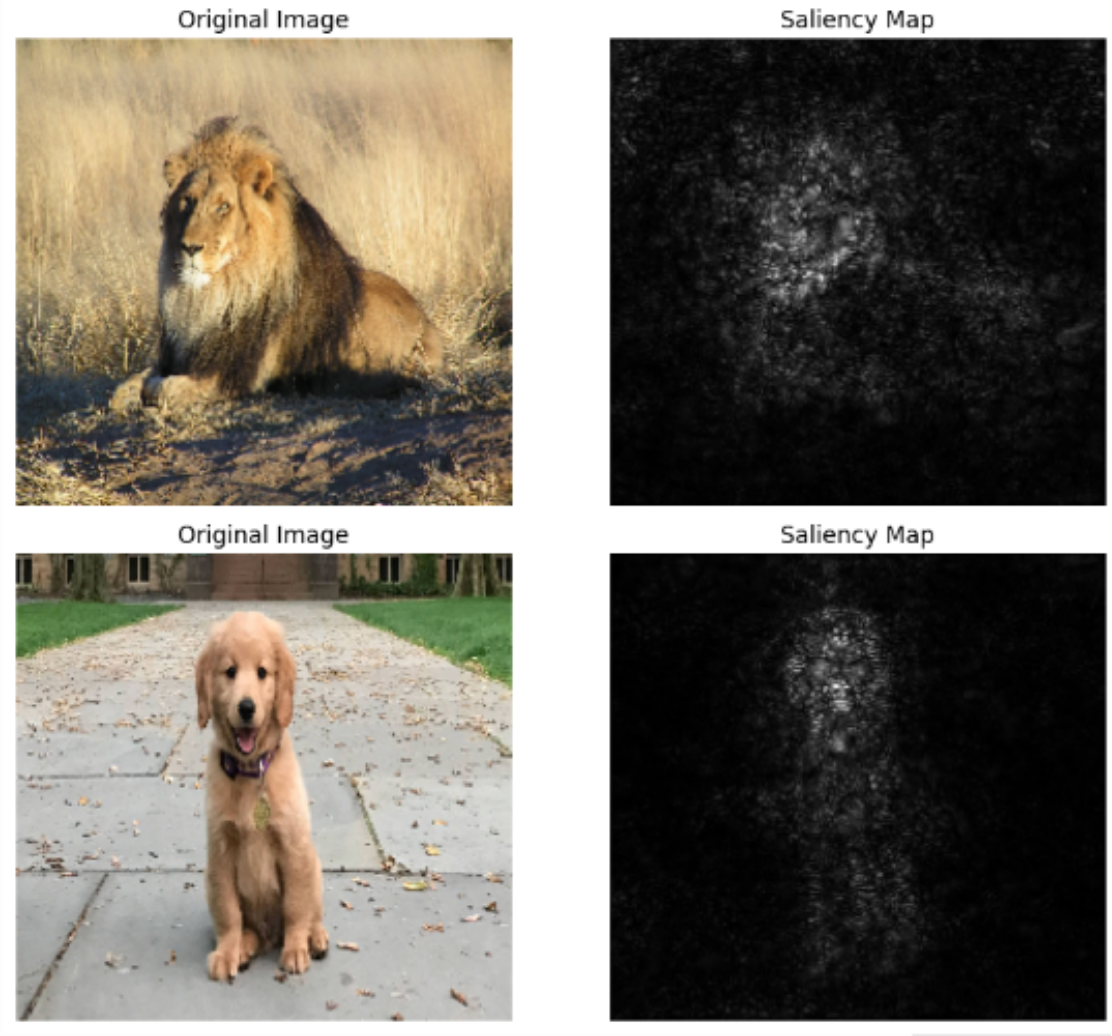

Figure 7.32: Saliency Map Generated by VGG16 trained on ImageNet

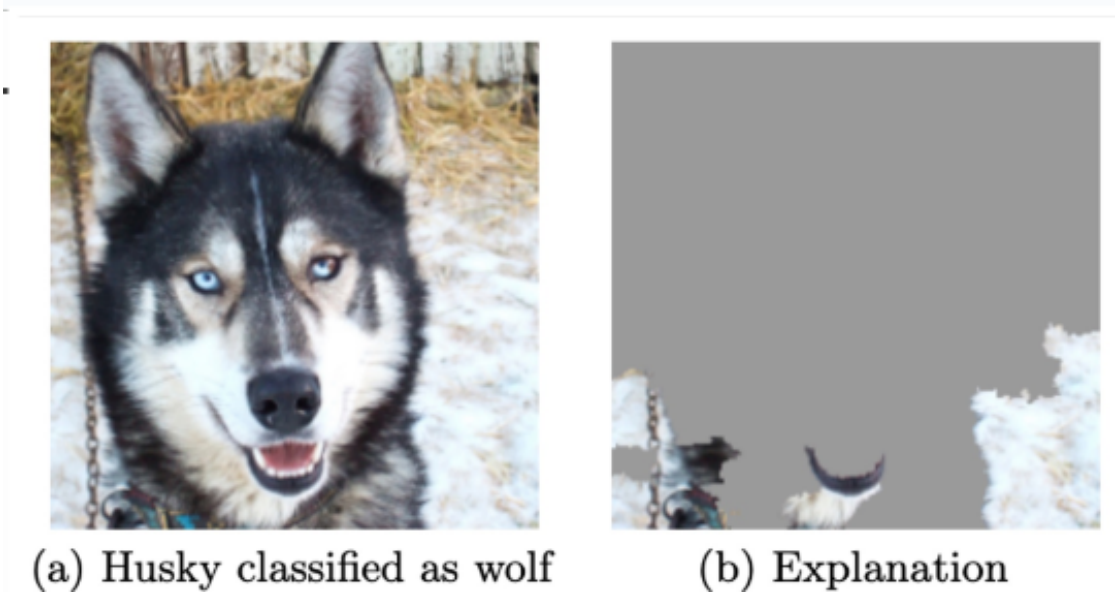

Figure 7.33: Saliency Map Generated by a Dog Classifier Model

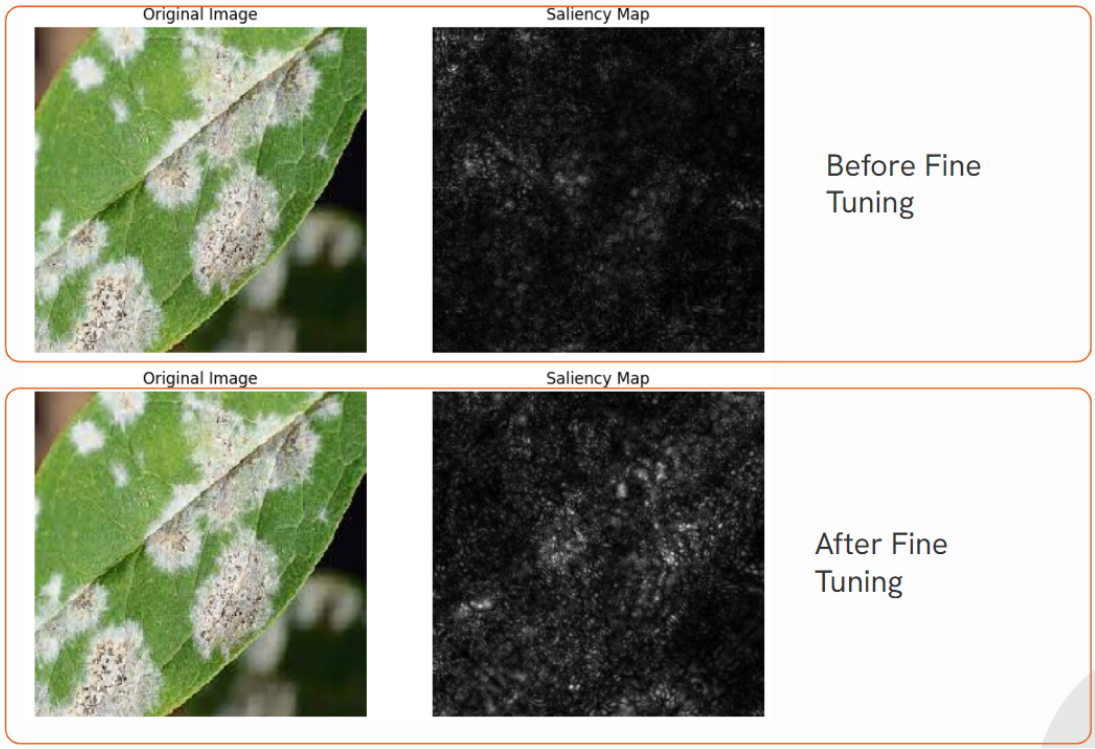

Figure 7.34: Saliency Map Generated by VGG16 base model before and after Fine Tuning on Plant Village Datasets

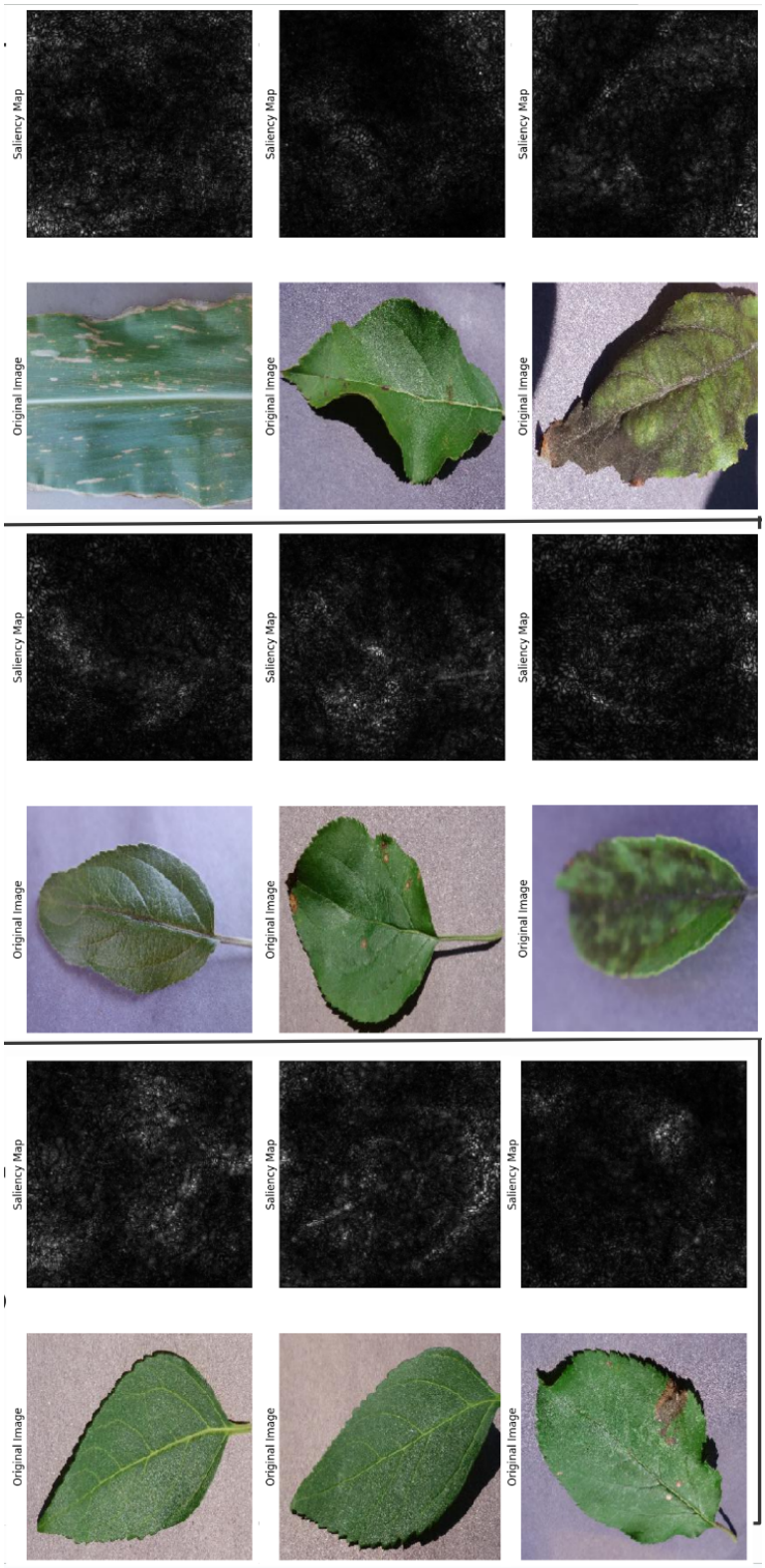

Figure 7.35: Saliency Map generated by VGG16 base model before fine tuning

In contrast, the fine-tuned model demonstrates a marked improvement in the specificity and relevancy of the saliency map. Fine tuning the model on plant village datasets allows it to adapt and improve its feature detection capabilities, resulting in more accurate and focused identification of critical regions of the image.

For example: in fig:7.34 we observe that before fine-tuning our models does not have a clear ideas on which areas to focus, but after fine tuning, we observe that the model tries to focus on the regions of leaves which are affected (white spot regions).

In fig: 7.35 and fig:7.36, we provide another visualization of saliency map produced by our model, before and after fine tuning on 9 data instances. Some of the observation that are notable ones is 3-row, 2-col image, we observe that the model focus on the whole regions of images, even the background, but after fine tuning, it focus only on the infected region of the leaves.

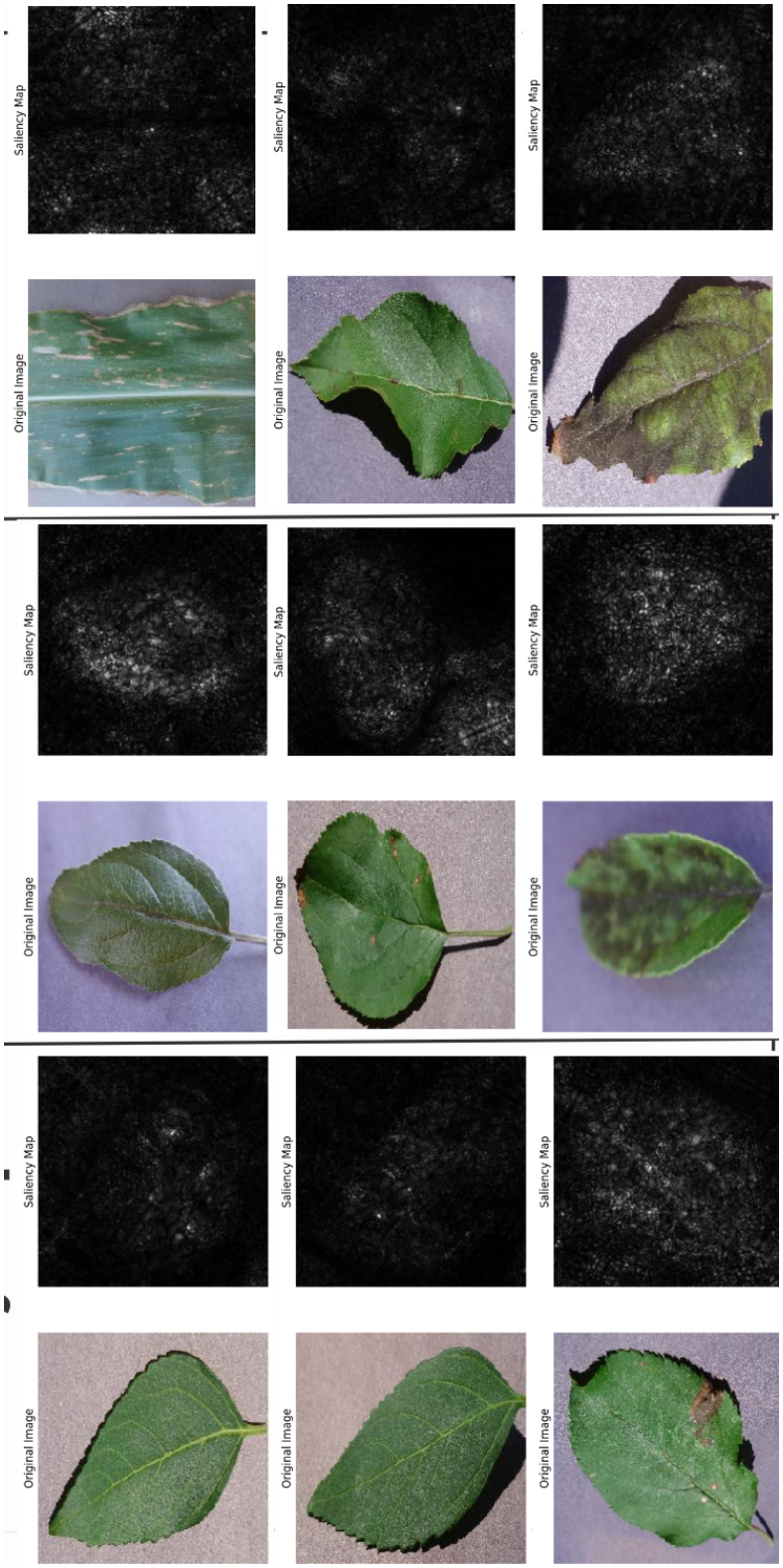

Figure 7.36: Saliency Map generated by VGG16 base model after fine tuning last

Chapter 8

Conclusion

8.1 Summary of Findings

In this project we have conducted various comprehensive studies of various image classification models,using the PlantVillage dataset to detect plant diseases. Our approach included utilizing several pre-trained models, namely MobileNetV2, VGG16, Xception for transfer learning and fine tuning, and also training traditional classifiers like SVM, KNN, Random Forest, and XGBoost on extracted features.

The results obtained from this study demonstrated that deep learning models, particularly convNet models, outperformed traditional machine learning classifiers in accuracy and other classifier evaluation metrics such as Precision, Recall, F-Score.

To further study the decisions made by the models, we also employed several model interpretation techniques. We visualized the first layer filters and feature maps of our best performing models, analyzed the model's response to masked images, and used dimensionality reduction techniques like PCA and t-SNE to plot feature spaces before and after fine-tuning. Additionally, we also implemented nearest neighbour search in feature space using cosine similarity and lastly, we also generate saliency maps to highlight important image regions influencing predictions. From these, we observe that our model generalised well on the Plant Village Datasets.

8.2 Future Work

For future work, we are planning to train our model on full 38 classes in Plant Village Dataset, instead of the reduced 11 ones (the current classes that is used in this work). And check whether the performance of CNN10L increases by training it on **ImageNet** and fine-tune with Plant Village.

We are also planning to include newer transformer based models [44] such as *Vision Transformer (ViT)* model [45] in our comparison study.

Furthermore, we are also planning to add more features to our Plant Disease Detection App. Some of the features that we are planning are: integrating alert system for plant disease based on local outbreak detected by our app, provide solution and remedy to take based on the detected disease and also to add community and expert support.

Bibliography

[1] A. Naebi and Z. Feng, "The performance of a lip-sync imagery model, new combinations of signals, a supplemental bond graph classifier, and deep formula detection as an extraction and root classifier for electroencephalograms and brain–computer interfaces," *Applied Sciences*, 2023.

[2] P. VARSHNEY, "Kaggle." `https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide`, 2020. Accessed on 2024-06-24.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arxiv*, 2017.

[4] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arxiv*, 2016.

[5] L. G. Serrano, *Grokking Machine Learning*. Manning, 2021.

[6] "How the sagemaker xgboost algorithm works." `https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html`. Accessed on 2024-05-07.

[7] "Xgboost." `https://www.geeksforgeeks.org/xgboost/`, 2023. Accessed on 2024-05-07.

[8] H. Gupta, "A gentle introduction to cross validation." `https://dockship.io/articles/6045b9c1c2c7fa59d94ed69a/a-gentle-introduction-to-cross-validation`, 2021. Accessed on 2024-05-07.

[9] A. Suresh, "What is a confusion matrix." `https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5`, 2020. Accessed on 2024-05-07.

[10] V. Ragav and L. Baoxin, *Convolutional Neural Networks in Visual Computing: A Concise Guide*. CRC Press, 2017.

[11] L. Stankovic and D. Mandic, "Convolutional neural networks demystified: A matched filtering perspective based tutorial," *arxiv*, 2022.

[12] "Convolutional neural networks (lenet)." `http://deeplearning.net/tutorial/lenet.html`, 2010. Accessed on 2024-06-24.

[13] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," *Twenty-Second International Joint Conference on Artificial Intelligence-Volume Volume Two. 2: 1237–1242*, 2013.

[14] J. Brownlee, "A gentle introduction to the rectified linear unit (relu)." machine learning mastery-relu, 2019. Accessed on 8 April 2021.

[15] J. West, D. Ventura, and S. Warnick, *Spring Research Presentation: A Theoretical Foundation for Inductive Transfer*. PhD thesis, Brigham Young University, College of Physical and Mathematical Sciences, 2007.

[16] J. Quinn, *Dive into deep learning: tools for engagement*. PhD thesis, Thousand Oaks, California, 2020.

[17] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," *Advances in Neural Information Processing Systems. Vol. 35. Curran Associates*, 2022.

[18] D. Zeiler Matthew and R. Fergus, "Visualizing and understanding convolutional networks," *ECCV*, 2013.

[19] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. Smith, "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping," 2020.

[20] Y. Yu, S. Zuo, H. Jiang, W. Ren, T. Zhao, and C. Zhang, "Very deep convolutional networks for large-scale image recognition," *Association for Computational Linguistics*, 2020.

[21] K. Simonyan and A. Zisserman, "Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach," *arxiv*, 2014.

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *arxiv*, 2018.

[23] C. CORTES and V. VAPNIK, "Support-vector networks," *1995 Kluwer Academic Publishers, Boston*, 1995.

[24] T. HOFMANN, B. SCHÖLKOPF, and A. J. SMOLA, "Kernel methods in machine learning," *The Annals of Statistics 2008, Vol. 36, No. 3, 1171–1220*, 2008.

[25] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, 1967.

[26] A. A. AnAj. CC BY-SA 3.0, via Wikimedia Commons.

[27] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research. Vol 11*, 1999.

[28] T. K. Ho, "Random decision forests," *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, 1995.

[29] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[30] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *arxiv*, 2016.

[31] T. Hastie, R. Tibshirani, and J. Friedman, *Elements of Statistical Learning: data mining, inference, and prediction. 2nd Edition.* Springer, 2009.

[32] S. A. (LIENS) and A. C. (MIA), "A survey of cross-validation procedures for model selection," *arxiv*, 2009.

[33] D. M. W. Powers, "Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation," *Research Gate*, 2008.

[34] D. L. Olson and D. Delen, "Advanced data mining techniques," *Springer, 1st edition (February 1, 2008), page 138, ISBN 3-540-76916-1*, 2008.

[35] D. Sarkar, "Model interpretation strategies." Explainable AI, Nov 2018. Accessed on 2024-06-24.

[36] Jolliffe, I. T, and J. Cadima, "Principal component analysis A review and recent developments," *Series A, Mathematical, physical, and engineering sciences vol. 374,2065 (2016):20150202. doi:10.1098/rsta.2015.0202*, 2016.

[37] J. Shlens, "A tutorial on principal component analysis," *arXiv*, 2017.

[38] T. T. Cai and R. Ma, "Theoretical foundations of t-sne for visualizing high-dimensional clustered data," *arxiv*, 2022.

[39] S. Arora, W. Hu, and P. K. Kothari, "An analysis of the t-sne algorithm for data visualization," *arxiv*, 2018.

[40] Z. Yang, Y. Chen, and J. Corander, "T-sne is not optimized to reveal clusters in data," *arxiv*, 2021.

[41] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arxiv*, 2013.

[42] A. Alqaraawi, M. Schuessler, P. Weiß, E. Costanza, and N. Berthouze, "Evaluating saliency map explanations for convolutional neural networks: A user study," *arxiv*, 2020.

[43] D. Sarkar, "Explainable ai: Saliency maps." `https://medium.com/@bijil.subhash/explainable-ai-saliency-maps-89098e230100`, Mar 2022. Accessed on 2024-06-24.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arxiv*, 2023.

[45] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *arxiv*, 2020.

Chapter 9

Appendices

9.1 Model Deployment

9.2 Application Prototype

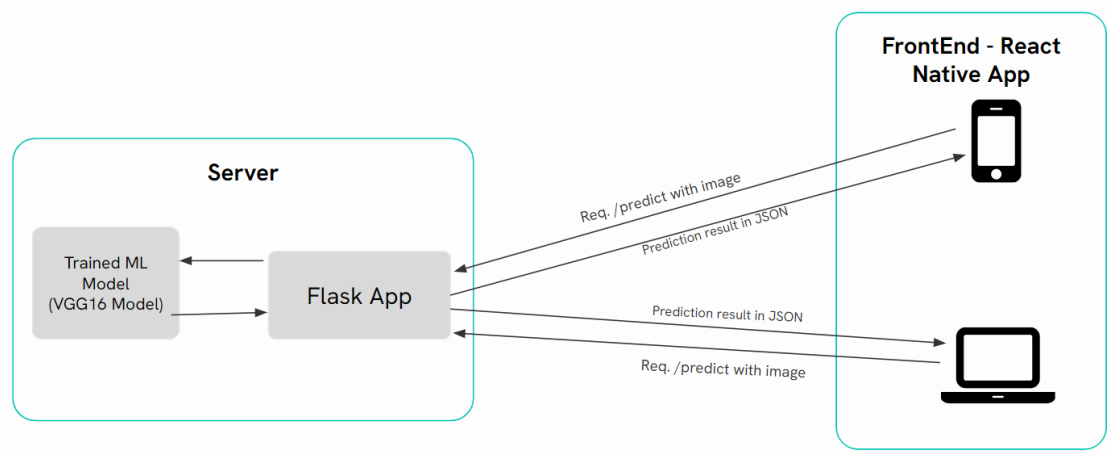

Figure 9.1: ML Model Deployment Pipeline

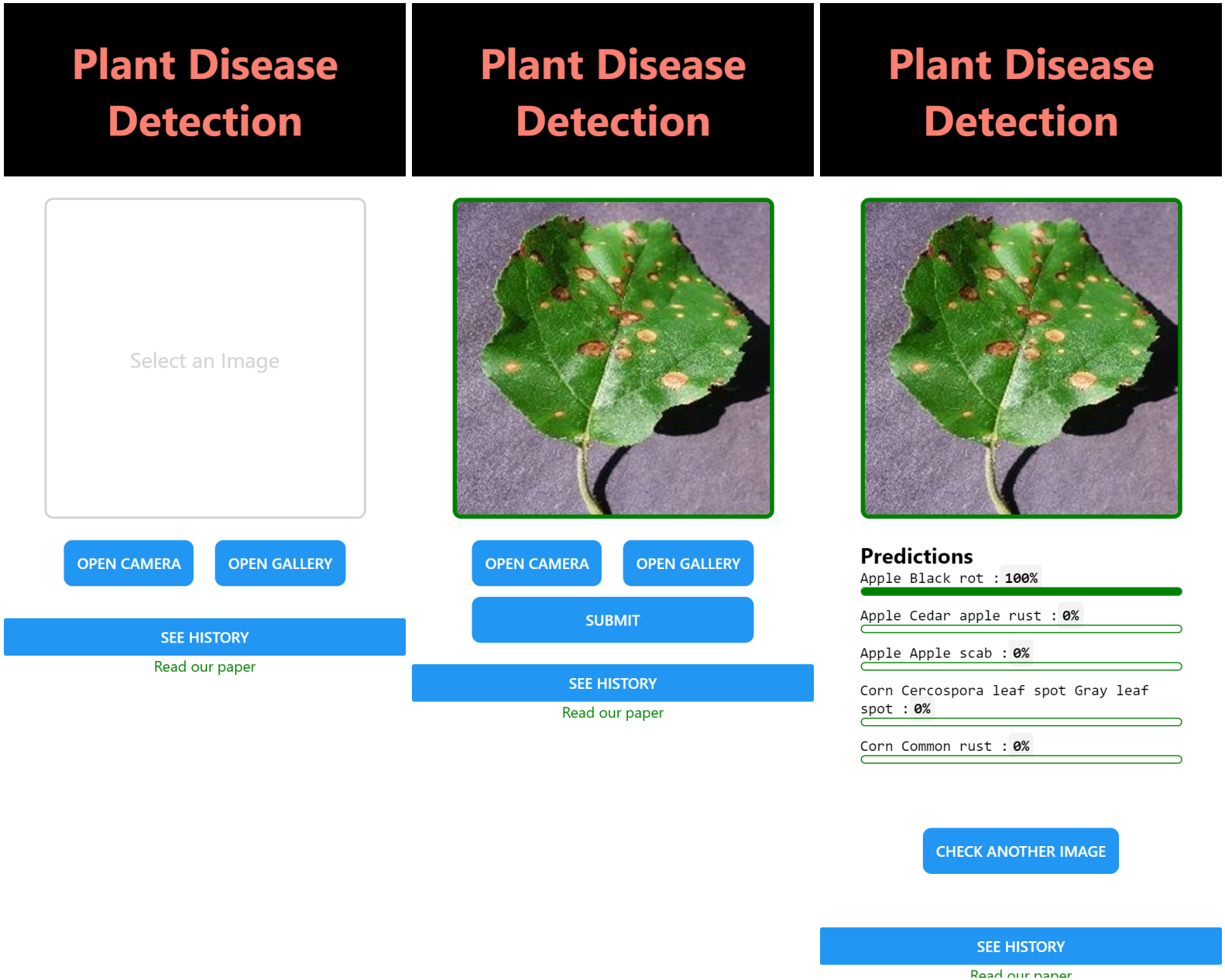

Figure 9.2: App prototype, From left to right: App Main Page, App Image Select Page and App Image Result Page

9.3 Additional Table

Class ID	Class Names
0	Apple__Apple_scab
1	Apple__Black_rot
2	Apple__Cedar_apple_rust
3	Apple__healthy
4	Blueberry__healthy
5	Cherry__healthy
6	Cherry__Powdery_mildew
7	Corn__Cercospora_leaf_spot_Gray_leaf_spot
8	Corn__Common_rust
9	Corn__healthy
10	Corn__Northern_Leaf_Blight

Table 9.1: Total Classes Number: 11

Class ID	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	63
1	1.00	1.00	1.00	62
2	0.96	1.00	0.98	26
3	1.00	1.00	1.00	164
4	1.00	1.00	1.00	150
5	1.00	1.00	1.00	85
6	1.00	1.00	1.00	105
7	1.00	0.78	0.88	65
8	1.00	1.00	1.00	119
9	1.00	1.00	1.00	116
10	0.86	0.99	0.92	85
Accuracy			0.99	1040
Macro avg	0.98	0.98	0.98	1040
Weighted avg	0.99	0.99	0.99	1040

Table 9.2: Classification report for MobileNetV2.

Class ID	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	63
1	1.00	1.00	1.00	62
2	1.00	1.00	1.00	27
3	1.00	1.00	1.00	164
4	1.00	1.00	1.00	150
5	1.00	1.00	1.00	85
6	1.00	1.00	1.00	105
7	0.71	1.00	0.83	36
8	1.00	1.00	1.00	119
9	1.00	1.00	1.00	116
10	1.00	0.87	0.93	113
Accuracy			0.99	1040
Macro avg	0.97	0.99	0.98	1040
Weighted avg	0.99	0.99	0.99	1040

Table 9.3: Classification report for VGG 16.

Class ID	Precision	Recall	F1-score	Support	
0	0.98	0.98	0.98	63	
1	1.00	1.00	1.00	62	
2	1.00	0.93	0.96	29	
3	1.00	0.97	0.98	169	
4	0.98	1.00	0.99	147	
5	0.99	1.00	0.99	84	
6	0.97	1.00	0.99	102	
7	0.88	0.94	0.91	48	
8	0.99	1.00	1.00	118	
9	1.00	1.00	1.00	116	
10	0.97	0.93	0.95	102	
	Accuracy			0.98	1040
	Macro avg	0.98	0.98	0.98	1040
	Weighted avg	0.98	0.98	0.98	1040

Table 9.4: Classification report for Xception.

Class Name	Precision	Recall	F1-score	Support
0	0.90	0.89	0.89	81
1	0.97	0.98	0.98	62
2	1.00	0.92	0.96	25
3	0.98	0.98	0.98	166
4	0.94	0.97	0.95	137
5	1.00	0.96	0.98	111
6	0.98	0.98	0.98	82
7	0.80	0.77	0.78	56
8	1.00	1.00	1.00	114
9	0.86	0.89	0.88	94
10	1.00	0.99	1.00	116
Accuracy			0.95	1044
Macro avg	0.95	0.94	0.94	1044
Weighted avg	0.95	0.95	0.95	1044

Table 9.5: Classification report for CNN10L.

Class Name	Precision	Recall	F1-score	Support
0	0.96	0.98	0.97	99
1	0.98	0.99	0.99	147
2	0.98	0.95	0.96	100
3	0.91	0.96	0.93	100
4	0.92	0.73	0.81	15
5	0.92	0.98	0.95	212
6	0.84	0.63	0.72	100
7	0.92	0.94	0.93	190
9	0.96	0.83	0.89	95
10	0.87	0.92	0.89	177
Accuracy			0.93	1402
Macro avg	0.92	0.90	0.91	1402
Weighted avg	0.93	0.93	0.93	1402

Table 9.6: Classification report for SVM.

Class	Precision	Recall	F1-score	Support
0	0.9604	0.9798	0.9700	99
1	0.9799	0.9932	0.9865	147
2	0.9794	0.9500	0.9645	100
3	0.9057	0.9600	0.9320	100
4	0.9167	0.7333	0.8148	15
5	0.9163	0.9811	0.9476	212
6	0.8400	0.6300	0.7200	100
7	0.9175	0.9368	0.9271	190
8	0.9634	0.8316	0.8927	95
9	0.8670	0.9209	0.8932	177
10	0.8764	0.9341	0.9043	167
Accuracy			0.9281	1402
Macro avg	0.9244	0.8995	0.9095	1402
Weighted avg	0.9282	0.9281	0.9267	1402

Table 9.7: Classification report for XGBoost.

Class	Precision	Recall	F1-score	Support
0	0.96	0.87	0.91	99
1	0.94	0.98	0.96	147
2	0.94	0.92	0.93	100
3	0.87	0.89	0.88	100
4	1.00	0.40	0.57	15
5	0.76	0.95	0.84	212
6	0.75	0.12	0.21	100
7	0.76	0.90	0.82	190
8	0.84	0.61	0.71	95
9	0.69	0.81	0.74	177
10	0.79	0.86	0.82	167
Accuracy			0.84	1402
Macro avg	0.86	0.75	0.77	1402
Weighted avg	0.84	0.84	0.82	1402

Table 9.8: Classification report for Random Forest.

Class	Precision	Recall	F1-score	Support
0	1.00	0.92	0.96	99
1	0.94	0.99	0.96	147
2	0.92	0.98	0.95	100
3	0.96	0.91	0.93	100
4	0.82	0.93	0.87	15
5	0.84	0.96	0.90	212
6	0.85	0.52	0.65	100
7	0.96	0.81	0.88	190
8	0.79	0.78	0.78	95
9	0.82	0.86	0.84	177
10	0.70	0.95	0.81	167
Accuracy			0.88	1402
Macro avg	0.88	0.87	0.87	1402
Weighted avg	0.89	0.88	0.88	1402

Table 9.9: Classification report for KNN (K Nearest Neighbour).